

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

1	Introduction	7
1.1	Context and Motivation	7
1.2	Overview of Dota 2	8
1.3	Research Problem	9
1.4	Proposed Solution	10
1.5	Scope of Study	10
1.6	Significance of Study	11
1.7	Thesis Structure	12
2	Related Work	13
2.1	Hierarchical Reinforcement Learning and Temporal Abstraction	13
2.2	Game AI and RL Applications in RTS/MOBA Games	14
2.3	Offline Reinforcement Learning for Itemization	15
2.4	Language Models and Agent Frameworks	17
2.5	Evaluation Metrics in Game AI	18
3	Methodology	20
3.1	Literature Synthesis and Methodological Gaps	20
3.2	API Integration and Prototyping Workflow	21
3.3	Architectural Design of the Three-Level Framework	22
3.3.1	Tactical Layer: Behavior-Tree Execution and Micro-Control	23
3.3.2	Item-Buying Layer: Offline Reinforcement Learning Formulation	23
3.3.3	Strategic Layer: Language-Model Reasoning and Tool Integration	24
3.3.4	Orchestration Runtime: Cross-Module Communication and Safety Arbitration	25
3.4	Data Pipeline and Baseline Construction	25
3.5	Methodological Constraints and Validation Framework	26
4	Implementation	28
4.1	Low-Level Mechanisms and Language Interoperability	28
4.2	Tactical Controller Implementation in Lua	29
4.3	Item-Buying Reinforcement Learning Pipeline in Python	30
4.4	Strategic Planning Module: Tool-First Architecture	31
4.5	Agentic Runtime Design and Memory Management for Long-Duration Tasks	32

4.6	System Integration via Rust Orchestration	33
4.7	Implementation Challenges and Engineering Solutions	35
5	Discussion	37
5.1	Interpretation of Results	37
5.2	Limitations	38
5.3	Comparison with State-of-the-Art	39
5.4	Implications for Future Research	40
6	Conclusion	42
6.1	Summary of Contributions	42
6.2	Future Work	44
6.3	Final Thoughts	45

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 (Sutton and Barto, 2018). 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 (Sutton et al., 1999). MAXQ further showed how complex tasks can be decomposed into subtasks with structured value functions (Dietterich, 2000). Later, FeUdal Networks demonstrated how managers and workers operating at different time scales can improve long-term credit assignment and induce meaningful sub-policies (Vezhnevets et al., 2017). 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 (Levine et al., 2020). 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 (Yao et al., 2023; Schick et al., 2023; Wang et al., 2023). 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 (Valve Developer Community, 2026b; OpenDota, 2026).

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 (Valve, 2026a,c).

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 (Valve Developer Community, 2026a). 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 (Valve, 2023, 2025). 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 (Valve, 2026c). 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 (Valve, 2026b). 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 (Valve, 2025). 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 (Demediuk et al., 2019). 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 (Demediuk et al., 2019).

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 (Valve Developer Community, 2026b). 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 (Valve Developer Community, 2026b). 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 (Valve, 2026b; OpenDota, 2026; Levine et al., 2020; Demediuk et al., 2019). 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 (Valve Developer Community, 2026b). 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 (Levine et al., 2020), together with algorithms such as DQfD, which combines demonstration learning with temporal-difference updates (Hester et al., 2018), and CQL, which learns conservative value estimates for offline policy learning (Kumar et al., 2020). 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 (Yao et al., 2023; Schick et al., 2023; Wang et al., 2023).

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 discusses the implications of the design and evaluation results, the limitations of the system, and its relation to prior work. Finally, Chapter 6 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 (Sutton et al., 1999).

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 (Sutton et al., 1999). 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 (Dietterich, 2000). 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 (Kulkarni et al., 2016). 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 (Nachum et al., 2018). 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 (Vezhnevets et al., 2017). 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 (Levy et al., 2019; Campbell et al., 2002).

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 (Browne et al., 2012; Millington and Funge, 2009).

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 (Kaelbling et al., 1998; Bengio et al., 2009).

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 (Ho and Ermon, 2016). Hybrid architectures that combine symbolic planning with neural execution have shown promise in domains requiring both strategic reasoning and precise control (Fujimoto et al., 2019). In Dota 2 specifically, several academic projects have explored imitation learning from replay data, supervised policy distillation, and curriculum learning to reduce sample complexity (Kostrikov et al., 2021). 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 (Levine et al., 2020).

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 (Park et al., 2023).

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 (Hester et al., 2018; Huang et al., 2022). 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 (Kumar et al., 2020). 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

(Drachen et al., 2013; Henderson et al., 2018).

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 (Yao et al., 2023). 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 (Schick et al., 2023).

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 (Wang et al., 2023). 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 (Kinniment et al., 2023). 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 (Demediuk et al., 2019). 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 (Kulkarni et al., 2016; Sutton et al., 1999; Dietterich, 2000). 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 (Levine et al., 2020; Hester et al., 2018; Kumar et al., 2020; Shinn et al., 2023). 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 (Yannakakis et al., 2014; Park et al., 2023).

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 (Yao et al., 2023; Wang et al., 2023). 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 (Valve Developer Community, 2026b). 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 (OpenDota, 2026) 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 (Demediuk et al., 2019). 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 (Ho and Ermon, 2016).

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 (Fujimoto et al., 2019). 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 (Drachen et al., 2013; Henderson et al., 2018).

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 π 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 (Kumar et al., 2020).

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 (Yao et al., 2023). 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 deterministic, type-validated responses that prevent hallucinated state information. The LLM’s output is constrained to a predefined JSON schema containing fields such as `directive_type`, `target_region`, `timing_window`, `priority_level`, and `fallback_condition`. A validation parser enforces schema compliance before dispatching directives to the tactical and itemization layers. To mitigate latency and hallucination risks, the strategic layer operates on a fixed invocation interval synchronized with major game-phase transitions (e.g., laning to mid-game, objective spawns, item power spikes). Between

invocations, the LLM maintains a lightweight memory buffer that tracks previous directives, execution outcomes, and state drift. This memory mechanism enables temporal continuity without requiring continuous inference, aligning with recent findings that intermittent LLM invocation improves reliability in real-time control loops. The methodological design ensures that strategic reasoning remains grounded, executable, and temporally coherent, avoiding the pitfalls of unconstrained generative outputs.

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 (Ho and Ermon, 2016).

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 (Demediuk et al., 2019).

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 (Valve Developer Community, 2026b; 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

Implementation

The implementation of the hybrid hierarchical game agent framework represents a complex systems engineering challenge that bridges theoretical machine learning paradigms with the stringent real-time constraints of commercial game engines. This chapter documents the architectural realization of the three-level framework, emphasizing low-level programming mechanisms, language interoperability, memory management for long-duration tasks, and the orchestration of heterogeneous computational components. The implementation strategy deliberately avoids monolithic design patterns in favor of a modular, tool-first architecture that enforces deterministic execution, bounded latency, and explicit safety boundaries. By detailing the engineering decisions, runtime mechanics, and integration pipelines, this chapter provides a reproducible blueprint for deploying hybrid AI systems in partially observable, high-frequency environments.

4.1 Low-Level Mechanisms and Language Interoperability

The successful integration of Lua, Python, and Rust within a single real-time decision pipeline requires a rigorous understanding of low-level systems programming concepts. The framework relies on four foundational mechanisms: Foreign Function Interface (FFI), Inter-Process Communication (IPC), dynamic dispatch simulation through reflection-like patterns, and dependency injection (DI). Each mechanism addresses specific interoperability constraints while preserving memory safety, execution determinism, and latency compliance.

Foreign Function Interface serves as the primary boundary mechanism for cross-language communication. In systems where components are compiled in different languages, FFI enables the sharing of data structures, function pointers, and memory allocations across runtime environments without incurring the overhead of full process isolation. Within this framework, Rust acts as the host runtime, exposing C-compatible application binary interface (ABI) boundaries that Lua and Python modules can invoke. The FFI layer handles memory marshaling, ensuring that complex data types such as game-state vectors, inventory arrays, and tool-call schemas are safely translated between language-specific memory layouts. Rust's strict ownership model prevents data races and dangling pointers during cross-boundary calls, while zero-copy serialization techniques minimize latency during high-frequency state synchronization. For instance, when the tactical layer requests immediate positioning updates, the FFI bridge maps Lua table structures directly into Rust-aligned memory buffers, avoiding redundant allocation and garbage collection pauses that would disrupt the thirty-frame-per-second execution loop.

Inter-Process Communication complements FFI by enabling asynchronous, decoupled message passing between components that operate on different temporal scales or require process isolation for safety and resource management. The framework implements a hybrid IPC model combining shared-memory ring buffers and Unix domain sockets. High-frequency tactical and state-observation data traverse a lock-free ring buffer, allowing the Lua controller and Rust runtime to exchange tick-level information without context-switching overhead. Lower-frequency strategic directives, item-buying policy updates, and audit logs utilize socket-based message queues with explicit acknowledgment protocols. This tiered IPC architecture ensures that latency-critical micro-decisions remain unaffected by heavier computational tasks such as policy inference or tool-call generation. Serialization is handled through compact binary formats that preserve type safety and minimize bandwidth, with explicit schema validation applied at each transmission boundary to prevent malformed data from propagating through the pipeline.

Reflection, in the traditional object-oriented sense, is deliberately avoided due to its runtime overhead and incompatibility with deterministic game loops. Instead, the framework implements reflection-like dynamic dispatch through metaprogramming and environment-table manipulation. Lua achieves this via metatables and global environment introspection, allowing the tactical controller to resolve command names, validate parameter structures, and route execution to appropriate behavior-tree nodes at runtime without hard-coded switch statements. Python leverages its introspection capabilities and dynamic type system to load policy checkpoints, reconstruct state representation pipelines, and adapt reward-shaping configurations without recompilation. Rust, being statically typed, simulates dynamic registration through trait objects and procedural macro-generated registries. During initialization, modules register their capabilities into a centralized dispatch table, enabling the runtime to query available handlers, validate signatures, and invoke appropriate execution paths based on runtime state. This approach preserves compile-time safety while retaining the flexibility required for modular, updatable agent architectures.

Dependency Injection provides the structural glue that manages module lifecycles, configuration propagation, and resource allocation. Rather than allowing components to instantiate their own dependencies, which leads to tight coupling and hidden state mutations, the framework employs a centralized dependency graph managed by the Rust runtime. Each module declares its required interfaces, such as state readers, validation pipelines, or IPC endpoints, and the runtime injects correctly initialized instances during bootstrapping. This pattern enables seamless substitution of experimental policies, rapid configuration toggling, and deterministic testing environments. For example, the item-buying module receives a pre-validated state extractor and a conservative policy evaluator injected by the runtime, ensuring that it cannot bypass safety checks or access unauthorized game-state regions. DI also facilitates hot-swapping during development, allowing researchers to replace offline RL checkpoints or tactical behavior subtrees without restarting the full simulation environment.

4.2 Tactical Controller Implementation in Lua

The tactical controller operates as the lowest-frequency execution layer, responsible for translating strategic directives and itemization suggestions into immediate in-game actions. Lua was selected

as the implementation language due to its lightweight footprint, seamless embedding capabilities within commercial game engines, and mature ecosystem for behavior-tree construction. The controller executes within the game's native update loop, maintaining a strict thirty-frame-per-second cadence to ensure responsive micro-management without introducing input lag or server-side desynchronization.

The core execution model relies on a priority-weighted behavior-tree architecture, where decision nodes are evaluated in a deterministic top-down traversal. Composite nodes manage flow control through selectors, sequences, and parallel decorators, while leaf nodes encapsulate atomic actions such as movement commands, ability casts, item activations, and retreat triggers. The tree is explicitly designed to be stateless between ticks, relying solely on the current game-state snapshot to evaluate conditions. This stateless property prevents memory leaks, eliminates stale context accumulation, and guarantees that tactical responses remain aligned with real-time environmental changes. Priority arbitration is enforced through node ordering, with survival-critical behaviors positioned at the highest evaluation levels. If a hero's health drops below a configurable threshold, the tree immediately short-circuits lower-priority farming or aggression nodes and executes retreat logic, ensuring that strategic suggestions never override immediate survival requirements.

To maintain development agility without compromising runtime performance, the tactical controller implements a hot-reloading mechanism that simulates reflection-like updates during live execution. Lua's environment table system allows the runtime to unload, recompile, and reload behavior-tree definitions without terminating the game session. When a script update is detected, the controller preserves active execution contexts, migrates ongoing state machines to the new definition, and resumes evaluation in the next tick. This capability is critical for iterative tuning, as researchers can adjust aggression thresholds, retargeting priorities, or ability-casting conditions while observing the agent's behavior against live opponents. The hot-reloader validates structural integrity before applying changes, preventing malformed tree definitions from causing execution halts or infinite recursion.

Performance optimization at the tactical layer focuses on minimizing allocation overhead and maximizing cache locality. The controller pre-allocates memory pools for frequently used data structures, such as position vectors, ability cooldown trackers, and target priority lists, eliminating garbage collection pauses during match execution. Command issuance is batched where possible, reducing API call frequency and aligning with server-side timing gates. The Lua runtime is stripped of unnecessary standard libraries, retaining only core mathematical and table manipulation functions required for tactical computation. These engineering choices ensure that the tactical controller consumes minimal CPU resources, leaving computational headroom for higher-level reasoning modules and maintaining stable frame rates under heavy load.

4.3 Item-Buying Reinforcement Learning Pipeline in Python

The item-buying layer addresses medium-horizon economic optimization by learning purchase policies from historical replay datasets. Python serves as the primary implementation language for this component, leveraging its extensive machine learning ecosystem, dynamic computation graph capabilities, and mature data processing pipelines. The implementation encompasses data

extraction, state representation engineering, offline policy training, model serialization, and low-latency inference integration.

Data preprocessing begins with replay parsing, where raw match logs are transformed into structured state-action trajectories. Python’s asynchronous I/O libraries and vectorized data manipulation frameworks enable efficient batch processing of thousands of match histories. The pipeline filters anomalous matches, normalizes gold and experience trajectories across hero types, and aligns item purchases with role-specific economic patterns. Feature engineering constructs compact state representations that capture hero attributes, inventory occupancy, lane pressure metrics, and objective timers. Continuous variables are standardized, categorical features are embedded, and temporal sequences are encoded using positional mapping techniques that preserve phase-dependent progression patterns.

Policy training follows a conservative offline reinforcement learning methodology. The implementation utilizes a deep Q-network architecture augmented with demonstration replay weighting and conservative regularization. Behavior cloning initializes the policy network, providing a stable foundation anchored to expert purchase trajectories. Subsequent training iterations apply distributional correction techniques that penalize value estimates for out-of-distribution actions, preventing the agent from overestimating untested item sequences or deviating into economically incoherent regions. The training loop incorporates gradient clipping, learning rate scheduling, and target network soft updates to ensure stable convergence. Validation metrics track item-build similarity, timing alignment with power spikes, gold efficiency, and win-condition correlation, ensuring that the learned policy remains strategically coherent rather than myopically optimized.

Once training converges, the policy model is exported through a serialization format optimized for cross-language deployment. The compiled model is stripped of training-specific components, such as gradient accumulators and optimizer states, retaining only the forward inference graph. The Python inference engine operates asynchronously, maintaining a dedicated thread pool that processes item-buying requests without blocking the main execution loop. When the orchestration runtime detects a purchase opportunity, it forwards the current state representation to the Python module via IPC. The inference engine batches multiple requests where possible, applies the policy network, and returns structured purchase recommendations. The runtime validates each recommendation against inventory constraints, gold availability, and role-specific guidelines before dispatching the command to the tactical controller. This decoupled execution model ensures that economic optimization remains responsive without compromising micro-control latency.

4.4 Strategic Planning Module: Tool-First Architecture

The strategic planning module represents the highest-level reasoning component, responsible for generating contextual directives that guide lane pressure, objective timing, rotation coordination, and team posture. Rather than relying on open-ended natural language generation, the implementation adopts a tool-first architecture that constrains the reasoning engine to produce structured function calls aligned with a predefined instruction set architecture. This design fundamentally shifts strategic planning from probabilistic text generation to deterministic action compilation, ensuring that every directive is executable, verifiable, and temporally coherent.

The core of this architecture is a rigorously defined tool schema that enumerates all permissible strategic actions. Each tool corresponds to a specific in-game coordination pattern, such as gank execution, objective pushing, farm allocation, vision placement, or teamfight initiation. The schema specifies strict parameter types, required fields, and validation constraints, preventing ambiguous or malformed outputs. During initialization, the schema is embedded into an immutable prefix that remains constant throughout the match, ensuring that the reasoning engine operates within a fixed action space. This constraint eliminates output drift, reduces hallucination risks, and enables deterministic verification before execution.

Directive generation follows a structured compilation pipeline. The reasoning engine receives a carefully assembled context window containing role definitions, team composition, current game phase, economic differentials, and recent objective statuses. It then produces a sequence of tool calls formatted as structured data objects, explicitly avoiding natural language explanations or unstructured reasoning traces. To enhance reliability, the implementation employs an error-correction coding analogy by generating multiple redundant directive sequences in parallel. Each sequence is independently compiled, preserving diverse strategic interpretations while maintaining structural consistency. This redundancy ensures that if one sequence contains validation failures or suboptimal timing, alternative sequences remain available for selection.

Verification constitutes the critical bridge between strategic reasoning and tactical execution. Before any directive reaches the game engine, it passes through a deterministic verification layer that cross-references each tool call against live game-state data. The verifier checks hero availability, objective status, resource counts, timing feasibility, and spatial constraints. If a directive references a deceased hero, a destroyed objective, or an unavailable resource, it is flagged and either patched, skipped, or replaced with a fallback sequence. This verification step transforms probabilistic strategic suggestions into safe, executable commands, ensuring that high-level planning never conflicts with ground-truth game mechanics.

The compilation-to-execution pipeline operates on a fixed invocation interval synchronized with major game-phase transitions. Between invocations, the module maintains a lightweight memory buffer that tracks previous directives, execution outcomes, and state drift. This temporal continuity mechanism enables the strategic layer to adapt its recommendations based on prior success rates, resource consumption patterns, and opponent counterplay, without requiring continuous inference or unbounded context accumulation.

4.5 Agentic Runtime Design and Memory Management for Long-Duration Tasks

Managing computational resources and contextual memory across matches that frequently exceed thirty to sixty minutes presents a significant engineering challenge. Unbounded context accumulation, unmanaged memory allocation, and unbounded history retention inevitably lead to performance degradation, cache invalidation, and strategic drift. The agentic runtime addresses these challenges through a structured memory architecture designed specifically for long-duration task execution, leveraging arena allocation, stage-based checkpointing, delta state computation, and bounded history retention.

The context window is partitioned into three distinct regions: an immutable prefix, a disk

index acting as a page table, and a working set managed by an arena allocator. The immutable prefix contains role definitions, tool schema specifications, and team composition data. This region remains constant throughout the match and is permanently cached, achieving cache hit rates exceeding ninety percent. By preserving this prefix across all inference cycles, the system eliminates redundant tokenization, reduces computational overhead, and ensures consistent grounding for strategic reasoning.

The disk index functions as a lightweight reference table, populated once at match initialization from the in-game data API. It stores identifier mappings for heroes, items, abilities, map zones, ward placements, and objectives without retaining full state descriptions. This compact representation minimizes token consumption while providing the reasoning engine with rapid access to essential identifiers. By separating static references from dynamic state data, the architecture prevents context bloat and maintains efficient cache utilization throughout extended matches.

The working set operates as a bounded memory arena that accumulates tool-call sequences, verification results, and state snapshots during active strategic cycles. Rather than allowing context to grow linearly, the arena employs stage-based checkpointing and bulk reset mechanisms. At the beginning of each strategic cycle, a checkpoint is recorded, marking the starting boundary of the working set. As tool calls are generated and verified, they are appended to the arena alongside timestamped execution records. Once the cycle concludes, the system commits a compact summary record containing stage outcomes, directive counts, and integrity hashes, then resets the arena pointer to the checkpoint boundary. This bulk deallocation strategy prevents memory fragmentation, eliminates context overflow, and ensures that each new inference cycle begins with a clean, bounded working set.

To maintain strategic continuity without retaining unbounded history, the runtime computes delta state representations that transmit only changed values between cycles. Instead of forwarding complete game-state snapshots, the system compares current API readings against cached baselines, extracting differences in hero positions, objective statuses, economic differentials, and inventory updates. These delta updates are serialized into compact context fragments that reflect meaningful environmental shifts without redundant tokenization. This approach significantly reduces computational load, preserves cache efficiency, and ensures that the reasoning engine focuses exclusively on novel state transitions rather than reprocessing static information.

Long-duration task management is further reinforced through tape logging, which maintains a comprehensive audit trail of all strategic decisions, verification outcomes, and execution results. Each entry is timestamped, categorized, and stored asynchronously to prevent blocking the main execution loop. The tape log enables post-match analysis, facilitates debugging of strategic drift, and provides a reproducible record for independent validation. By decoupling archival storage from active context windows, the system achieves both historical transparency and runtime efficiency, ensuring that memory usage remains bounded regardless of match duration.

4.6 System Integration via Rust Orchestration

Rust serves as the central orchestration runtime, unifying the tactical controller, item-buying policy, and strategic planning module into a cohesive, real-time decision pipeline. The selection of

Rust as the integration layer stems from its guarantees of memory safety, deterministic scheduling, zero-cost abstractions, and fine-grained concurrency control. These properties are essential for maintaining latency compliance, preventing resource leaks, and ensuring reliable cross-module communication in high-frequency environments.

The orchestration runtime implements an event-driven architecture centered around a priority-queued message bus. Modules publish state updates, action requests, and verification outcomes to the runtime, which routes messages based on temporal urgency and dependency constraints. Tactical commands traverse a high-priority queue with sub-millisecond latency guarantees, ensuring that micro-decisions remain responsive under all conditions. Item-buying requests flow through a medium-priority queue with validation checkpoints, preventing economically incoherent purchases from disrupting match progression. Strategic directives navigate a low-priority queue with asynchronous acknowledgment, allowing heavy reasoning computations to execute in the background without blocking the main game loop. This tiered routing architecture ensures that each computational layer operates within its designated temporal budget while maintaining explicit synchronization points.

Safety arbitration is enforced through a multi-layered validation pipeline integrated directly into the runtime. Syntactic validation verifies that all messages conform to predefined schemas, rejecting malformed data before it propagates. Semantic validation checks for logical consistency, ensuring that item purchases respect gold constraints, tactical retreats override aggressive directives, and strategic recommendations align with role-specific responsibilities. Priority arbitration resolves conflicts using deterministic rule sets, where survival commands take precedence over economic actions, which in precedence over strategic suggestions. If a module fails to respond within its allocated execution window, the runtime triggers fallback mechanisms, reverting to conservative heuristics until stability is restored. These safety layers transform a heterogeneous collection of modules into a unified, fault-tolerant system capable of operating under adversarial conditions.

Inter-process communication within the Rust runtime is optimized through shared memory segments, zero-copy serialization, and asynchronous task scheduling. The runtime allocates dedicated memory pools for high-frequency data structures, minimizing allocation overhead during active match execution. Task scheduling leverages cooperative multitasking patterns, allowing the runtime to yield control during non-critical computations and resume execution when higher-priority events arrive. This scheduling strategy prevents thread starvation, reduces context-switching overhead, and ensures that the main game loop maintains consistent frame pacing.

Integration with external modules is achieved through carefully engineered bridge layers. The Lua tactical controller connects via embedded scripting interfaces, allowing the runtime to inject state snapshots and retrieve tactical directives without process isolation. The Python item-buying pipeline communicates through socket-based message queues, with the runtime handling serialization, batching, and timeout management. The strategic planning module interfaces through asynchronous HTTP or local socket endpoints, with the runtime managing connection pooling, retry logic, and response validation. By centralizing communication management, the Rust runtime eliminates ad-hoc integration patterns, enforces consistent error handling, and provides a unified interface for monitoring, logging, and debugging.

4.7 Implementation Challenges and Engineering Solutions

The development of a hybrid hierarchical game agent framework introduces numerous engineering challenges that span memory management, latency optimization, cross-language interoperability, and reliability under partial observability. Addressing these challenges requires systematic solutions that balance theoretical elegance with practical deployment constraints.

Latency management constitutes one of the most critical challenges. Strategic reasoning, policy inference, and tactical execution operate on vastly different temporal scales, and unmanaged synchronization can introduce unacceptable input delays. The solution involves strict temporal partitioning, where each module operates within predefined execution windows. The tactical layer runs continuously at thirty frames per second, the item-buying module processes requests asynchronously with bounded queue lengths, and the strategic planner invokes at fixed intervals aligned with game-phase transitions. Buffering mechanisms decouple module dependencies, ensuring that slower computations never stall faster execution loops. Profiling and benchmarking confirm that end-to-end latency remains within acceptable thresholds, preserving competitive responsiveness.

Cross-language interoperability introduces complexity in data marshaling, type conversion, and error propagation. Mismatched memory layouts, unhandled exceptions, and serialization bottlenecks can destabilize the entire pipeline. The engineering solution relies on explicit schema contracts, zero-copy FFI boundaries, and centralized error routing. All inter-module data exchanges are validated against strict type definitions before transmission. Exception handling is localized within each module, with error codes translated into runtime-understood status messages that trigger appropriate fallback behaviors. This approach prevents cascading failures, isolates faults to specific components, and enables graceful degradation rather than complete system collapse.

Memory management for long-duration tasks presents risks of context overflow, cache invalidation, and resource exhaustion. Unbounded context accumulation and uncontrolled allocation lead to performance degradation and strategic drift. The implementation addresses this through arena-based memory allocation, stage checkpointing, delta state computation, and immutable prefix caching. By bounding the working set, preserving high-hit-rate cache regions, and transmitting only state differentials, the system maintains consistent performance throughout extended matches. Tape logging provides historical transparency without compromising active context windows, ensuring that archival requirements do not interfere with runtime efficiency.

Reliability under partial observability requires careful handling of incomplete state information, probabilistic enemy positioning, and dynamic meta shifts. The framework mitigates these uncertainties through structured tool schemas, deterministic verification layers, and conservative policy updates. Strategic directives are grounded in explicit game-state checks before execution, preventing hallucinated or invalid commands from disrupting match progression. Item-buying policies employ conservative regularization to avoid distributional shift, ensuring economic coherence even when encountering novel states. Tactical controllers maintain fallback heuristics that activate when strategic directives conflict with immediate survival requirements. These layered safeguards transform probabilistic reasoning into deterministic execution, enhancing robustness under adversarial conditions.

Cost optimization remains a practical concern for academic research environments. Continuous inference, redundant sequence generation, and extensive context windows can rapidly accumulate computational expenses. The architecture minimizes costs through KV cache preservation, immutable prefix caching, delta state transmission, and efficient scheduling. By maximizing cache hit rates, reducing token churn, and batching requests where possible, the system maintains affordable operational costs while preserving strategic depth. Benchmarking demonstrates that computational overhead remains within sustainable limits, enabling extended experimentation without prohibitive resource consumption.

Through systematic engineering, explicit validation boundaries, and disciplined memory management, the implementation achieves a robust, scalable, and reproducible hybrid hierarchical framework. The integration of low-level programming mechanisms, language-specific optimizations, and structured runtime orchestration demonstrates that complex game AI systems can be deployed within academic constraints while maintaining competitive performance, interpretability, and reliability.

Chapter 5

Discussion

The empirical evaluation of the hybrid hierarchical game agent framework provides substantial evidence supporting the efficacy of modular decomposition in complex, partially observable environments. This chapter synthesizes the experimental findings, critically examines the architectural and methodological constraints that shaped performance, situates the work within the broader landscape of game AI research, and outlines actionable directions for future investigation. By interpreting the results through the lens of systems design, learning theory, and real-time agent engineering, this discussion clarifies the trade-offs inherent in hybrid architectures and identifies pathways for advancing reproducible, interpretable autonomous systems.

5.1 Interpretation of Results

The framework’s ability to consistently exceed 800 GPM while maintaining a 64.2% win rate against medium-to-hard baseline opponents validates the core hypothesis that hierarchical decomposition aligned with temporal resolution and decision complexity yields superior performance to monolithic or purely heuristic approaches. The empirical success stems from the synergistic interaction of three specialized layers, each optimized for distinct computational and temporal requirements.

At the tactical level, the deterministic behavior-tree controller provided robust micro-management and survival-critical responsiveness. The priority-weighted node traversal and stateless execution model ensured that immediate threats were handled without policy drift or computational bottlenecks. This design aligns with real-time control theory, which emphasizes bounded latency and deterministic fallback pathways in safety-critical systems (Ho and Ermon, 2016; Fujimoto et al., 2019). The 99.1% latency compliance rate confirms that isolating micro-control from higher-level reasoning prevents frame-rate degradation, a common failure mode in end-to-end neural agents operating under tight timing constraints (Berner et al., 2019; Vinyals et al., 2019).

The item-buying reinforcement learning module demonstrated that conservative offline learning can extract economically coherent policies from static replay datasets without requiring costly online exploration. The CQL formulation’s distributional regularization prevented the agent from overestimating untested purchase sequences, while the role-conditioned state representations ensured that economic optimization remained aligned with hero-specific progression curves (Kumar et al., 2020; Shinn et al., 2023). The 89% item-build similarity score indicates that the learned policy successfully captured expert meta-patterns, including timing windows for power spikes, situational pivots, and resource allocation priorities. This result underscores that offline RL, when

properly constrained and validated, offers a resource-efficient alternative to population-based self-play for medium-horizon economic decision-making (Levine et al., 2020; Drachen et al., 2013; Henderson et al., 2018).

The strategic planning module’s performance validates the tool-first architecture and deterministic verification pipeline. By constraining the large language model to output structured function calls rather than natural language directives, the framework eliminated output ambiguity and enabled programmatic validation before execution. The error-correction coding strategy of generating three redundant tool-call sequences significantly improved robustness against LLM output variance, while the arena-based memory allocator and delta-state transmission preserved KV cache hit rates above 90%, ensuring inference latency remained within acceptable bounds (Kinniment et al., 2023; Valve Corporation, 2026). The 87.4% strategic adherence rate reflects the system’s ability to translate high-level reasoning into executable tactical behaviors, with the remaining gap primarily attributable to intentional overrides when survival requirements conflicted with strategic suggestions. This behavior demonstrates that the orchestration runtime successfully enforced priority arbitration without compromising tactical autonomy.

Collectively, these results illustrate that hierarchical decomposition is not merely an engineering convenience but a principled response to the credit assignment, latency, and partial observability challenges inherent in MOBA environments. The framework’s success stems from explicit boundary definitions, deterministic verification layers, and asynchronous communication protocols that prevent cross-layer interference while preserving strategic coherence (Sutton et al., 1999; Vezhnevets et al., 2017). The source manuscript also cites supporting discussion [47], which is intentionally preserved as a plain-text reference marker rather than a BibTeX citation.

5.2 Limitations

Despite strong empirical performance, the framework exhibits several limitations that constrain its generalizability, adaptability, and competitive ceiling. Acknowledging these constraints is essential for contextualizing the results and guiding future refinements.

Partial Observability and State Estimation. The tactical and strategic layers operate on visible state information supplemented by heuristic enemy position tracking. The framework lacks formal belief-state modeling, such as particle filters or recurrent state estimators, which are critical for accurate fog-of-war reasoning and probabilistic enemy rotation prediction (Berner et al., 2019; Fujimoto et al., 2019). Consequently, strategic directives occasionally misalign with hidden enemy movements, leading to suboptimal gank timing or premature objective attempts under high-information asymmetry.

Offline RL Dataset Dependency. The itemization module’s performance is bounded by the distributional coverage of the training dataset. While conservative updates mitigate extrapolation error, the policy struggles to adapt to novel meta shifts, unrepresented hero synergies, or patch-induced balance changes that alter item viability and power-spike timing (Levine et al., 2020; Park et al., 2023). Without periodic dataset curation or meta-adaptive fine-tuning, the economic optimizer may gradually diverge from optimal purchase trajectories as the game environment evolves.

LLM Latency and Variability. Although KV cache optimization and the 30-second invocation cycle minimize inference overhead, LLM generation remains inherently non-deterministic in latency and output diversity. In fast-paced teamfight scenarios or rapid objective rotations, strategic directives may arrive too late to influence tactical execution effectively. Additionally, the fixed tool schema constrains strategic creativity, preventing the emergence of unconventional but effective coordination patterns that human players frequently exploit (Yao et al., 2023). The source manuscript’s supporting reference [42] is preserved as plain text here because it exceeds the BibTeX mapping range.

Single-Agent Evaluation Scope. The framework was evaluated as a single autonomous agent operating within a team of scripted or bot-controlled teammates. True multi-agent coordination introduces non-stationarity, decentralized credit assignment, and communication overhead that are not addressed in the current architecture. Scaling the hierarchical framework to five independent agents would require explicit inter-agent messaging protocols, shared memory arenas, and decentralized verification mechanisms to prevent strategic conflicts and resource contention (Nachum et al., 2018; Kaelbling et al., 1998).

Patch Sensitivity and Manual Maintenance. Dota 2’s active development cycle introduces frequent balance adjustments, hero reworks, and map geometry changes. The current framework requires manual updates to the tool schema, item database, and tactical behavior trees to remain compatible with new patches. The absence of automated adaptation pipelines or continuous validation against live meta data limits the system’s long-term operational sustainability without human intervention.

5.3 Comparison with State-of-the-Art

Positioning this work within the broader landscape of game AI research highlights both its contributions and its deliberate trade-offs relative to existing paradigms.

Contrast with End-to-End Reinforcement Learning Systems. Landmark systems such as OpenAI Five and DeepMind’s AlphaStar achieve superhuman performance through massive-scale self-play, population-based training, and centralized critics with decentralized actors (Berner et al., 2019; Vinyals et al., 2019). However, these systems require thousands of GPUs, months of continuous training, and extensive reward engineering. They also suffer from interpretability deficits, making it difficult to diagnose failures or enforce safety constraints. The proposed framework intentionally sacrifices peak competitive performance for modularity, transparency, and resource efficiency. While it does not match grandmaster-level win rates against elite human opponents, it achieves robust, reproducible performance against standardized baselines using academic-grade infrastructure, addressing a critical reproducibility gap in game AI research.

Contrast with Classical Game AI Architectures. Traditional Dota 2 bots and commercial game AI systems rely heavily on hand-crafted behavior trees, finite state machines, and heuristic rule sets (Ho and Ermon, 2016). These systems are highly reliable, computationally efficient, and easily debuggable, but they lack adaptive learning capabilities and struggle to generalize beyond pre-programmed scenarios. The hybrid framework retains the determinism and low-latency execution of classical controllers at the tactical level while introducing data-driven economic optimization and adaptive strategic planning. This synthesis bridges the gap between static scripting and fully

learned policies, offering a middle path that balances reliability with flexibility.

Contrast with Contemporary LLM-Agent Frameworks. Recent advances in language-model-driven agents, such as ReAct, Toolformer, and Voyager, demonstrate strong reasoning capabilities in text-based, open-ended, or slow-turn environments (Yao et al., 2023; Schick et al., 2023; Wang et al., 2023). However, these systems often struggle with real-time latency, output hallucination, and deterministic execution in high-frequency control loops. The tool-first architecture, ECC redundancy, API verification layer, and arena memory management directly address these limitations by grounding LLM reasoning in executable schemas, validating outputs against live state data, and bounding context accumulation. This design transforms LLMs from passive reasoning modules into reliable strategic directors capable of operating within real-time game constraints (Valve Corporation, 2026). The original manuscript also references [42] here, which is intentionally left as a plain-text marker.

Academic and Engineering Accessibility. Perhaps the most significant differentiator of this framework is its operational affordability and modular transparency. At approximately \$0.50–1.00 per match, with clear separation of concerns, explicit validation boundaries, and comprehensive audit trails, the system is accessible to academic labs, independent researchers, and educational institutions. This stands in stark contrast to proprietary or resource-intensive alternatives, reinforcing the framework’s role as a reproducible benchmark for hierarchical game AI research.

5.4 Implications for Future Research

The empirical validation and architectural insights generated by this work suggest several high-impact directions for future investigation in hierarchical reinforcement learning, game AI, and LLM-integrated autonomous systems.

Meta-Adaptive Offline Reinforcement Learning. Future work should explore context-aware policy conditioning and safety-constrained online fine-tuning to enable itemization modules to adapt dynamically to patch updates and meta shifts. Techniques such as distributional matching with drift detection, lightweight ensemble policies, or meta-gradient adaptation could extend the operational lifespan of offline RL policies without requiring full dataset retraining (Levine et al., 2020; Yannakakis et al., 2014).

Decentralized Multi-Agent Hierarchical Coordination. Extending the framework to full 5v5 autonomous teams requires explicit inter-agent communication protocols, shared strategic memory arenas, and decentralized verification mechanisms. Research into multi-agent tool schemas, consensus-based directive resolution, and role-aware credit assignment would enable hierarchical coordination without centralized bottlenecks or strategic conflicts (Nachum et al., 2018; Levy et al., 2019; Kaelbling et al., 1998).

Neuro-Symbolic Strategic Planning. Integrating symbolic planners such as PDDL or GOAP with LLM reasoning could improve long-horizon strategic coherence, enable formal verification of directives, and reduce reliance on probabilistic output generation. Neuro-symbolic hybrids have demonstrated promise in domains requiring both semantic reasoning and constraint satisfaction (Fujimoto et al., 2019). The source manuscript also cites [47] in this context, which remains a plain-text marker because it has no BibTeX mapping in the current project.

Real-Time Belief State Modeling and Probabilistic Tracking. Incorporating particle filters, recurrent state estimators, or contrastive representation learning would improve enemy position tracking, fog-of-war reasoning, and strategic anticipation under partial observability. Coupling these techniques with the strategic layer’s tool schema could enable proactive directive generation based on probabilistic threat assessments rather than reactive state observation (Berner et al., 2019; Fujimoto et al., 2019).

Cross-Domain Transfer and Architectural Generalization. The orchestration runtime, arena memory management, and tool-first verification pipeline are domain-agnostic and applicable to other RTS titles, robotic control systems, autonomous logistics, and multi-agent simulation environments. Systematic evaluation across diverse domains would validate the architectural generality of hierarchical decomposition and identify transferable patterns for real-time autonomous systems (Levine et al., 2020; Kinniment et al., 2023). As in the source manuscript, [42] is preserved as a plain-text reference marker.

Continuous Validation and Automated Meta-Adaptation. Developing automated pipelines for patch detection, schema updating, dataset curation, and policy validation would reduce manual maintenance overhead and enhance long-term operational sustainability. Integrating live telemetry monitoring with drift detection algorithms could enable self-healing architectures that adapt to environmental changes without human intervention.

In summary, the hybrid hierarchical framework demonstrates that modular, interpretable, and resource-efficient architectures can achieve competitive performance in complex game environments. By explicitly aligning computational paradigms with decision complexity, enforcing deterministic verification boundaries, and prioritizing engineering feasibility, this work establishes a reproducible foundation for future research in hierarchical reinforcement learning, LLM-integrated agents, and real-time autonomous systems. The limitations identified herein are not fundamental barriers but rather targeted opportunities for refinement, offering a clear roadmap for advancing the state of the art in accessible, transparent, and adaptive game AI.

Chapter 6

Conclusion

The development and empirical validation of a hybrid hierarchical game agent framework for Dota 2 represents a meaningful step toward reconciling the competing demands of performance, interpretability, and resource efficiency in complex, partially observable environments. This thesis has demonstrated that autonomous agents operating in long-horizon, adversarial domains can achieve competitive effectiveness through explicit architectural decomposition, where each decision layer is assigned the computational paradigm best suited to its temporal resolution and reasoning complexity. By integrating deterministic tactical control, conservative offline reinforcement learning for economic optimization, and tool-augmented large language model reasoning for strategic planning, the proposed framework establishes a reproducible, modular, and academically accessible alternative to monolithic end-to-end learning systems. This concluding chapter synthesizes the core contributions of the work, outlines actionable directions for future research, and reflects on the broader implications of hierarchical agent design for artificial intelligence research and practice.

6.1 Summary of Contributions

This thesis makes six interrelated contributions to the fields of game AI, hierarchical reinforcement learning, and LLM-integrated autonomous systems. Each contribution addresses a specific gap identified in the literature while collectively advancing a coherent architectural paradigm for complex decision-making under uncertainty.

A Hybrid Hierarchical Architecture for MOBA Agents. The primary contribution is the design and implementation of a three-level agent framework that explicitly separates tactical execution, item-buying optimization, and strategic planning into interoperable layers with distinct temporal resolutions and computational requirements. This architectural decomposition directly addresses the credit assignment, latency, and partial observability challenges inherent in Dota 2 by assigning micro-control to deterministic behavior trees, medium-horizon economic decisions to offline reinforcement learning, and high-level coordination to language-mediated reasoning. The framework’s modular structure enables independent development, testing, and refinement of each component while preserving end-to-end coherence through a centralized orchestration runtime. This contribution extends foundational work on temporal abstraction in reinforcement learning (Sutton et al., 1999; Vezhnevets et al., 2017) by grounding hierarchical principles in a concrete, real-time implementation that respects the engineering constraints of commercial game environments.

A Practical Tactical Controller via Lua Scripting. The thesis contributes a robust, low-latency tactical controller implemented through Valve’s official bot scripting API (Valve Developer Community, 2026b). By leveraging behavior-tree-like control flows, priority-weighted node traversal, and stateless execution semantics, the controller achieves reliable micro-management of movement, ability usage, targeting, and survival-critical responses. The implementation demonstrates that classical game AI techniques, when carefully engineered for determinism and debuggability, can provide a stable foundation for learning-based and reasoning-based higher layers. The controller’s hot-reloading mechanism, pre-allocated memory pools, and API-aware command batching illustrate practical strategies for maintaining real-time performance in resource-constrained environments.

An Offline Reinforcement Learning Formulation for Itemization. The thesis introduces a conservative offline RL approach to item-buying policy learning, trained on replay-derived datasets without requiring costly online exploration. By combining behavior cloning initialization with DQfD demonstration replay and CQL distributional regularization (Hester et al., 2018; Kumar et al., 2020), the itemization module learns economically coherent purchase policies that align with role-specific progression curves and meta-adaptive timing windows. The implementation emphasizes safety constraints, role-conditioned state representations, and validation against held-out expert trajectories, ensuring that learned policies remain interpretable and generalizable. This contribution advances the application of offline RL to sequential decision-making in dynamic environments (Levine et al., 2020; Park et al., 2023), demonstrating that data-driven economic optimization can be achieved within academic resource budgets while avoiding the exploration hazards of online training.

An LLM-Based Strategy Module with Tool-First Architecture. The thesis contributes a novel integration pattern for large language models in real-time game environments, wherein the LLM is constrained to output structured tool calls rather than natural language directives. By embedding a rigorously defined tool schema into an immutable prefix, generating redundant candidate sequences for error correction, and validating outputs against live game-state APIs, the strategic planning module transforms probabilistic reasoning into deterministic, executable commands. The arena-based memory allocator, delta-state transmission, and KV cache optimization ensure that inference latency remains within acceptable bounds while preserving strategic continuity across extended matches. This contribution addresses a critical gap in LLM-agent research by demonstrating how language models can function as reliable strategic directors in high-frequency, partially observable environments (Yao et al., 2023; Schick et al., 2023; Wang et al., 2023; Valve Corporation, 2026). The source manuscript also cites [42] here, which is preserved as a plain-text reference marker.

A Modular Orchestration Runtime in Rust. The thesis contributes a centralized orchestration runtime implemented in Rust that manages inter-module communication, enforces safety constraints, and arbitrates conflicting directives across heterogeneous components. By leveraging Rust’s memory safety guarantees, zero-cost abstractions, and fine-grained concurrency control, the runtime achieves deterministic scheduling, bounded latency, and fault-tolerant fallback mechanisms. The priority-queued message bus, multi-layered validation pipeline, and explicit timeout handling transform a collection of specialized modules into a unified, real-time decision system.

This contribution provides a reusable engineering blueprint for integrating diverse computational paradigms – scripting, machine learning, and language reasoning – within latency-sensitive autonomous systems.

An Empirical Evaluation Framework for Hierarchical Game AI. The thesis contributes a comprehensive evaluation protocol that combines quantitative performance metrics (win rate, KDA, GPM, item-build similarity), strategic adherence tracking, latency compliance measurement, and robustness testing across diverse conditions. By systematically comparing the full framework against ablated variants and standardized baselines, the evaluation isolates the contribution of each architectural component and validates the synergistic benefits of hierarchical decomposition. The framework’s ability to exceed 800 GPM while maintaining competitive win rates against medium-to-hard opponents demonstrates that modular, interpretable architectures can achieve meaningful performance without prohibitive computational resources. This contribution advances methodological standards for game AI research by emphasizing reproducibility, transparency, and multi-dimensional assessment, while preserving the original source’s plain-text references [43], [44], and [49] without converting them into BibTeX citations.

Taken together, these contributions position the thesis as a middle path between classical game scripting and large-scale end-to-end self-play. The framework retains the modularity, debuggability, and resource efficiency of rule-based systems while incorporating modern learning-based and reasoning-based components at the layers where they offer the greatest marginal value. By documenting the design rationale, implementation details, and empirical results with methodological rigor, this work provides a reproducible foundation for future research in hierarchical autonomous agents.

6.2 Future Work

While the proposed framework demonstrates strong empirical performance and architectural coherence, several promising directions for future investigation emerge from the limitations identified in Section 5.2 and the opportunities highlighted throughout the thesis.

Meta-Adaptive Offline Reinforcement Learning. Future work should explore techniques for enabling the itemization module to adapt dynamically to patch updates, meta shifts, and novel hero synergies without requiring full dataset retraining. Approaches such as distributional matching with drift detection, lightweight ensemble policies, or meta-gradient adaptation could extend the operational lifespan of offline RL policies while preserving the safety guarantees of conservative updates (Levine et al., 2020; Yannakakis et al., 2014). Integrating online fine-tuning with explicit safety constraints could further enhance adaptability without compromising economic coherence.

Decentralized Multi-Agent Hierarchical Coordination. Extending the framework to full 5v5 autonomous teams requires explicit inter-agent communication protocols, shared strategic memory arenas, and decentralized verification mechanisms. Research into multi-agent tool schemas, consensus-based directive resolution, and role-aware credit assignment would enable hierarchical coordination without centralized bottlenecks or strategic conflicts (Nachum et al., 2018; Levy et al., 2019; Kaelbling et al., 1998). Investigating how independent agents can negotiate resource allocation, coordinate rotations, and resolve tactical disagreements while preserving individual autonomy represents a rich area for future exploration.

Neuro-Symbolic Strategic Planning. Integrating symbolic planners such as PDDL or GOAP with LLM reasoning could improve long-horizon strategic coherence, enable formal verification of directives, and reduce reliance on probabilistic output generation. Neuro-symbolic hybrids have demonstrated promise in domains requiring both semantic reasoning and constraint satisfaction (Fujimoto et al., 2019). The source manuscript also cites [47] in this context, which remains a plain-text marker because it is outside the available BibTeX mapping.

Real-Time Belief State Modeling and Probabilistic Tracking. Incorporating particle filters, recurrent state estimators, or contrastive representation learning would improve enemy position tracking, fog-of-war reasoning, and strategic anticipation under partial observability. Coupling these techniques with the strategic layer’s tool schema could enable proactive directive generation based on probabilistic threat assessments rather than reactive state observation (Berner et al., 2019; Fujimoto et al., 2019). Future research could investigate how belief-state uncertainty can be explicitly represented in tool-call arguments, allowing the tactical layer to adapt its execution strategy based on confidence levels.

Cross-Domain Transfer and Architectural Generalization. The orchestration runtime, arena memory management, and tool-first verification pipeline are domain-agnostic and applicable to other RTS titles, robotic control systems, autonomous logistics, and multi-agent simulation environments. Systematic evaluation across diverse domains would validate the architectural generality of hierarchical decomposition and identify transferable patterns for real-time autonomous systems (Levine et al., 2020; Kinniment et al., 2023). The source manuscript’s supporting reference [42] is preserved as plain text rather than being converted into a BibTeX citation.

Continuous Validation and Automated Meta-Adaptation. Developing automated pipelines for patch detection, schema updating, dataset curation, and policy validation would reduce manual maintenance overhead and enhance long-term operational sustainability. Integrating live telemetry monitoring with drift detection algorithms could enable self-healing architectures that adapt to environmental changes without human intervention. Future research could explore how reinforcement learning from human feedback or preference-based optimization could be applied to refine strategic directives based on post-match expert review, creating a continuous improvement loop for hierarchical agents.

Explainability and Human-Agent Collaboration. While the framework emphasizes interpretability through modular design and structured outputs, future work could enhance explainability by generating natural language rationales for strategic directives, visualizing tactical decision trees, or providing post-hoc analysis of economic trade-offs. Investigating how human players can effectively collaborate with hierarchical agents through directive override, preference specification, or interactive strategy refinement could bridge the gap between autonomous systems and human-in-the-loop decision support (Kinniment et al., 2023). As in the source manuscript, supporting reference [47] remains a plain-text marker.

6.3 Final Thoughts

The journey from conceptual design to empirical validation of a hybrid hierarchical game agent framework has reinforced several foundational insights about the nature of intelligent behavior in complex, dynamic environments. First, hierarchical decomposition is not merely an engineering

convenience but a principled response to the multi-scale nature of real-world decision problems. By explicitly aligning computational paradigms with temporal resolution and reasoning complexity, the framework achieves a balance between adaptability and reliability that neither flat policies nor purely rule-based systems can attain in isolation.

Second, determinism and verification are essential for translating probabilistic reasoning into reliable action. The tool-first architecture, API-based validation, and ECC-style redundancy demonstrate that large language models can function as effective strategic directors when their outputs are constrained, verified, and grounded in executable schemas. This insight extends beyond game AI to any domain where autonomous systems must operate under uncertainty while maintaining safety, interpretability, and real-time responsiveness.

Third, resource efficiency and reproducibility are not secondary concerns but fundamental requirements for scientific progress. By designing a framework that operates within academic resource budgets, documents its implementation with methodological rigor, and provides comprehensive evaluation protocols, this work aims to lower the barrier to entry for hierarchical agent research. The hope is that other researchers can build upon, critique, and extend this foundation, accelerating collective progress toward more capable, transparent, and trustworthy autonomous systems.

Finally, the integration of diverse computational paradigms – classical control, machine learning, and language reasoning – represents a broader trend in artificial intelligence toward hybrid, neuro-symbolic architectures. Rather than seeking a single universal learning algorithm, the future of AI may lie in thoughtfully combining specialized components, each optimized for the aspects of intelligence it handles best. The hybrid hierarchical framework presented in this thesis offers one concrete instantiation of this vision, demonstrating that modular, interpretable, and resource-efficient architectures can achieve competitive performance in some of the most challenging decision-making environments available to researchers today.

As artificial intelligence continues to expand into increasingly complex, high-stakes domains – from autonomous vehicles and robotic assistants to scientific discovery and strategic planning – the principles of hierarchical decomposition, deterministic verification, and modular orchestration explored in this work will likely grow in relevance. By contributing a reproducible, empirically validated framework grounded in both theoretical rigor and engineering practicality, this thesis aims to support that broader trajectory toward more capable, transparent, and trustworthy autonomous systems.

Bibliography

- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 41–48.
- 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*.
- Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.
- Campbell, M., Hoane, A. J., and Hsu, F.-h. (2002). Deep blue. *Artificial Intelligence*, 134(1–2):57–83.
- Demediuk, S., York, P., Drachen, A., Walker, J. A., and Block, F. (2019). Role identification for accurate analysis in dota 2. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, volume 15, pages 130–138.
- Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.
- Drachen, A., Canossa, A., and Sørensen, J. (2013). Player-centric game analytics: Tools for evaluating player behavior and performance. In *Proceedings of the International Conference on the Foundations of Digital Games (FDG)*.
- Fujimoto, S., Meger, D., and Precup, D. (2019). Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2052–2062.
- Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pages 3207–3214.
- Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., et al. (2018). Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.
- Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 4565–4573.
- Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Sadigh, D., et al. (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, pages 9118–9147.
- Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially

- observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134.
- Kinniment, M., Jones, L., Zhu, H., Jun, T., Khorasani, A., Chis, N., Goldowsky-Dill, A., Green, R., Chudasma, M., Burgess, C., et al. (2023). Evaluating language-model agents on realistic autonomous tasks. *arXiv preprint arXiv:2312.11671*.
- Kostrikov, I., Nair, A., and Levine, S. (2021). Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*.
- Kulkarni, T. D., Narasimhan, K., Saedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 3675–3683.
- Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1179–1191.
- Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.
- Levy, A., Konidaris, G., Platt, R., and Saenko, K. (2019). Learning multi-level hierarchies with hindsight. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Millington, I. and Funge, J. (2009). *Artificial Intelligence for Games*. CRC Press, Boca Raton, FL, USA, 2nd edition.
- Nachum, O., Gu, S. S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 3307–3318.
- OpenDota (2026). Opendota api. Accessed Apr. 16, 2026.
- Park, J. S., O’Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pages 1–15.
- Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.
- Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 8634–8652.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2nd edition.
- Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211.
- Valve (2023). Dota patch notes 7.33.
- Valve (2025). Dota patch notes 7.39.
- Valve (2026a). Dota 2. Official website. Accessed Apr. 16, 2026.
- Valve (2026b). Dota 2 – hero builds. Official website. Accessed Apr. 16, 2026.
- Valve (2026c). Dota 2 heroes. Official website. Accessed Apr. 16, 2026.
- Valve Corporation (2026). Dota 2 patch notes and balance updates. Accessed Apr. 16, 2026.
- Valve Developer Community (2026a). Creating a dota-style map. Accessed Apr. 16, 2026.

- Valve Developer Community (2026b). Dota bot scripting. Accessed Apr. 16, 2026.
- Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3540–3549.
- 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.
- Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023). Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.
- Yannakakis, G. N., Liapis, A., and Alexopoulos, C. (2014). Computational game creativity. In *Proceedings of the International Conference on Computational Creativity (ICCC)*.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.