Generating Plausible Cheating Trajectories for Multi-Player Games using LLM-Generated Behavior Trees

Romina Abachi, Bettina Hein, Joshua Romoff
{romina.abachi,bettina.hein,joshua.romoff}@ubisoft.com

Ubisoft/Vantage Studios

Abstract

Cheating in online multi-player games harms player experience and developer revenue, yet data-driven cheat detection is difficult due to the scarcity of labeled data. In this early-stage idea, we propose generating synthetic cheater trajectories by having a Large Language Model (LLM) act as a meta-policy that composes behavior trees over a library of reusable skills, sidestepping the difficulty of end-to-end low-level control. Our two-stage framework first synthesizes human-like agents to populate the game, then refines a cheater policy that aims to win against this population while evading a fixed cheat detector.

1 Introduction

Cheating is widespread and pervasive in online multi-player games. If not addressed, it leads to lowered player enjoyment and engagement, becomes a driver of negative social media coverage and lost revenue for the game developer. Even in small numbers, cheaters can impact a large proportion of the player base, as a single cheater can encounter many legitimate players over just a few matches. In a recent survey, more than 60% of online gamers felt impacted by cheating (Irdeto, 2018).

At the same time, data-driven anti-cheat efforts face a persistent challenge in cheat detection: rule-based detectors are brittle, and learned detectors are limited by the scarcity of labeled examples of cheater trajectories. This scarcity is not merely a matter of data collection, but stems from the inherent difficulty of identifying cheating in the first place. Modern multi-player games unfold over long matches with rich, complex dynamics, where players employ a wide range of legitimate strategies that can superficially resemble cheating, and where skilled cheaters deliberately mimic legitimate play.

Even when a player is known to cheat, it is often unclear which of their matches were affected, and within a flagged match, precisely when the cheating occurred. Pinpointing the exact moments of cheating, for example a particularly accurate aim, a suspiciously timed reaction, or information a player should not have had, is difficult even for expert human reviewers, let alone at the scale required to train modern detection models. As a result, the labels available for supervised learning are typically coarse, noisy, and sparse, providing limited signal about the fine-grained behaviors that distinguish cheaters from legitimate players.

In addition, online video games are dynamic, and constantly changing to keep the game interesting. For example, in the First-Person Shooter (FPS) game *Rainbow Six Siege* (R6S), which is the focus of this proposal, new characters are added every few months, each with unique abilities and tools. Existing characters and environments are also constantly evaluated based on player feedback, and modified for fairness and compatibility with each other. As a result, player-collected gameplay trajectories become stale quickly, leading a learned model to make incorrect and unreliable predictions.

In this early-stage idea paper, we propose a framework to generate cheater trajectories that resemble human ones given a small amount of data, rather than relying only on collected player trajectories. This is a challenging problem because for the generated trajectories to convincingly resemble those of a real cheater, all other players, both teammates and opponents, must also behave realistically, employing complex strategies and cooperation over an extended horizon.

End-to-end imitation learning or reinforcement learning over primitive actions has historically struggled in such settings, succeeding only in a handful of high-profile efforts that required massive compute, carefully engineered training regimes, or restricted problem scope. Notable examples include OpenAI Five in Dota 2 (Berner et al., 2019), AlphaStar in StarCraft II (Vinyals et al., 2019), and more recent work on Counter-Strike (Pearce & Zhu, 2022) and Gran Turismo (Wurman et al., 2022), each of which demanded substantial resources and game-specific design choices that are impractical to replicate per game, let alone per cheating behavior of interest.

Instead, we sidestep the difficulty of learning low-level control from scratch by having an LLM operate at the level of structured policies over reusable skills, while leaving execution to those skills. Concretely, we propose a two-stage framework, shown in Figure 1, in which an LLM Orchestrator first synthesizes human-like behavior trees to populate a game with realistic non-cheater agents. A gameplay evaluation module assesses the performance and authenticity of the resulting trajectories against a curated dataset of human play.

In the second stage, specific "cheating" skills are added to the skill library, and the LLM Orchestrator refines a separate cheater policy that must achieve high task return while remaining competitive against the surrounding human-like population, all the while avoiding detection by a fixed cheat-detection module. The resulting cheater trajectories provide labeled, interpretable data for downstream cheat-detection systems.

2 What is Cheating?

A "cheat" is illicit software that gives its user an unfair advantage in a match. We focus on a class of information-providing cheats known as Extra-Sensory Perception (ESP), though our approach generalizes to other cheat types. ESP cheats reveal privileged information to the player, most commonly the position or skeleton of opponents through walls and other obstructions. Armed with this information, a cheater can anticipate enemy movements, pre-fire before opponents have a chance to react, and play more efficiently against the game's time pressure, for example by ignoring angles that are known to be safe or rushing objectives without the usual caution.

Note that our framework operates entirely at the level of behavioral telemetry, and for this proposal, we do not model client-side memory tampering, network-layer attacks, or hardware cheats below the action-API abstraction, though these may be parts of the wider anti-cheat ecosystem of a game.

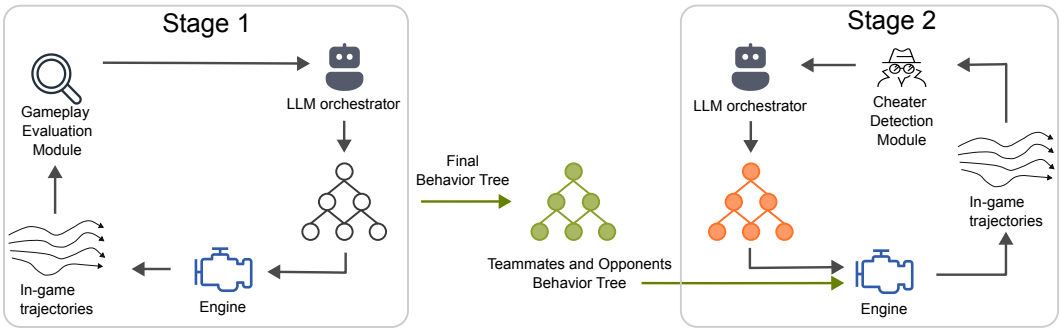

Figure 1: The two stages of our proposed cheater simulation approach.

3 Proposal

In commercial game AI, Behaviour Trees (BTs) (Isla, 2005) are the dominant control formalism, capable of expressing complex, long-horizon decision-making through modular and reusable subtrees while remaining interpretable and easy for designers to create, debug, and modify.

A BT b is a directed rooted tree where the internal nodes are control-flow operators (e.g., sequence, selector, decorator) and the leaves are execution nodes which invoke specific skills. At each tick, the tree is traversed from the root, with control-flow nodes deciding which leaf nodes to execute based on their outcomes, yielding reactive yet structured behavior (Colledanchise & Ögren, 2018).

The leaves draw from a fixed library of primitive skills $\Omega = \{\omega_1, \ldots, \omega_K\}$, including navigation, aiming, target selection, and item-use routines. These skills can be viewed as options (Sutton et al., 1999) with their own initiation conditions, internal policies, and termination criteria, and may be scripted or pretrained.

We propose to create and refine BTs through an LLM-based Orchestrator. Rather than producing low-level actions directly, the LLM acts as a meta-policy $\pi_{\text{LLM}} : \mathcal{H} \to \mathcal{B}$ that maps a context $h \in \mathcal{H}$ to a BT $b \in \mathcal{B}$ over the skill set Ω, with the skills themselves handling low-level execution (e.g., "go to target"). The context h captures the target agent role (e.g., human-like player or cheater), the available skills, and any prior BT attempts along with their evaluation feedback. The same Orchestrator is reused across both stages with role-specific contexts, and learning happens in-context via iterative refinement rather than weight updates.

3.1 Stage 1: Synthesizing Human-Like Population Behavior

In this stage, the goal is to be able to produce a policy that resembles human gameplay behaviour. This stage establishes the environment in which the cheater will subsequently operate. Since R6S is a multi-player game, we need to model every teammate and opponent of the cheater introduced in Stage 2. A research question to explore here is whether to use a single BT that generates actions for all players, or to use a separate BT for each player. For this proposal, we assume the latter case.

As shown in Figure 1, once a modified BT is provided by the Orchestrator, it is executed in the game to generate multiple trajectories. This is similar to Monte Carlo rollouts (Sutton & Barto, 1998), where instead of providing value estimates, we use a Gameplay Evaluation Module to provide verbal feedback as well as task-specific values such as match score, number of kills/deaths, whether the bomb was defused, etc. This type of feedback can be more informative than scalar value estimates, as observed by Shinn et al. (2024), and with a memory of previous trajectories and feedback, can help the Orchestrator improve the proposed BT in the next step.

We also include a human-likeness discriminator, defined as $D_{\text{imitation}}(\tau_{b_k}, \tau_{\text{player}}) = \mathcal{D}\big(\rho_{\tau_{b_k}} \;||\; \rho_{\tau_{\text{player}}}\big)$, where $\tau_{b_k}, \tau_{\text{player}}$ are the sets of trajectories generated by b_k (the BT at iteration k) and real collected player data, respectively, ρ denotes occupancy measures and $\mathcal{D}$ is a divergence measure. This is a GAIL-style discriminator (Ho & Ermon, 2016), to encourage the BT trajectories to be indistinguishable from player data. For this version of the work, we do not use a parameterized discriminator that can be learned during the feedback loop, but that can be a future extension.

The Orchestrator iteratively refines candidate trees $b_k \sim \pi_{\text{LLM}}(\cdot \mid h_k)$ using rollout feedback until improvements to in-game performance cannot be made and $D_{\text{imitation}}$ can no longer distinguish between generated and collected trajectories, or a limit is reached. This BT is frozen before Stage 2.

3.2 Stage 2 — Iterative Refinement of the Cheater Policy

In Stage 2, we instantiate $n - 1$ agents, n being the number of players in a match, each with a respective BT from Stage 1, and a single cheater agent controlled by an initial BT $b_c^0 \in \mathcal{B}$, that gets refined by the Orchestrator. The cheater has access to the same skill library Ω, plus primitives that include extra inputs for perception of normally hidden states which we define as Ω_{cheat}. Note that

this naturally extends to other forms of cheats than ESP by modifying the skill library to match the advantages provided by a specific cheat.

The cheater's objectives are largely the same as the players in Stage 1, with the additional goal of remaining undetected by the anti-cheat of the game. Thus, we propose to provide the same feedback to the Orchestrator as in Stage 1, extended by including a frozen discriminator that has been trained on labeled cheater and non-cheater trajectories. Since this discriminator's only purpose is to encourage stealthy behaviors, it does not need to be trained on a large labeled dataset. The combined evaluation and feedback providers are denoted as the Cheat Detection Module in Figure 1. Adversarial co-training of the discriminator can be left as future work (Park et al., 2026).

4 Related Works

A growing line of work uses LLMs as high-level controllers for embodied or game agents: SayCan (Ahn et al., 2022) grounds LLM proposals via learned affordances, Code-as-Policies (Liang et al., 2023) treats program synthesis as policy specification, and Voyager (Wang et al., 2023) maintains a growing skill library in Minecraft via LLM-generated code with environment feedback. We share this LLM-as-meta-policy view but use behavior trees as the output structure, treating the LLM as a proposal distribution over BTs rather than a free-form code generator. Reflexion (Shinn et al., 2024), Self-Refine (Madaan et al., 2023), and Tree of Thoughts (Yao et al., 2023) show that LLMs can improve their outputs through verbal feedback loops, with Voyager (Wang et al., 2023) extending this to environment-grounded feedback. We frame iterative refinement explicitly as black-box policy iteration over a program-structured action space, where in-context conditioning plays the role of policy improvement and rollout returns serve as value estimates.

Several recent works use LLMs to generate BTs from natural-language specifications, primarily for robotic control. Lykov & Tsetserukou (2024) and Cao & Lee (2023) take open-loop approaches, finetuning on a fixed dataset and generating primitive tasks with a frozen LLM, respectively, and Merino-Fidalgo et al. (2025) similarly synthesize BTs from textual descriptions. Paduraru et al. (2025) target video games but focus on natural-language authoring and BT validation rather than in-game performance.

Closer to our setting, Xu et al. (2025) achieve strong performance in a multi-player FPS by having an LLM generate BT-structured policies in a Domain-Specific Language and closing the loop with in-game rollouts and Reflexion-style refinement. Our Stage 1 mirrors this design but additionally (i) incorporates adversarial teammate and opponent dynamics, and (ii) targets cheater gameplay specifically. COS-PLAY (Wu et al., 2026) goes further by co-evolving the skill library itself, a promising direction for cheater BTs where the relevant skill set is not known a priori. Unlike all of the above, our goal is not to maximize task performance but to generate trajectories emulating a specific player class, cheaters, for downstream study.

5 Limitations and Extensions

Cheater behavior may vary significantly across skill levels. Lower-skill cheaters may be easier to identify, as the gap between their baseline ability and their cheating behavior can be pronounced, making it harder for them to blend in. To address this issue, we can study how different skill settings influence both cheating strategies and detectability, e.g. by incorporating skill tiers derived from data or adjusting available player actions. Additionally, the current setup assumes no skill or playstyle diversity within a match, treating all players as operating at the same level, whereas real matches can exhibit meaningful differences that affect gameplay dynamics and match outcomes.

Typically, in real matches, players can report teammates and opponents if they suspect them to be cheating, making it easier for them to be detected by data-driven cheat detection methods. To take this into account, we could train our cheat detection module not only on player trajectories but also on available meta information, such as player reports and squad information.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal JĂłzefowicz, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Yue Cao and CS Lee. Robot behavior-tree-based task generation with large language models. *arXiv preprint arXiv:2302.12927*, 2023.

Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Irdeto. Irdeto global gaming survey: The last checkpoint for cheating, 2018. URL https://irdeto.com/hubfs/resources/reports/global-gaming-report-2018.pdf.

Damian Isla. Managing complexity in the halo2 ai. In *Game Developer's Conference Proceedings, 2005*, 2005.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Artem Lykov and Dzmitry Tsetserukou. Llm-brain: Ai-driven fast generation of robot behaviour tree based on large language model. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pp. 392–397. IEEE, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Sergio Merino-Fidalgo, Celia Sánchez-Girón, Eduardo Zalama, Jaime Gómez-García-Bermejo, and Jaime Duque-Domingo. Behavior tree generation and adaptation for a social robot control with llms. *Robotics and Autonomous Systems*, pp. 105165, 2025.

Ciprian Paduraru, Miruna Paduraru, and Alin Stefanescu. Enhancing game ai behaviors with large language models and agentic ai. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pp. 286–296, 2025.

Inkyu Park, Jeong-Gwan Lee, Taehwan Kwon, Juheon Choi, Seungku Kim, Junsu Kim, and Kimin Lee. Adversarial reinforcement learning framework for ESP cheater simulation, 2026. URL https://openreview.net/forum?id=ZBpFEtLw3H.

Tim Pearce and Jun Zhu. Counter-strike deathmatch with large-scale behavioural cloning. In *2022 IEEE Conference on Games (CoG)*, pp. 104–111. IEEE, 2022.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv. org/abs/2303.11366*, 8, 2024.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.

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.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

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. *arXiv preprint arXiv:2305.16291*, 2023.

Xiyang Wu, Zongxia Li, Guangyao Shi, Alexander Duffy, Tyler Marques, Matthew Lyle Olson, Tianyi Zhou, and Dinesh Manocha. Co-evolving llm decision and skill bank agents for long-horizon tasks. *arXiv preprint arXiv:2604.20987*, 2026.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

Zhongwen Xu, Xianliang Wang, Siyi Li, Tao Yu, Liang Wang, Qiang Fu, and Wei Yang. Agents play thousands of 3d video games. *arXiv preprint arXiv:2503.13356*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.