Tencent 腾讯

Agents Play Thousands of 3D Video Games

Zhongwen Xu, Xianliang Wang, Siyi Li, Tao Yu, Liang Wang, Qiang Fu and Wei Yang

Tencent

Abstract: We present PORTAL[1], a novel framework for developing artificial intelligence agents capable of playing thousands of 3D video games through language-guided policy generation. By transforming decision-making problems into language modeling tasks, our approach leverages large language models (LLMs) to generate behavior trees represented in domain-specific language (DSL). This method eliminates the computational burden associated with traditional reinforcement learning approaches while preserving strategic depth and rapid adaptability. Our framework introduces a hybrid policy structure that combines rule-based nodes with neural network components, enabling both high-level strategic reasoning and precise low-level control. A dual-feedback mechanism incorporating quantitative game metrics and vision-language model analysis facilitates iterative policy improvement at both tactical and strategic levels. The resulting policies are instantaneously deployable, human-interpretable, and capable of generalizing across diverse gaming environments. Experimental results demonstrate PORTAL's effectiveness across thousands of first-person shooter (FPS) games, showcasing significant improvements in development efficiency, policy generalization, and behavior diversity compared to traditional approaches. PORTAL represents a significant advancement in game AI development, offering a practical solution for creating sophisticated agents that can operate across thousands of commercial video games with minimal development overhead. Experiment results on the 3D video games are best viewed on our project website.

An agent must learn from its experience.
– Rich Sutton

1. Introduction

Games have served as foundational testing grounds for artificial intelligence development since the field's inception [2, 9, 15, 17, 26, 27]. They provide ideal environments for measuring and advancing cognitive capabilities including pattern recognition, memory utilization, reasoning, sophisticated planning and generalization. The pursuit of creating AI systems capable of generalizing effectively across diverse game environments represents a significant and ongoing challenge at the intersection of academic AI research and the commercial game industry.

The gaming landscape underwent a profound transformation with the rise of user-generated content (UGC) platforms, most notably pioneered by Roblox. UGC platforms have demonstrated unprecedented scale in both creation and consumption. By allowing users to design, build, and share their own games within a unified ecosystem, UGC games fundamentally altered the game development paradigm.

The explosive growth of UGC gaming created a unique challenge for traditional game-playing AI systems. While conventional AI approaches had made remarkable progress in mastering specific games with well-defined rulesets – from Go [17] to StarCraft II [26] – they faced inherent limitations in environments

[1] Portal symbolizes a gateway that provides access to thousands of game worlds and it represents connection and transition between different gaming domains.

characterized by constant flux and unbounded creativity. Traditional game AI typically relies on extensive domain-specific engineering, carefully crafted heuristics, or reinforcement learning methodologies that require millions/billions of training iterations within a single game environment. Such approaches simply cannot scale to match the pace and diversity of user-created content, where thousands of new games with unique mechanics, aesthetics, and objectives emerge daily. This fundamental mismatch between traditional AI capabilities and the rapidly expanding universe of user-generated games highlights a critical need for more adaptive, generalizable AI approaches – a gap that we are aiming to bridge here.

This paper introduces PORTAL(**P**olicy **O**ptimization and **R**easoning for **T**actical **A**rtificial **L**earning), a novel approach to generating game-playing artificial intelligence (AI) agents capable of operating across thousands of 3D video games. By leveraging large language models (LLMs) [1, 10, 22–24] to generate specialized behavior trees (BTs) [3] expressed in domain-specific language (DSL), we establish a new paradigm for AI agent development that combines the strategic reasoning of LLMs with the real-time performance requirements of commercial video games. Unlike traditional reinforcement learning approaches [2, 5, 14, 26, 32] that require massive simulation resources and distributed training, our method decouples the tactical planning process from execution, enabling rapid development and deployment of sophisticated agents. This decoupling allows for the generation of diverse, adaptable, and human-like behaviors while maintaining the computational efficiency necessary for commercial application. PORTAL represents a significant advancement in the field by transforming complex decision-making problems into language modeling challenges. Through an innovative hybrid architecture that integrates rule-based nodes with neural network components, we demonstrate how LLM-generated policies can be effectively executed in dynamic, real-time environments without the latency limitations that typically constrain LLM-based agents.

On methodology, we propose a *meta algorithm* for decision-making problems that fundamentally reimagines the role of large language models (LLMs) in game AI systems. Unlike prior works [1, 8, 27] where LLMs directly function as *actor policies*, our approach positions these models as *architects* that generate *policy structures*. This important innovation creates a hybrid architecture that leverages the strengths of both paradigms: LLMs design sophisticated policy frameworks composed of interconnected basic building blocks, with each node representing discrete functionality. These nodes can then be implemented as neural networks, enabling dynamic adjustment to changing game environments. The nodes can also be implemented with rule-based codes, where it brings exact control over behaviors. The resulting system maintains the strategic sophistication and broadly contextual awareness of LLMs while incorporating the adaptability and responsiveness of neural network approaches.

The remainder of this paper is organized as follows: Section 2 reviews related work in behavior trees, reinforcement learning, and language models for game playing AIs. Section 3 details our methodology, including the hybrid policy structure, DSL representation, and an agent-environment loop with reflection feedbacks. Section 4 presents experimental results, and Section 5 discusses implications and potential applications beyond gaming. We conclude our work and point out directions for future research.

2. Related Work

Behavior Trees: Behavior Trees (BTs) [3] have established themselves as a predominant paradigm for artificial intelligence in modern game development, particularly within sophisticated engines such as

Unreal Engine 5[2] and Unity. Their popularity stems from their capacity to organize complex AI behaviors in a structured, modular, and visually intuitive manner. The foundational components of BTs include:

- `Selector` Nodes: Implement prioritized decision-making by evaluating child nodes sequentially until one succeeds, enabling fallback behaviors.
- `Sequence` Nodes: Execute a series of actions in order, succeeding only if all component actions succeed, facilitating complex, multi-step behaviors.
- `Task` Nodes: Perform atomic game actions such as movement, attack sequences, or environmental interactions.
- `Condition` Nodes: Evaluate environmental states to determine execution paths.

This architecture is complemented by a Blackboard system that maintains state information accessible to all nodes within the tree. In commercial implementations such as Unreal Engine 5, a dedicated AI controller periodically evaluates the behavior tree against current game state, determining appropriate actions for non-player characters (NPCs).

The modular structure of BTs facilitates code reuse, simplifies debugging, and enables designers to implement sophisticated behaviors without extensive programming expertise. However, traditional BTs rely primarily on manually crafted rules and conditions, limiting their adaptability to novel or highly dynamic scenarios. Our work extends this established framework by incorporating neural components and LLM-based generation, preserving the interpretability and modularity of classical BTs while enhancing their adaptability and generalization capabilities.

Reinforcement Learning: Reinforcement Learning (RL) [20] has demonstrated remarkable achievements in mastering complex environments across various domains. AlphaGo [17]'s victory over world champion Lee Sedol in 2016 marked a watershed moment, demonstrating how deep neural networks combined with Monte Carlo Tree Search (MCTS) could achieve superhuman performance in challenging strategic domains. This approach initially leveraged human expert data before transitioning to pure self-play in subsequent iterations such as AlphaGo Zero [18] and AlphaZero [19], which surpassed its predecessor's capabilities without human guidance. Beyond board games, RL has tackled increasingly complex video game environments. OpenAI Five [2] demonstrated emergent team strategies in Dota 2, while DeepMind's AlphaStar [26] achieved Grandmaster-level performance in StarCraft II – both representing significant advances in handling high-dimensional state and action spaces with partial observability. Similarly, Tencent's Honor of Kings AI system [32] has pushed the boundaries further in complex Multiplayer Online Battle Arena (MOBA) games.

These systems [5, 14] typically employ large-scale distributed architectures, training on millions or billions of game frames to develop effective policies. While impressive in their capabilities, these approaches incur substantial computational costs – often requiring hundreds or thousands of GPUs and enormous CPUs for game simulations, operating for weeks or months – and frequently struggle with generalization beyond their specific training environments. Our work addresses these limitations by decoupling strategic planning from execution, significantly reducing computational requirements while enhancing cross-environment generalization.

Language Models for Game-playing AIs: Recent research has begun exploring the potential of large language models (LLMs) for creating game-playing agents. As comprehensively documented in the survey

[2] Unreal Engine 5 Documentation on Behavior Trees

by [6], these approaches have primarily focused on three categories of games:

- **Abstract Token Games**: Board games like Chess [19] and Go, where states and actions can be represented as discrete, symbolic tokens;
- **Text-Based Games**: Interactive fiction environments like Zork [25], where game interactions naturally occur through textual commands and descriptions;
- **API-Mediated Games**: Complex environments where game state is transformed into textual descriptions and LLMs generate actions as API calls to game interfaces.

Voyager [27] exemplifies the API-mediated approach, leveraging GPT-4 [10] to generate code for Minecraft tasks through the Mineflayer [11] JavaScript API. A distinctive feature of Voyager is its curriculum-based skill acquisition, progressively building a library of capabilities through tool usage and exploration. However, Voyager fundamentally relies on textual representations of game states rather than direct engagement with raw environmental data. Other research [8] has applied similar principles to complex strategy games like StarCraft II through frameworks like TextStarCraft, which translates game states into textual descriptions and facilitates LLM interaction through API calls. While demonstrating the potential of LLMs for strategic reasoning, these approaches face a significant limitation: the inherent latency of LLM inference renders them impractical for real-time applications, with reported gameplay durations extending to seven hours per match – clearly prohibitive for commercial deployment. More recently, Anthropic's Claude 3.7 Sonnet [1] has demonstrated impressive capabilities in Pokémon Red with zero-shot generalization, defeating Gym Leaders through extended reasoning. However, its success relies on relaxed time constraints that would be unacceptable in commercial gaming contexts. Our approach differs fundamentally from these precedents in three critical aspects:

- **Offline Policy Generation**: Rather than relying on LLM inference during gameplay, we utilize LLMs to generate policies expressed in Domain-Specific Language (DSL), which are then interpreted into efficient, executable code, eliminating inference latency during gameplay.
- **Hybrid Architecture**: We combine the strategic reasoning capabilities of LLMs with the real-time performance of neural networks and rule-based systems, leveraging the strengths of each approach while mitigating their individual limitations.
- **Cross-Game Generalization**: To our knowledge, we are the first to demonstrate agents capable of playing thousands of distinct 3D video games through a unified approach, highlighting the generalization capabilities of our method.

This synthesis of LLM-based policy generation with efficient execution mechanisms represents a novel contribution to the field, addressing the practical constraints that have limited the application of language models in real-time gaming environments.

3. Method

3.1. Overview

Our approach introduces a novel policy representation that formalizes the hybrid structure of game-playing AI agents. We define a policy π as a triple (Π, Θ, Φ), where:

- Π represents the hierarchical tree structure that defines the control flow and relationships between nodes. Formally, Π can be expressed as a directed acyclic graph (DAG) where nodes are drawn from $\Theta \cup \Phi$, and edges represent the execution flow between components;
- Θ denotes the set of neural network-parameterized task nodes: $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$. Each θ_i implements a mapping from an observation space O (a subset of the complete environment state S) to either a specific action or a probability distribution over possible actions. Mathematically, this can be expressed as $\theta_i : O \to A$ or $\theta_i : O \to P(A)$, where A represents the action space;
- Φ comprises the set of rule-based nodes: $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$. These include condition nodes and simple action nodes implemented through traditional programming. Each ϕ_j represents a deterministic function that maps an observation to either a boolean value (for condition nodes) or a specific action (for action nodes). This relationship can be formalized as $\phi_j : O \to \{\text{True}, \text{False}\}$ or $\phi_j : O \to A$.

This formalization offers several advantages over conventional approaches. First, it establishes a clear separation between the strategic structure of the policy (Π) and its implementation details (Θ and Φ). This separation enables independent optimization of each component, facilitating rapid iteration and refinement. Second, by combining rule-based and neural network components, the policy can leverage both the interpretability and reliability of hand-crafted rules as well as the adaptability and pattern recognition capabilities of neural networks. Finally, this representation provides a natural framework for language models to generate and modify policies by focusing on the structural aspects (Π) while leaving the implementation details of neural components (Θ) to specialized training procedures.

Our policy representation can be formally defined as:

Definition 3.1. A policy $\pi = (\Pi, \Theta, \Phi)$ consists of a tree structure Π, neural network task nodes $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$, and rule-based nodes $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, where Π determines the hierarchical organization and execution flow of elements from Θ and Φ.

3.2. Domain-specific Language Representation

We have developed a domain-specific language (DSL) to represent behavior trees in a format that is both human-readable and machine-executable. This DSL serves as the interface between large language models and the policy execution environment, enabling seamless translation of strategic concepts into operational policies.

The syntax of our DSL follows a hierarchical structure that directly mirrors the organization of behavior trees. Key syntactic elements include:

- **Hierarchical Structure**: Indentation denotes parent-child relationships between nodes, providing a visual representation of the tree's organizational structure. This design choice enhances readability while maintaining a direct correspondence to the underlying computational representation.
- **Node Types**: The DSL supports four primary node types, each serving a distinct role in policy execution:
  - Selector Nodes (`selector:`): Implement prioritized execution, attempting child nodes sequentially until one succeeds.

- Sequence Nodes (`sequence:`): Execute child nodes in order, continuing only if each node succeeds.
- Condition Nodes (`condition:<condition_key>`): Evaluate environmental predicates, determining execution paths based on game state.
- Task Nodes (`task: <action_key><param_key>`): Execute specific actions within the game environment, potentially utilizing neural networks for implementation.
- Logical Operations: The DSL incorporates logical operations, such as negation (`condition: no`), enabling the construction of complex conditional statements without requiring additional syntax.

The following example illustrates a simple policy for a first-person shooter (FPS) game:

```
selector:
  sequence:
    condition: has_enemy_in_view
    task: shoot random_enemy_in_view
  sequence:
    condition: no
    condition: has_enemy_in_view
    task: move_to random_enemy_location
```

Listing 1: A Behavior Tree DSL Example

This policy implements a straightforward combat strategy: if an enemy is visible, select a random visible enemy and attack; otherwise, move toward a known enemy location. The simplicity and clarity of this representation belies the sophisticated decision-making process it encodes, demonstrating the expressive power of our DSL.

A critical advantage of this representation is its compatibility with large language models, which can generate, interpret, and modify these tree structures based on high-level strategic descriptions. Furthermore, the DSL's hierarchical nature facilitates incremental refinement and composition of policies, enabling the construction of complex behaviors from simpler building blocks. This property is particularly valuable for the iterative improvement process employed in our approach.

3.3. Extensions Beyond Classic Behavior Trees

Traditional behavior trees typically rely exclusively on rule-based implementations, employing deterministic algorithms like A* pathfinding for navigation tasks. While effective in static environments, these conventional approaches falter when confronted with dynamic scenarios featuring moving obstacles, adversarial agents, or complex environmental interactions. We address this limitation by introducing a hybrid architecture that augments the classical behavior tree framework with neural network-parameterized task nodes.

As shown in Figure 1, our architecture instantiates specific task nodes as neural networks, particularly for operations that demand adaptive responses to environmental complexity. For instance, rather than implementing a `move_to` task with a deterministic pathfinding algorithm, we utilize a neural network trained to navigate dynamic environments while avoiding obstacles and responding to changing conditions. *Note that the neural nets applied in our work are tiny nets such as two layers of fully-connected layers / convolutional layers.*

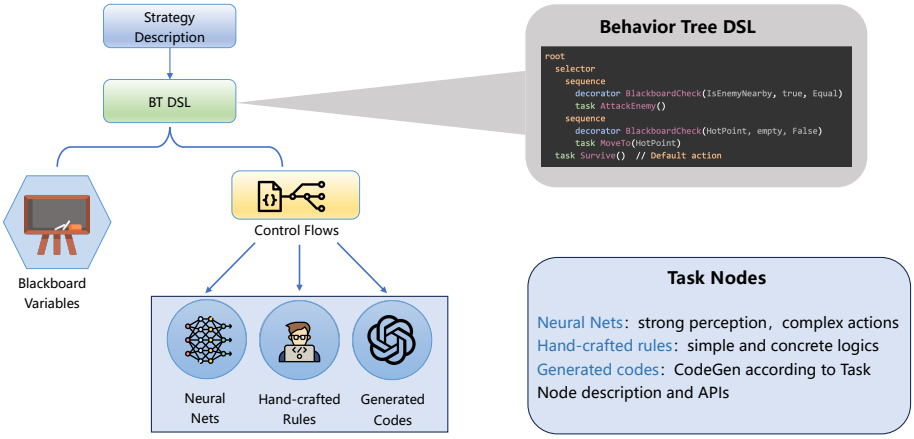

Figure 1: Overview of the behavior tree generation process using LLMs.

This hybrid approach represents a fundamental advancement over both purely rule-based behavior trees and end-to-end neural policies. By combining the strengths of symbolic reasoning (through the behavior tree structure) with subsymbolic learning (via neural networks), we achieve policies that are simultaneously more adaptive, interpretable, and generalizable than either approach in isolation.

A fundamental innovation in our approach is the reconceptualization of decision-making problems as language modeling tasks. Rather than training policies through direct interaction with environments, we leverage the capabilities of LLMs to generate DSL representations of behavior trees. This transformation offers several compelling advantages:

- **Rapid Iteration**: The language modeling approach facilitates rapid prototyping and refinement of policies, as modifications can be made at the structural level without requiring extensive retraining of neural components.
- **Enhanced Environmental Perception**: Neural network-parameterized task nodes leverage sophisticated perceptual capabilities to process environmental information, enabling more informed decision-making in complex scenarios. Rather than responding to simplified abstractions of the environment, these nodes can integrate rich sensory data to guide their actions.
- **Simplified Reward Structures**: Each neural network task node is designed with a focused, often unidimensional reward function. For example, a `move_to` node might optimize solely for minimizing distance to a target location, while a shoot node maximizes accuracy against a specific enemy. This decomposition of complex, multi-objective reward landscapes into simpler, targeted optimization problems significantly enhances learning efficiency and generalization capabilities.

The power of the proposed system demonstrates that even with tiny neural networks Θ , a well-designed Domain-Specific Language (DSL) can leverage them to construct a powerful policy π.

Transformation from Decision-Making into Language Modeling

We transform the decision-making problems into language modeling problems of building up DSL for policies. This transformation allows us to leverage the full potential of LLMs as reasoning engines for strategic planning, while maintaining the computational efficiency necessary for real-time execution in dynamic environments.

3.4. An Agent-Environment Loop

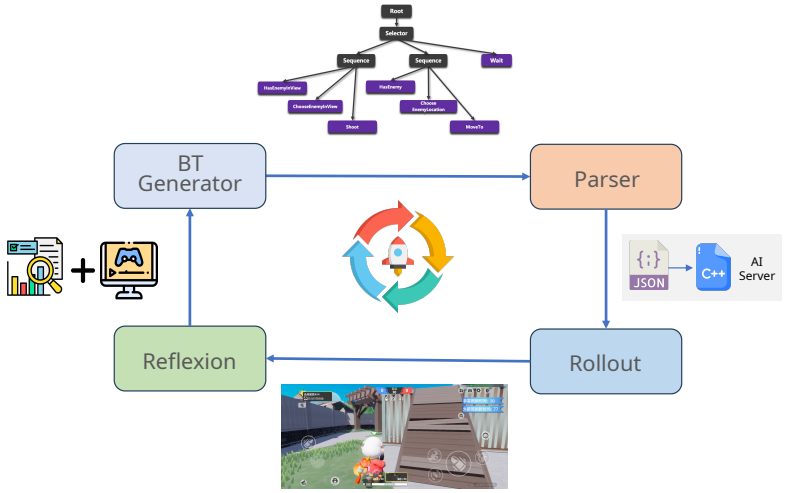

Figure 2: System architecture showing the integration of LLM-based behavior tree generation with environment interaction and reflexion

Following the reinforcement learning paradigm, we conceptualize the interaction between our agents and their environments as a cyclical process of policy refinement. However, our approach diverges from traditional methods in its implementation of this cycle. The agent continuously refines its policy parametrization, denoted as Π, through structured interaction with the environment and analysis of the resulting outcomes.

Central to our approach is the Reflexion [16, 31] module, which serves as the bridge between environmental feedback and policy improvement. This module processes two distinct types of feedback:

- **Quantitative Game Metrics**: Direct numerical measurements extracted from the game environment, such as kills, deaths, objective completions, and resource utilization. These metrics provide precise, objective evaluations of policy performance across specific dimensions. To make these metrics accessible to language models, we translate them into natural language descriptions that highlight relevant patterns and trends.
- **Vision-Language Model Analysis**: A complementary feedback mechanism that utilizes a vision-language model (VLM) to analyze "mini-map" representations of gameplay from a bird's-eye perspective. This analysis provides high-level strategic insights that may not be captured by numerical metrics alone, such as spatial control patterns, tactical adaptations, and coordination effectiveness.

The dual-feedback mechanism enables comprehensive policy evaluation and improvement at multiple levels of abstraction. The quantitative metrics offer granular feedback on tactical execution, while the VLM analysis provides broader strategic context. By integrating these complementary perspectives, our system can identify both local optimizations and global strategic shifts that may improve overall performance. The iterative improvement process follows a structured cycle: the LLM generates an initial behavior tree expressed in DSL, based on high-level strategic descriptions. This behavior tree is compiled into an executable policy and deployed in the game environment. The agent interacts with the environment,

generating gameplay data. The Reflexion module processes this data, extracting quantitative metrics and producing VLM analyses. These insights are provided to the LLM, which generates an improved behavior tree that addresses identified weaknesses. This cycle continues iteratively, with each generation refining the policy based on accumulated insights from previous deployments. The result is a continuously improving agent that adapts to the specific challenges and opportunities presented by its environment. The overall procedure is shown in Figure 2.

Policy Improvement with Experience

Through Reflexion on environmental experience, the LLM updates the policy DSL Π to optimize for either quantitative game metrics or global tactical analysis.

3.5. Chain-of-Thought for Tree Generation and Reflexion

The hierarchical nature of our policy representation aligns naturally with a structured reasoning approach. Our policies exhibit a tree-like organization where higher levels manage strategic planning while lower levels execute tactical actions to achieve specific objectives. For instance, within this hierarchy, `Sequence` nodes decompose complex goals into sequential subtasks executed from left to right, while `Selector` nodes implement prioritized decision-making, attempting child nodes in order until one succeeds.

To leverage this hierarchical structure effectively, we implement a level-by-level generation process utilizing Chain-of-Thought (CoT) prompting [29]. This technique instructs the LLM to construct the behavior tree progressively, beginning with root nodes and systematically expanding through each subsequent level. This approach offers several advantages – **Focused Reasoning**: By addressing one level at a time, the LLM can concentrate its reasoning capacity on a manageable subset of the policy structure, improving overall coherence and strategic alignment; **Iterative Refinement**: The level-by-level approach facilitates targeted revision of specific tree components without requiring complete regeneration of the entire policy.

Complementing this generative process, we implement a level-by-level reflexion mechanism that evaluates and revises specific sections of the tree based on performance feedback. When behavioral deficiencies are identified, the LLM can focus its analysis on the relevant subtree, proposing targeted modifications that address specific shortcomings while preserving effective components.

Our CoT implementation employs structured prompting with explicit reasoning steps, guiding the LLM through a systematic tree construction process: First, the LLM identifies the primary strategic objectives and potential challenges. Next, it determines the appropriate root node type (typically a Selector) to organize high-level decision-making. For each child of the root, the LLM articulates a specific subgoal and constructs an appropriate subtree. This process continues recursively until all paths terminate in executable task nodes. Throughout this process, the LLM explicitly documents its reasoning, explaining why particular node types, conditions, and actions were selected. This explicitness not only improves the quality of the generated policies but also enhances interpretability, providing human designers with insight into the strategic principles underlying the policy structure.

Key Takeaways

Tree-like CoTs facilitate better reasoning and reflection in strategy planning.

3.6. Sampling and Post-training

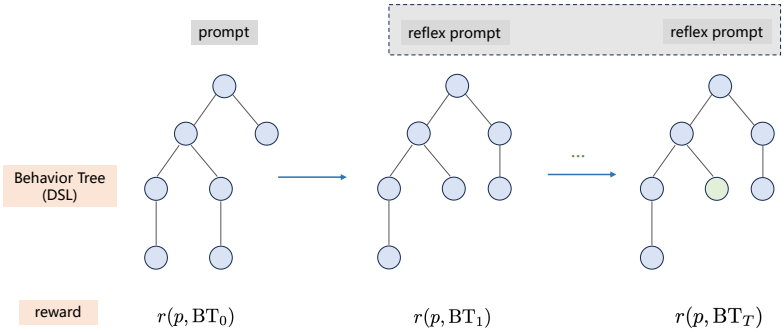

Figure 3: The LLM-based behavior tree generation and refinement process

By recasting decision-making as a language modeling problem, we can generate extensive training data through systematic exploration of the policy space. This exploration utilizes breadth-first search (BFS) to sample N potential behavior trees in parallel, retaining the K best-performing policies in each iteration based on direct environmental rewards.

The sampling process generates valuable trajectory data that can be leveraged for model improvement. Unlike traditional reinforcement learning, which collects environment trajectory data (i.e., state-action pairs), our approach collects LLM trajectory data – sequences of prompts, DSL outputs, and associated rewards:

$$[(\text{prompt}_0, \text{DSL}_0, r_0), (\text{prompt}_1, \text{DSL}_1, r_1), (\text{prompt}_2, \text{DSL}_2, r_2), \ldots (\text{prompt}_T, \text{DSL}_T, r_T)]$$

This data can be utilized for Supervised Fine-Tuning (SFT) or reinforcement learning approaches tailored to language models. The structured format directly connects strategic descriptions, policy implementations, and performance outcomes, enabling the model to learn the relationships between high-level goals and effective policy structures.

As illustrated in Figure 3, each iteration i produces a specific DSL output representing a policy and receives reward feedback r_i from the environment. By structuring each step in the LLM inference process with explicit tags like `<reflection>...</reflection>` and `<think>...</think>` [4], we create "Meta CoTs" [30] that enable the model to systematically explore the solution space and iteratively improve its policy outputs.

3.7. A Policy Network to Schedule Policies

The generative capabilities of LLMs, whether frozen or fine-tuned, enable the production of diverse policy candidates expressed as DSL descriptions. These candidates represent various strategic approaches to addressing game challenges. However, this diversity introduces a new challenge: determining which policy is most appropriate for a specific environment state at a given moment. While exhaustive evaluation of all candidates is theoretically possible, it would be computationally prohibitive in real-time applications.

To address this challenge, we introduce a policy scheduling network that dynamically selects from a repertoire of pre-generated behavior trees based on environmental conditions. This network receives the same state observations as traditional reinforcement learning policies but produces a fundamentally

different output. Rather than directly selecting atomic actions, it identifies the most suitable behavior tree for the current context from a pre-defined library of options.

This approach draws inspiration from the options [20, 21] framework in hierarchical reinforcement learning, where options represent temporally extended courses of action that an agent can follow. In our implementation, each behavior tree constitutes an option – a predetermined policy that, once selected, governs the agent's behavior until completion or interruption. The policy network operates at a meta-decision level, determining when to switch between these options based on changing environmental conditions. The policy network receives environmental observations and selects the most appropriate behavior tree from the available library. This tree is then executed until completion or until the network determines that a different strategy would be more effective given updated environmental conditions.

4. Experiments

We conducted experiments on Yuan Meng Star[3], a platform developed by TiMi Studio under Tencent Games, which hosts tens of millions of User Generated Content (UGC) games. Yuan Meng Star provides players with a comprehensive UGC platform featuring built-in editors, tools, and assets to create their own games. The platform encompasses a diverse range of game genres, including speedrunning, survival, first-person shooter (FPS), and party games. For our research, we demonstrated the efficacy of our proposed method specifically on FPS games created by thousands of players, though the same principles can be applied to develop agents for other game genres as well. The LLMs used to produce the DSL are `Qwen2.5-32B-Coder` [24]. All the materials of the shown BTs can be found on our project website, including the illustration, DSL and JSON files of the BTs. The neural networks in the task nodes have only two Convolutional layers + Fully-connected layers.

Due to copyright restrictions associated with UGC games, we present only a select subset of representative games in this paper. However, it should be noted that comprehensive testing across a substantially broader range of UGC environments has been conducted internally, with consistent results supporting our findings. The examples showcased here were specifically chosen to illustrate key aspects of our methodology while respecting intellectual property constraints.

4.1. Optimization for Game Metrics

We first demonstrate the effectiveness of our approach through a Breadth-first Search (BFS) procedure designed to optimize specific game metrics. Figure 4 illustrates two independent experimental runs optimizing the "time between kills" metric, where 100% represents optimal performance. Individual Behavior Trees (BTs) are depicted in lighter colors, while the top-performing BT is highlighted with solid dots. The results clearly show that through successive reflection iterations, the individual BTs converge toward near-optimal performance. Notably, the top-performing BTs maintain consistent performance across iterations. These findings confirm that our proposed algorithm, through structured agent-environment interactions, successfully optimizes specified game metrics by refining the underlying policy structure. This demonstrates the viability of our meta-algorithmic approach in which LLMs architect policies rather than directly outputting actions.

[3] https://en.wikipedia.org/wiki/Yuan_Meng_Star

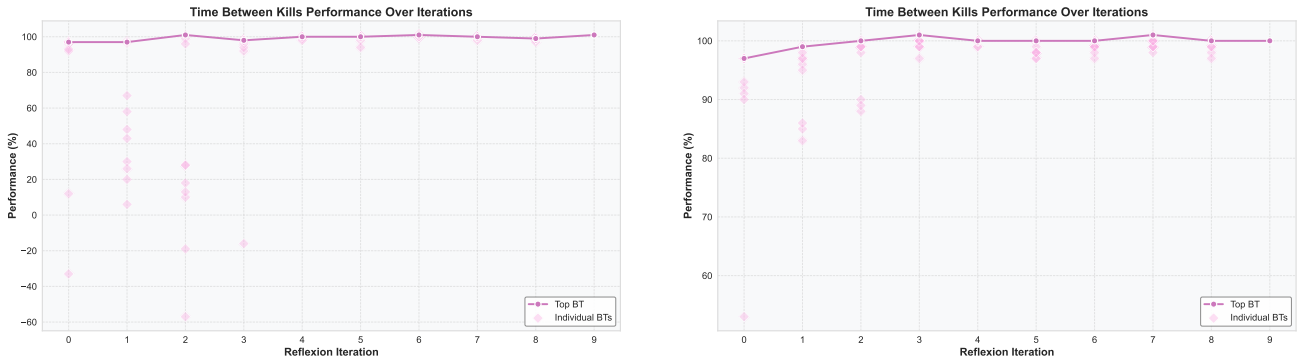

Figure 4: Two independent runs of Breadth-first Search (BFS).

4.2. Vision-Language Models for Reflexion

Vision-Language Models (VLMs) such as Gemini [22] provide powerful video understanding capabilities that can analyze extended gameplay footage through natural language interaction. This multimodal understanding capacity offers a valuable mechanism for tactical planning assessment and refinement. In traditional competitive games like Dota and StarCraft II, mini-maps provide players with critical global information to inform strategic decision-making. While the games in our experimental environment do not natively provide such mini-maps, we generate them synthetically using environmental data including map layouts, player positions, status indicators, and other relevant information. We accumulate this information throughout gameplay sessions and feed the resulting replay recordings into a VLM, enabling comprehensive analysis of global tactical strategies. For our evaluation of first-person shooter games, we structure this analysis across five critical dimensions:

- **Map Control**: Assessment of territorial dominance, strategic position maintenance, and spatial resource utilization.
- **Adaptability**: Evaluation of the agent's capacity to adjust strategies in response to changing circumstances or opponent behaviors.
- **Team Coordination**: Analysis of collaborative behaviors, role specialization, and synchronized actions.
- **Team Aggression**: Measurement of offensive initiative, pressure application, and risk-taking behavior.
- **Goal Achievement**: Assessment of progress toward primary and secondary objectives.

For each dimension, we employ specialized prompts that guide the VLM to provide detailed analysis with specific justifications and quantitative ratings. The resulting multi-dimensional assessment, visualized in Figure 5, provides a comprehensive view of tactical performance that complements traditional numerical metrics. The integration of VLM-based analysis into our Reflexion module has demonstrated significant benefits for behavior tree generation. As illustrated in Figure 5, policy improvements guided by VLM analysis show consistent enhancement across all five tactical dimensions. This improvement pattern highlights the complementary nature of VLM-based reflection and traditional numerical assessment – while numerical metrics effectively capture local performance indicators (e.g., accuracy, resource

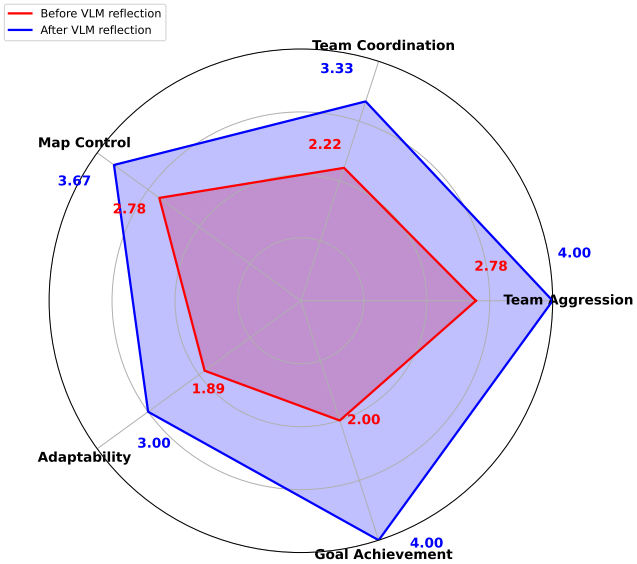

Figure 5: Effects of VLM reflexion on game plays.

efficiency), VLM analysis excels at identifying global tactical patterns that might not be immediately apparent from statistical measures alone.

4.3. Playing Thousands of First-Person Shooter (FPS) Games

Figure 6: Illustrations of a basic policy playing $3 \times 3 = 9$ FPS games.

Figure 6 presents visual evidence of our policies operating across a diverse set of 9 first-person shooter games (arranged in a 3×3 grid). Video demonstrations, all the materials about the behavior trees (including illustration, DSL, and JSON files) are available on our project website. These results demonstrate the remarkable generalization capabilities of our hybrid policies, which successfully transfer across games with varying visual styles, mechanical implementations, and environmental configurations.

Figure 7: Illustrations of a team attack policy playing $3 \times 3 = 9$ FPS games.

We present a case study focused on team coordination optimization. We applied our iterative refinement process with specific emphasis on team coordination metrics. The resulting progression, culminating in the structure shown in Figure 7, demonstrates substantial qualitative improvements in collaborative behaviors, including coordinated positioning, covering fire patterns, and objective-focused role specialization. An additional example of advantage point policy is shown on our project website.

4.4. Instant Development Procedure

A particularly significant advantage of our approach is its enablement of a novel and highly efficient deployment pipeline. Traditional approaches to behavioral adjustment in game AI typically require computationally expensive retraining of neural networks, resulting in substantial development delays and resource requirements. In contrast, our methodology facilitates immediate adaptation through dynamic behavior tree generation. PORTAL generates new Behavior Trees based on specified criteria, exporting the resulting policy as a JSON configuration file that can be directly ingested by the game server. This mechanism enables real-time adaptation of agent behavior within the game environment without requiring any recompilation or neural network retraining. The practical implications of this capability are substantial:

- **Rapid Iteration**: Game designers can specify behavioral modifications in natural language, receive an updated policy within minutes, and immediately test the results in-game.
- **Dynamic Difficulty Adjustment**: Servers can automatically generate customized opponent behaviors tailored to specific player skill levels or play styles.
- **Environment-Specific Adaptation**: Policies can be dynamically adjusted to accommodate new game environments, objectives, or rule modifications without developer intervention.

This instant development procedure represents a paradigm shift in game AI deployment, transitioning from the traditional cycle of specification, implementation, training, and deployment to a streamlined process where natural language descriptions directly translate to executable policies. The resulting efficiency gains not only accelerate development but also enable entirely new approaches to dynamic content generation and personalized gaming experiences.

5. Discussions

Beyond FPS Games: The methodology described in this paper extends readily beyond the first-person shooter genre explored in our experiments. The core architecture – leveraging LLMs to generate behavior trees expressed in DSL – provides a flexible framework applicable to diverse game genres and interaction paradigms. Adapting our approach to new game categories follows a standardized procedure:

- **Identify the basic action nodes** required for the target genre, such as resource management in strategy games, spell casting in RPGs, or evasive maneuvers in racing games.
- **Train focused neural networks** to handle specific navigation and skill requirements for the targeted genre, ensuring effective execution of low-level tasks.
- **Modify the prompting** framework to incorporate relevant game mechanics descriptions and desired tactical strategies appropriate for the new context.

The fundamental decomposition principle – separating high-level strategic planning from low-level execution – remains consistent across genres, enabling rapid adaptation to new gaming paradigms. This versatility stems from the generality of the behavior tree formalism and the flexibility of our DSL representation, which can accommodate diverse action spaces and strategic considerations without structural changes to the underlying architecture.

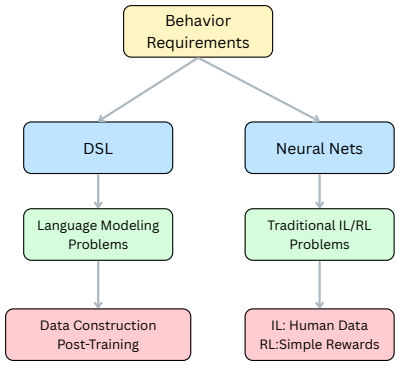

Figure 8: Decomposition of AI behavior requirements and How to tackle them.

Generalization: A persistent challenge in traditional reinforcement learning is achieving robust generalization to unseen environments. Conventional RL agents typically exhibit substantial performance degradation when confronted with even minor variations in environment dynamics or reward structures—a phenomenon attributable to overfitting to specific training conditions. Our approach addresses this challenge through a two – tiered generalization strategy as illustrated in Figure 8:

- **Tactical-Level Generalization**: The high-level strategic structure, represented through behavior trees or DSL expressions, captures generalizable decision principles that transfer effectively across environments. For example, the fundamental strategic patterns in FPS games – such as prioritizing threat elimination, position optimization, and resource management – remain consistent regardless of specific game implementation details. These tactical structures transfer seamlessly between games from different studios or created by different developers, providing a robust foundation for cross-environment performance.

- **Action-Level Generalization**: At the execution level, we parameterize specific actions using focused neural networks trained with simplified, single-dimensional reward functions. This decomposition represents a significant departure from conventional approaches that attempt to optimize multi-objective reward functions simultaneously. By isolating specific objectives, such as navigation efficiency, target acquisition, or threat avoidance, we create more generalizable task-specific policies that perform consistently across diverse environmental conditions.

The decomposition inherent in our hybrid architecture converts complex, multi-objective optimization problems into collections of simpler, more tractable subproblems. This approach mitigates the ambiguity and conflicting objectives often present in monolithic reward functions, resulting in substantially improved generalization. Each neural component focuses on a well-defined task with clear success criteria, enabling more effective learning and transfer across environments.

Diversity of Behaviors: Behavioral diversity represents a critical factor in creating engaging and realistic game environments populated by non-player characters. Our framework facilitates diverse agent behaviors through systematic variation at both tactical and execution levels as shown in Figure 8. At the tactical level, diversity emerges from several sources: variations in the prompting descriptions provided to the LLM, resulting in structurally distinct behavior trees; differences in the available node sets accessible to the DSL, constraining or expanding strategic possibilities; explicit diversity objectives incorporated into the generation process, encouraging exploration of alternative strategic approaches.

At the execution level, diversity manifests through: training local control networks on different distributions of human data, capturing varying skill levels and play styles; introducing controlled stochasticity into neural network outputs, creating natural variations in execution; parameterizing reward functions to emphasize different execution characteristics, such as aggression, caution, or efficiency.

The hierarchical decomposition of our approach makes achieving behavioral diversity substantially more tractable than in classical reinforcement learning systems. Rather than attempting to discover diverse behaviors through environmental exploration – a computationally intensive process with limited guarantees, we can explicitly engineer diversity through structural variations in the generated policies. This approach provides game designers with unprecedented control over the distribution and characteristics of AI behaviors, enabling the creation of rich, varied game experiences with minimal development overhead.

Multi-agent Collaboration: While our primary focus has been on single-agent scenarios, our framework extends naturally to multi-agent collaboration contexts. We adapt our approach to collaborative settings by drawing inspiration from the Centralized Training with Decentralized Execution (CTDE) paradigm prevalent in multi-agent reinforcement learning. For multi-agent scenarios, we extend our DSL to incorporate joint actions and states involving multiple agents. In this extended formalism, nodes can specify coordinated behaviors such as "agents A and B move to flanking positions while agent C provides covering fire." This higher-level representation captures the strategic coordination essential for effective team play. Following the construction of this multi-agent DSL representation, we decompose it into agent-specific behavior trees, enabling decentralized execution. Each agent receives an individualized behavior tree derived from the joint strategy, tailored to its specific role within the coordinated plan. This decomposition preserves the strategic coordination specified in the joint representation while enabling autonomous execution by individual agents. The resulting multi-agent system exhibits coordinated behaviors emerging from explicit strategic planning rather than implicit policy convergence, yielding more interpretable and controllable collaborative dynamics than typically achieved through end-to-end multi-agent reinforcement learning.

Connections to Meta Chain-of-Thought: Recent work on Meta Chain-of-Thought (Meta-CoT) [30] surveys techniques for enabling Large Language Models (LLMs) to perform complex reasoning, proposing a framework to model the reasoning process as:

$$p(\mathbf{a}, \mathbf{z}_1, \ldots, \mathbf{z}_n | \mathbf{x}) \propto \int \underbrace{p(\mathbf{a}, \mathbf{z}_1, \ldots, \mathbf{z}_n | \mathbf{z}_1, \ldots, \mathbf{z}_k, \mathbf{x})}_{\text{Joint Answer+CoT}} \underbrace{\prod_{t=1}^{K} p(\mathbf{z}_t | \mathbf{z}_{<t}, \mathbf{x})}_{\text{Meta-CoT}} d\mathbf{z},$$

where $\mathbf{x}$ represents the input, $\mathbf{a}$ is the final answer, and $\mathbf{z}_i$ are the intermediate reasoning steps (Chain-of-Thought). The process $\mathbf{x} \to \mathbf{z}_1 \to \ldots \to \mathbf{z}_K$ is referred to as Meta-CoT, representing a higher-level reasoning process about the *reasoning steps themselves*. This framework emphasizes the iterative refinement of thought processes, a key aspect of System 2 reasoning.

Our methodology embodies a similar meta-reasoning paradigm but operates at a different level of abstraction. Rather than using LLMs to map environment states directly to actions, our system employs them to generate policy structures expressed in DSL. This distinction can be formalized as:

$$\text{Policy: } \pi : S \to A$$
$$\text{Meta-Policy: } \Pi : \text{Policy Description } \to \pi$$

In this formulation, the meta-policy Π accepts a description of a policy as input and produces an executable policy π as output. This represents a meta-reasoning process where the LLM designs the policy structure rather than directly controlling agent actions.

This approach positions LLMs as *policy architects* rather than direct controllers, leveraging their reasoning capabilities for strategic design while delegating execution to more efficient specialized components. The parallels with Meta-CoT suggest promising directions for further research, particularly in developing more sophisticated meta-reasoning frameworks for policy generation and refinement.

Tool Uses and Function Calls: There are connections to tool uses [12, 13] and function calls, but our proposed system offers distinct advantages. Tool uses enable language models to interact with external tools to overcome inherent limitations in performing specialized tasks. While one might conceptualize our proposed system as a form of tool use if neural network task nodes are viewed as tools, our approach fundamentally differs in several key aspects. The primary distinction is that previous work positions LLMs as actors interacting directly with the environment, whereas our system deploys LLMs as architects. In our framework, LLMs do not process environment states as inputs to determine tool or function selection, which enables our system to integrate seamlessly with real-time game engines. Importantly, all planning and Domain-Specific Language (DSL) architecture is executed offline rather than online, resulting in significant performance benefits for time-sensitive applications while maintaining the strategic advantages of LLM-based planning.

Extensions to Other Applications: The structural principles underlying our approach extend naturally beyond gaming to various other domains involving hierarchical control problems. The behavior tree DSL abstraction we have developed provides a generalizable framework for complex decision-making across diverse applications. Particularly promising extension domains include:

- **Embodied AI**: Robotic systems face similar challenges in decomposing complex tasks into manageable components while maintaining strategic coherence. Our approach offers a framework for generating interpretable policies that combine high-level strategic planning with specialized execution components tailored to specific robotic embodiments. An idea in the same principle has been explored in a concurrent work Gemini Robotics [7].
- **Autonomous Driving**: Vehicle control involves complex decision hierarchies spanning strategic route planning, tactical maneuver selection, and precise motion control. The DSL representation can capture these hierarchical relationships while neural components handle execution under varying environmental conditions.

Our approach offers two key advantages for these extended applications. First, the high-level policies expressed in DSL exhibit strong transfer capabilities across different tasks and environments, enabling knowledge reuse across similar problem domains. Second, the hierarchical decomposition inherent in the behavior tree structure significantly reduces the complexity of training complete end-to-end systems, facilitating better generalization across diverse embodiments and tasks. Furthermore, the interpretability of the DSL representation addresses a critical requirement in many real-world applications, particularly those subject to safety constraints or regulatory oversight. By making policy structures explicit and readable, our approach facilitates analysis, verification, and targeted modification – essential capabilities for deployment in sensitive domains.

6. Conclusions

In this paper, we have presented PORTAL, a novel framework for generating game-playing AI agents using large language models to produce behavior trees expressed in domain-specific language. Our approach fundamentally reconceptualizes AI agent development for video games by transforming complex decision-making problems into language modeling tasks, enabling rapid policy generation and deployment across thousands of diverse gaming environments.

These innovations collectively address critical limitations of previous approaches. Unlike traditional reinforcement learning methods, our framework eliminates the need for massive simulation resources and distributed training, reducing development time from weeks or months to hours or minutes. In contrast to existing LLM-based approaches, our method avoids the latency issues associated with online inference, enabling deployment in games with strict real-time requirements.

Our experimental results demonstrate the effectiveness of this approach across thousands of first-person shooter games, showcasing significant improvements in development efficiency and policy generalization. The instant development procedure enabled by our framework represents a paradigm shift in game AI deployment, offering unprecedented capabilities for rapid iteration and adaptation.

PORTAL represents a significant step toward more efficient, adaptable, and capable artificial intelligence for interactive systems. By bridging the gap between the strategic reasoning capabilities of language models and the real-time performance requirements of dynamic environments, our approach opens new possibilities for the next generation of decision-making AI and beyond.

References

[1] Anthropic. Visible extended thinking in large language models. https://www.anthropic.com/research/visible-extended-thinking, 2025. Accessed: March 2025.

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chelsea Hunn, Imanol Luengo, Jack Rae, Rachel , Bob McGrew, Tyna Eloundou Nekoul, Matthias Plappert, Dario Amodei, and Wojciech Zaremba. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[3] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018.

[4] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, and ... (add other authors as needed or use et al.). DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[5] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[6] Roberto Gallotta, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. Large language models and games: A survey and roadmap. *IEEE Transactions on Games*, 2024.

[7] Gemini Robotics Team, Google DeepMind. Gemini Robotics: Bringing AI into the physical world. *arXiv preprint*, 2025.

[8] Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Runji Lin, Yuqiao Wu, Jun Wang, and Haifeng Zhang. Large language models play StarCraft II: Benchmarks and a chain of summarization approach. *Advances in Neural Information Processing Systems*, 37:133386–133442, 2024.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015. doi: 10.1038/nature14236.

[10] OpenAI, Josh Achiam, Sasha Adler, Sandhini Agarwal, Sandeep Ahmad, Ilge Akkaya, Alty Aleman, David Almeida, Elie Altman, Alekh Alvarez, Haiming Anderson, Mira Anderson, Jyoti Aneja, Anthony Anton, Amanda Askell, Haidar Aslam, Alex Azer, Karsen Bach, Yuntao Bai, Mark Balwit, Kendra Banks, Kaylee Batmanghelich, Daniel Baxter, Pierre Beres, Stella Biderman, Nicholas Burnell, Alessandro Achille, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[11] PrismarineJS. Mineflayer: A high-level JavaScript API for creating Minecraft bots. https://github.com/PrismarineJS/mineflayer, 2025. Accessed: 2025-03-06.

[12] Yujia Qin, Shengding Liang, Houqiang Ye, Ge Liu, Yi Zhang, Weizhu Han, Xin Jin, Lifan Yang, Nianwen Xue, Jiawei Han, and Jie Tang. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023.

[13] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[15] Claude E. Shannon. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950. doi: 10.1080/14786445008521796.

[16] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

[17] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

[18] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017. doi: 10.1038/nature24270.

[19] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

[20] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[21] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs, a framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[22] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[23] Qwen Team. Qwen2.5: An innovative, generalist, and open large language model family. *arXiv preprint*, June 2024. URL https://qwenlm.github.io/blog/qwen2.5/. Alibaba Cloud.

[24] Qwen Team. Qwen2.5-Coder: An enhanced large language model for code understanding and generation. *arXiv preprint*, June 2024. URL https://qwenlm.github.io/blog/qwen2.5-coder/. Alibaba Cloud.

[25] Chen Feng Tsai, Xiaochen Zhou, Sierra S Liu, Jing Li, Mo Yu, and Hongyuan Mei. Can large language models play text games well? Current state-of-the-art and open questions. *arXiv preprint arXiv:2304.02868*, 2023.

[26] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Tom Eccles, Norman Casares, Thomas Budden, Simon Osindero, Aliaksei Veliantsev, Johannes Agapiou, James Devlin, Yuval Tassa, Brendan Tracey, Demis Hassabis, and Koray Kavukcuoglu. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z.

[27] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. VOYAGER: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2023.

[28] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents, 2024.

[29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain of Thought prompting elicits reasoning in Large Language Models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[30] Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, et al. Towards System 2 reasoning in LLMs: Learning how to think with meta Chain-of-Thought. *arXiv preprint arXiv:2501.04682*, 2025.

[31] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.

[32] Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, Qiaobo Chen, Yinyuting Yin, Hao Zhang, Tengfei Shi, Liang Wang, Qiang Fu, Wei Yang, and Lanxiao Huang. Mastering complex control in MOBA games with deep reinforcement learning. In *AAAI*, pages 6672–6679. AAAI Press, 2020.

A. Prompt Template

We use a prompt template as in OpenHands [28] which is rendered by Jinja2 engine to enable flexible prompt management. The template is extended with Markdowns from other folders and files to form a concrete prompt for the LLM to process.

```
# Task
You are a diligent AI programmer working on behavior trees design for game AI.
You use domain specific language (DSL) to behavior tree implementation.
Your goal is to implement the DSL given a specified tactic.

## Game Scenario
{{instructions.scenarios.cs}}

## Available Nodes
You are also given a collection of existing basic nodes in the game.
{{ instructions.actions.selector }}
{{ instructions.actions.sequence }}
{{ instructions.actions.condition }}
{{ instructions.actions.param }}
{{ instructions.actions.action }}

## Tactics
{{ state.tactics }}

## DSL Format
{{ instructions.format.dsl_syntax }}

## Response
{{ instructions.format.dsl_nlu }}

{% if history.dsl_tree %}
### History Format Errors
Here's your last DSL implementation and its corresponding format problems, you should improve
upon it and fix the format problems.

{{ history.dsl_tree }}
{{ history.message }}
{% endif %}
```