Transplorer: An LLM-Orchestrated Harness for Solving Commodore 64 Games with Hard-Coded, Tabular, and Distilled Neural Policies

Pannous
Independent Research

September 8, 2026

Abstract

We describe and evaluate Transplorer, a working system that wraps a live Commodore 64 emulator (VICE) as a uniform Gymnasium-API environment and uses a large-language-model-driven agent loop to play, understand, and solve unmodified commercial C64 games. Rather than committing to one algorithm family, the agent chooses per game among hard-coded Python policies, tabular or evolved neural policies, and LLM-derived logic, using Go-Explore-style frontier exploration [Ecoffet et al., 2021] to discover game structure and DAgger-style distillation [Ross et al., 2011] to compile expensive policies into fast networks, gated by a held-out promotion test before any policy is trusted. No component algorithm here is new; what we evaluate is the composition and its results on a real corpus. We find no prior academic work targeting Commodore 64 specifically as an RL benchmark platform, in contrast to the well-established Atari/ALE [Bellemare et al., 2013] and Gym Retro [Nichol et al., 2018] lineages. On a standard continuous-control ladder (Gymnasium classic control and the MuJoCo suite) the system solves 10 of 11 tasks near-optimally, with one honestly reported failure (`Humanoid-v5`) and one deliberately deferred negative-transfer finding (`BipedalWalker-v3-hardcore`). On an initial corpus of 89 attempted C64 titles, results are mixed and reported without inflation: several games are fully solved, most attempted games are partly solved with quantified gaps, and a large enumerated backlog remains untried. We additionally contribute and test a concrete improvement to the system's promotion gate — a sequential, variance-adaptive stopping rule — and show on real Blackjack data that the system's existing fixed-episode-count gate can report confident-looking promotion verdicts that are not actually statistically justified, a genuine and previously undetected reliability gap that our gate closes.

1 Introduction

Reinforcement learning benchmarks built on emulated video games have a long history, from the Arcade Learning Environment's wrapping of Atari 2600 titles [Bellemare et al., 2013] to Gym Retro's extension to SNES and Sega Genesis titles [Nichol et al., 2018]. These platforms standardize a Gym-style reset/step/observation/action interface around an emulator, letting a wide variety of algorithms be compared on a common corpus of real, unmodified commercial games. To our knowledge, no comparable platform exists for the Commodore 64, despite it hosting one of the largest surviving commercial game libraries of the 8-bit era.

Transplorer wraps the VICE C64 emulator (`x64sc`) as a Gymnasium environment, `ViceC64Env`, exposing four observation modes (raw RAM, text/glyph-grid, screen-plus-color grid, and rendered RGB frames) and a six-field action vector covering both joystick ports and the keyboard. On top of

this environment, an LLM-driven agent loop performs affordance discovery on an unfamiliar game, forms hypotheses about its mechanics, and then constructs a policy using whichever representation seems most appropriate: a hard-coded Python heuristic or planner, a tabular or evolved neural policy, or LLM-generated control logic. A stated design principle (worked out over the project's development, not imposed a priori) is that execution speed favors compiled/hard-coded policies while generalization favors LLM-driven ones, and that policies should be promoted or distilled between representations as evidence warrants, rather than committing to one algorithm class from the start.

This paper makes no claim that any individual algorithmic component is new. Go-Explore-style frontier exploration [Ecoffet et al., 2021], DAgger-style imitation distillation [Ross et al., 2011], tabular Q-learning with novelty bonuses, and LLM-agent code-generation loops [Wang et al., 2023, Ma et al., 2023] are all established techniques. Our contribution is descriptive and empirical: we report what this specific composition achieves on a real, demanding, and previously unaddressed benchmark domain, honestly including its failures and gaps, and we contribute one concrete, tested improvement to its evaluation methodology.

2 System Description

Environment. `harness/vice_env.py` implements `ViceC64Env`, a standard `gymnasium.Env` around a live VICE process. An action is the C64's full input surface, expressed as six numbers: two joystick ports and a keyboard key, each with a hold duration that can be specified in frames (stepped/raster timing, where the CPU is halted between frames so there is no real-time deadline) or in seconds (wall-clock timing, where the game runs at native speed and the agent must keep up).

Exploration. `harness/go_explore.py` and `harness/frontier_archive.py` implement a Go-Explore-lite mechanism [Ecoffet et al., 2021]: promising, previously-reached states are archived and can be returned to, so exploration can resume from a discovered frontier rather than restarting from scratch, and partial policies can be derived directly from the resulting explored-state graph.

Policy representations and distillation. `harness/tabular_q.py` implements tabular Q-learning with a tuned novelty bonus; `harness/lqr.py` provides linear-model control for continuous-control (MuJoCo) tasks; `harness/dagger.py` implements standard DAgger [Ross et al., 2011]: round zero is behavior cloning from an expert, and each subsequent round rolls the current network out, relabels the states it visits with the expert's action, and retrains on the aggregated dataset — the mechanism by which an expensive policy (a hard-coded heuristic or, in principle, an LLM-driven controller via `harness/coroutine_policy.py`) is compiled into a fast network only when speed is actually needed.

Promotion gate. Before any distilled or trained policy is trusted, `harness/policy_runner.evaluate_for_prom` trains and measures performance on one batch of episodes, then re-measures on a disjoint held-out batch, requiring both to clear a task-specific threshold. This is deliberately conservative evaluation-hygiene rather than a novel algorithm, but as Section 5 shows, its fixed-batch-size version has a real, previously undetected reliability gap.

Epistemic tooling. `harness/cheat.py` and `harness/disasm.py` provide live and static 6502 disassembly, used strictly to understand a game's true reward and collision logic during development; by explicit project convention (`cheating.md`) this information may not be used inside a

game's final demonstrated policy, and this boundary is enforced by convention and audited in the notes we sampled.

3 Related Work

Emulator-based game benchmarks. The Arcade Learning Environment [Bellemare et al., 2013] and the DQN result that made it famous [Mnih et al., 2015] established the template of wrapping an emulator as a uniform RL interface across dozens of real, unmodified games. Gym Retro [Nichol et al., 2018] and the Retro Learning Environment [Bhonker et al., 2016] extend this to SNES/Genesis-era titles, with an explicit focus on generalization across levels. A targeted search found no published Commodore 64/VICE-based Gymnasium-style RL benchmark or agent corpus in the RL literature; the only adjacent project we found, an LLM tool-chain for *writing* 6510 assembly games with VICE feedback, addresses game authoring, not RL environment control. We report this as a genuine platform gap rather than an under-citation.

Exploration. Go-Explore [Ecoffet et al., 2021] solves notoriously hard-exploration Atari games (Montezuma's Revenge, Pitfall) via a remember-states, return-then-explore strategy; Transplorer's frontier-archive mechanism is a direct, named implementation of this idea.

Imitation-based distillation. DAgger [Ross et al., 2011] corrects the compounding-error problem in naive behavior cloning by iteratively querying the expert on states the learner's own policy visits. `notes/dagger_distillation.md` reports a measured result on LunarLander: plain behavior cloning collapses (train reward -150.5, held-out -179.8, both below the promotion threshold and correctly refused), while DAgger repairs this (round-zero 148.9 $\rightarrow$ round-three peak 265.3 $\rightarrow$ round-four 251.9, against an expert baseline of 267.0), passing an automated test that requires DAgger to beat behavior cloning by more than 50 reward.

LLM agents for control and code generation. Voyager [Wang et al., 2023] uses an LLM-driven agent with an automatic curriculum and a growing library of executable skill code in Minecraft; Eureka [Ma et al., 2023] uses an LLM to write and evolve reward-function code for RL training; Code as Policies [Liang et al., 2022] has an LLM emit executable control code directly. Transplorer's LLM-orchestrated selection and construction of per-game policies is closest in spirit to Voyager's curriculum-and-skill-library pattern, applied to an emulated-game corpus rather than an open-world game.

World models. `harness/world_map.py` maintains an explicit, discovered-state graph/atlas rather than a learned latent dynamics model in the sense of World Models [Ha and Schmidhuber, 2018] or MuZero [Schrittwieser et al., 2020]; we are careful in this paper to describe it as such rather than borrow the stronger claims of that literature.

4 Results

4.1 Classic control and continuous control

The non-C64 Gymnasium ladder — CartPole, Acrobot, LunarLander (discrete and continuous), Pendulum, MountainCar (discrete and continuous), Taxi, FrozenLake 8x8, CliffWalking (with and without slip), Blackjack, several custom gridworlds, and the full MuJoCo continuous-control suite

(Inverted (Double) Pendulum, Reacher, Pusher, Swimmer, HalfCheetah, Hopper, Walker2d, Ant, HumanoidStandup, Humanoid, and BipedalWalker with and without the `hardcore` variant) — is solved end-to-end for 10 of 11 MuJoCo tasks, several at measured or provable optimality via value-iteration cross-checks. The one clear failure is plain `Humanoid-v5`: linear and cross-entropy-method policies plateau around a return of 345 against a 500 solved-threshold, and the project's own notes flag this honestly as the one environment genuinely requiring a full deep-RL method such as PPO or SAC rather than the harness's lighter-weight policy classes. `BipedalWalker-v3-hardcore` is explicitly left untrained after measuring that a strong flat-terrain policy transfers *worse* to the hardcore variant than a weak one — a real, reported negative finding about generalization rather than a masked failure.

4.2 Commodore 64

89 task folders exist under active development, against a much larger enumerated backlog of untried ROM titles — an honest large backlog, not a hidden claim of broad coverage. Of the attempted titles: several are cleanly solved (Tetris, ranked first in the game's own in-ROM high-score table at 6788 points; Astral Attack; one full scene of Dare Devil Denis); most are partly solved with quantified, non-cherry-picked gaps (Boulder Dash clears 12 of 16 caves, with cave 13 at 4 of 6 sub-goals; Pac-Man survives complete games but never clears a board; Q*bert routes correctly but never clears a level; Pengo clears rounds but never finishes a game); and a few are explicitly reported as unsolved or blocked (a geometrically judged uncrossable pit in one title; several dead-ends from broken ROM cracks or software lockups, recorded so future attempts do not repeat them). The project's notes repeatedly flag and correct benchmark-methodology traps — in-process replay inflating measured scores, single-trajectory results being rejected in favor of independently reproduced runs — which we take as evidence of a genuinely self-critical measurement culture rather than a polished showcase.

5 Our Contribution: A Sequential, Variance-Adaptive Promotion Gate

5.1 The gap

The existing promotion gate always draws a *fixed* number of held-out episodes, chosen in advance by whoever authored the task, and thresholds the resulting point estimate. This is a real, already-documented weakness: Blackjack's per-hand reward has a standard deviation of approximately 0.95, so a 2000-episode batch carries a standard error of about ± 0.021 — roughly four times the entire measured gap between a mediocre and a near-optimal policy on that task. A fixed-size gate has no built-in way to know, after the fact, whether its own batch was ever large enough to answer the question it was asked.

5.2 Method

`probes/sequential_promotion_gate.py` (purely additive; it does not modify `harness/policy_runner.py` or any existing task or test) implements a sequential evaluation procedure: episodes are run in batches of 200, tracking the running mean and standard error, and evaluation stops as soon as $|\text{mean} - \text{threshold}| > 1.96 \times \text{SEM}$ (a 95%-confidence decision), up to a 200,000-episode cap; if no confident decision is reached, the procedure reports `decided=False` rather than a false-confidence

Policy	Method	n episodes	Mean	±SEM	Decided	Promoted	Seconds
random	sequential	200	−0.3700	0.0651	Yes	No	2.4
random	fixed (2000)	2000	−0.3915	0.0201	Yes	No	11.4
basic_strategy	sequential	**27,200**	−0.0449	0.0058	**Yes**	Yes	207.4
basic_strategy	fixed (2000)	2000	−0.0320	0.0214	**No**	Yes	13.5
q_table	sequential	**32,200**	−0.0459	0.0053	**Yes**	Yes	266.2
q_table	fixed (2000)	2000	−0.0310	0.0214	**No**	Yes	13.8

Table 1: Real, measured results on `Blackjack-v1`, against threshold −0.05656.

pass or fail. We compare this against a `fixed_n_evaluate` function that reproduces the harness's own 2000-episode default gate for a like-for-like comparison.

5.3 Experiment

We ran both procedures against the real `Blackjack-v1` task and its three already-shipped policies (a random baseline, the trained tabular-Q policy, and the hand-coded basic-strategy policy), against the task's real solved threshold of −0.05656.

5.4 Honest interpretation

For the clearly bad policy (random), the sequential gate is simply more efficient, reaching a confident rejection in 200 episodes rather than 2000, because the gap to threshold is large relative to the noise — a real but unglamorous efficiency gain.

For the two near-ceiling policies, the result is more consequential: they required 27,200 and 32,200 episodes respectively — 13–16× the harness's fixed default — before the sequential method would report a confident decision. The fixed-2000-episode batch, run on the identical policies against the identical threshold, reported `decided=False` in both cases under the same statistical test: its point estimate happens to land above the threshold, but its own confidence interval (±0.0214) is wide enough to also be consistent with failing the threshold. Both gates ultimately agree on the final promote/reject label here, but for the wrong reason on the fixed side: *the existing fixed-episode promotion gate is currently capable of returning a confident-looking "promoted" verdict on evidence that does not actually support that level of confidence.* A different random seed for the same 2000-episode batch could plausibly have produced a mean below threshold and rejected a policy that the larger, ground-truth run confirms genuinely clears the bar.

This improvement is not free: an honest answer took roughly 3.5–4.5 minutes of wall-clock time per near-ceiling policy, versus about 13 seconds for the fixed batch. We therefore do not propose replacing the fast default gate universally, but propose a cheap, targeted improvement suggested directly by this measurement: compute the confidence interval the existing fixed-size batch already implies, and only escalate to the more expensive sequential procedure when that interval actually straddles the threshold. Applied to this data, that rule would have caught, using real numbers rather than a hypothetical, that Blackjack's two near-ceiling promotion verdicts were resting on intervals wide enough to have gone the other way.

6 What This Work Can and Cannot Claim

We can claim: a working, uniform Gym-API C64 harness spanning dozens of unmodified commercial titles; a demonstrated LLM-orchestrated pipeline that selects among and distills between

policy representations, with measured before/after numbers for that distillation; a promotion-gate methodology that has caught a real overfitting failure in practice; near-ceiling solves on 10 of 11 standard continuous-control tasks; and one concrete, tested improvement to the promotion gate's statistical reliability, demonstrated on real data.

We cannot yet claim that the C64 corpus is solved in aggregate — most attempted titles remain partly solved with quantified gaps, and the enumerated backlog of untried titles is far larger than the attempted set — nor that `world_map.py` constitutes a learned world model in the MuZero/World-Models sense, nor any general algorithmic contribution beyond the specific promotion-gate improvement in Section 5. The system's core value, on the evidence gathered here, is a systems and methodology contribution: the compositional, LLM-orchestrated application of several known techniques to a real, demanding, and previously unaddressed benchmark domain.

Code Availability

The full harness, all task folders, notes, and the probe implementing the promotion-gate improvement in Section 5 are available at `https://github.com/pannous/transplorer`.

References

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. arXiv:1207.4708.

N. Bhonker, S. Rozenberg, and I. Hubara. Playing SNES in the retro learning environment. arXiv:1611.02205, 2016.

A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. First return, then explore. *Nature*, 590:580–586, 2021.

D. Ha and J. Schmidhuber. World models. arXiv:1803.10122, 2018.

J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *ICRA*, 2023. arXiv:2209.07753.

Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. arXiv:2310.12931, 2023.

V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in RL. arXiv:1804.03720, 2018.

S. Ross, G. J. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

J. Schrittwieser, I. Antonoglou, T. Hubert, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020.

G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. arXiv:2305.16291, 2023.