NETHACK+: A CONFIGURABLE ENVIRONMENT FOR LONG-HORIZON AGENT RESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

NetHack remains a challenging benchmark for long-horizon decision-making despite extensive research employing reinforcement learning, imitation learning, and language-model agents. A substantial difficulty in NetHack derives from the lack of infrastructure surrounding the game. As a result, we present NetHack+, an upgraded game engine with 42 knobs for varying game difficulty, $14\times$ faster throughput than NetHack Learning Environment (NLE), and integrations into Gym, PufferLib, and Prime Intellect's Verifiers. We accomplish this by refactoring NLE to allow saving and restoring game state, configurable game mechanics, and custom observation and action interfaces embedded in modern LLM harnesses (Prime Agent and Claude Code). We also package a new metric $\mathrm{BAL}_{\min}$ to reward balanced progression in dungeon depth and experience, motivated by successful human runs and designed to limit rewards for agents that descend without gaining experience. Experimentally, interface improvements and continual harness optimization yield modest gains in $\mathrm{BAL}_{\min}$, reaching approximately $1.5\times$ baseline performance. To demonstrate the utility of checkpointing, we introduce Prime-agent-explore, an orchestrator-based search algorithm that selects saved checkpoints and guides subsequent exploration. Using 30 attempts with full vision, Prime-agent-explore reaches 11.38% $\mathrm{BAL}_{\min}$ and 34.66% $\mathrm{BAL}_{\max}$, improving over the single-life baseline by $8.6\times$ and $3.5\times$, respectively. With fog-of-war, Prime-agent-explore reaches 6.03% $\mathrm{BAL}_{\min}$ and 30.93% $\mathrm{BAL}_{\max}$, exceeding the previously reported LLM $\mathrm{BAL}_{\min}$ baseline by $3.3\times$. We also evaluate Prime-agent-explore on MiniHack, Crafter, and TextWorld, providing state-of-the-art performance across all three benchmarks.

1 INTRODUCTION

For decades, games have served as benchmarks for machine learning, driving new architectures and better learning algorithms (Tesauro, 1994; Campbell et al., 2002; Mnih et al., 2013; Silver et al., 2016; Karten et al., 2025; ARC Prize Foundation, 2026). With rich, synthetic environments, games offer the ability to test agents on complex tasks while maintaining full visibility over the environment. Each game requires its own profile of skills, making them strong targets for measuring generalization of diverse abilities, including perception, planning, and control.

Substantial research has focused on applying language models to play games (Hu et al., 2025; Zhang et al., 2026a), with rapid progress being made over the past year. Language models now reach human-level performance on Pokémon battles (Karten et al., 2025; 2026a), Baba Is You (Cloos et al., 2024), and other games in BALROG (Paglieri et al., 2025). However, NetHack (Küttler et al., 2020) remains unsolved. At the NeurIPS 2021 NetHack Challenge (Hambro et al., 2022a), none of the agents came close to completing the game, and the winning entry was symbolic. LLM-agent methods (Jeurissen et al., 2024; Paglieri et al., 2025) have matched deep-RL models but remain weaker than the state-of-the-art, neuro-symbolic bot AutoAscend (Sypetkowski & Sypetkowski, 2021).

Immediately, NetHack contains the intersection of many of the hardest classical reinforcement learning (RL) challenges, including limited visibility with overloaded glyphs, stochasticity in both game transitions and level generation, long-horizon navigation across procedurally generated floors, and a high-dimensional action space.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Beyond the game mechanics, poor infrastructure plays a large role in NetHack's difficulty. NetHack's user interface requires `vi`-style movement keys, and toggle-based menus waste token context and hide attributes of the NetHack environment that were never designed to be exposed programmatically (Matthews et al., 2026a). Furthermore, the NetHack Learning Environment (NLE) (Küttler et al., 2020) can only play one game per process since the game state consists of hundreds of global variables scattered across the code base. This prevents NLE from supporting checkpoints and state restoration and checkpoint-based search algorithms that use them.

Finally, evaluating progress is difficult when agents rarely win. The sparse reward has motivated research into intermediate progression metrics, including learned intrinsic rewards and weighted combinations of exploration, health, and other game statistics (Klissarov et al., 2024; Henaff et al., 2026; Matthews et al., 2026b). For language model evaluations, BALROG (Paglieri et al., 2025) is the prevalent progression metric. This metric, which we denote $\mathrm{BAL}_{\mathrm{max}}$, scores a trace by the higher of two human-derived ascension-probability scores, given a dungeon depth and experience level. However, this maximum allows agents to improve their score by descending without gaining experience. Our evaluations show that agents descend rapidly but die early (Figure 2), motivating a metric that rewards progress in both dimensions.

To modernize the infrastructure, we introduce NetHack+, a more flexible port of the NetHack game engine. Already integrated into `PufferLib` (Suarez, 2024; Sanders, 2026) and Prime Intellect `verifiers` (Senghaas et al., 2026), NetHack+ provides $14\times$ faster simulation speed than NLE, live-editing of gameplay constants (e.g., toggling off fog-of-war, adjusting health), and constant-time checkpointing. Within NetHack+, we integrate common LLM harnesses, such as Prime Agent and Claude Code, and use a default set of skills instead of single actions. Together, these refactors let us ablate the effect of perception and control inefficiencies on a model's planning capability. In one example, we test the hypothesis that locked doors waste agent actions by unlocking standard doors (Section 5.1). We also run baseline Continual Harness experiments, in which an orchestrator edits the agent's memories and skills between games (Section 5.2) (Karten et al., 2026c).

We propose $\mathrm{BAL}_{\mathrm{min}}$ as the lower of the two human-derived Dlvl and XL scores. This design is motivated by the consistent progression in both depth and experience observed in successful human runs (Figure 5). Four of the five interventions that increase $\mathrm{BAL}_{\mathrm{min}}$ decrease $\mathrm{BAL}_{\mathrm{max}}$, illustrating how the choice of metric changes the assessment of agent progress (Table 3).

To demonstrate the utility of checkpointing functions, we introduce a search algorithm Prime-Agent-Explore in NetHack for the first time, resulting in state-of-the-art performance. Furthermore, we test generalization to MiniHack, TextWorld, and Crafter, finding similar improvements (Section 5.3). We summarize our contributions below:

1. **NetHack+:** A more expressive game engine than NLE (42 extra knobs) with $14\times$ faster simulation speed and modern agent harness integrations.
2. **$\mathrm{BAL}_{\mathrm{min}}$:** A metric that rewards balanced progression in dungeon depth and experience.
3. **Prime-Agent-Explore:** A checkpointed exploration algorithm based on Go-Explore that achieves 11.38% $\mathrm{BAL}_{\mathrm{min}}$ with full vision and 6.03% $\mathrm{BAL}_{\mathrm{min}}$ with fog-of-war, $3.3\times$ the prior state of the art of 1.84% (Paglieri et al., 2025), and also provides state-of-the-art results on Crafter, TextWorld, and MiniHack.

2 Background Work

NetHack. NetHack, from 1987, is one of the oldest roguelike games. In the terminal game, a player descends to the lowest floor of a procedurally generated dungeon to retrieve the Amulet of Yendor and return to the surface without dying, battling hundreds of monsters and navigating environmental traps along the way. NetHack is challenging for humans and across the 46.8k human games from the NetHack Learning Dataset (NLD) (Hambro et al., 2022b), only 1.8% end in an ascension.

Popularized through the Gym environment NLE (Küttler et al., 2020) and the NeurIPS NetHack Challenge (Hambro et al., 2022a), NetHack has been studied with symbolic agents (Sypetkowski & Sypetkowski, 2021), deep reinforcement learning (Piterbarg et al., 2023; Henaff et al., 2026; Matthews et al., 2026b), imitation learning (Mazoure et al., 2023; Tuyls et al., 2024), and language models (Klissarov et al., 2024; Jeurissen et al., 2024; Paglieri et al., 2025).

NetHack Benchmarks. Despite minimal progress in NetHack since the NetHack Challenge, the infrastructure for the game has not improved, nor have easier, scaffolding variants been introduced. Only one other benchmark, MiniHack (Samvelyan et al., 2021), is built on NetHack. MiniHack designs custom NetHack dungeon levels with the same actions and observations, but with different objectives. SkillHack (Matthews et al., 2022) extends MiniHack with procedurally generated environments. However, RL models and LLMs have been able to saturate MiniHack, yet neither method transfers to NetHack (Paglieri et al., 2025). MiniHack fixes the data format and varies objectives in an environment. In contrast, in NetHack+, we fix the objective of NetHack, but add 42 game parameters that can be adjusted to make the game more approachable.

BALROG's Limitations. BALROG is a benchmark that measures LLM performance on a suite of games, including NetHack. It finds that models score under 7% on NetHack (Paglieri et al., 2025). However, models ranked on this leaderboard are evaluated in a fixed harness that limits their performance due to not being up to standard with modern harnesses. The largest two issues are the fixed history of 16 observations and the lack of agent skills that provide action chunking over NetHack vi commands. Research on popular and modern agent harnesses (Anthropic, 2025; Anysphere, 2025; OpenAI, 2025; DeepSeek-AI, 2026; Zechner & Earendil Works, 2026; Karten et al., 2026b) find that providing all observations rather than a fixed number better represents the capabilities of frontier models. Furthermore, use of skills has been found to enable cheaper and more performant agents. Recent benchmarks allow models to use specific user-provided harnesses, informing our decision to offer both Prime Agent and Claude Code as configurable harnesses in NetHack+.

Search-based Algorithms. Game-playing agents frequently use search to identify promising sequences of actions. These methods require the ability to save and restore states, an ability that underpins tree search in board games (Silver et al., 2016; 2018; Schrittwieser et al., 2020), exploration under sparse rewards (Mouret & Clune, 2015; Ecoffet et al., 2021), reverse curricula methods (Florensa et al., 2017; Resnick et al., 2018; Salimans & Chen, 2018), and skill libraries that accumulate reusable behaviors (Wang et al., 2023; 2024; 2025). Following similar work for agents (Dong et al., 2026; Zhang et al., 2026b), we adopt a framework similar to Go-Explore and its successors, returning to checkpoints before exploring onward (Ecoffet et al., 2021; Zhang et al., 2024; Lu et al., 2025).

3 METHODS

Formalism. We model NetHack as a partially observable Markov Decision Process (POMDP) (Sutton & Barto, 1998). A POMDP is defined as a tuple $\langle S, A, \Omega, P, O, R \rangle$, denoting the state space S, action space A, and observation space Ω. At time t with state s_t, an agent receives an observation $o_t \in \Omega$ drawn from the observation function $O(\cdot \mid s_t)$ and makes a decision $a_t \in A$. The environment then transitions to $s_{t+1} \sim P(\cdot \mid s_t, a_t)$. The environment also returns a reward $r_t = R(s_t, a_t)$. We write $h_t = (o_1, \ldots, o_t)$ for the observation history available at time t.

NetHack+ Engine. To support studies of long-horizon planning, we make three modifications to the game. We make features such as partial observability configurable through game flags, we expose difficulty via environment variables, and we programmatically expose game internals, enabling the development of custom observations and skills (as seen in Section 2).

In our refactor of NetHack's codebase of over 100,000 lines, we gather the mutable state into a single per-environment struct, with the most commonly modified fields placed on its entries to optimize for cache lines. Example user-set knobs that we include are vision toggles, fog-of-war range, health regeneration speed, map generation, luck ratios, experience level, and dungeon level. Table 1 notes a key set of functionalities, and we provide a full list with a detailed comparison to NLE and MiniHack in Appendix A.

Agent Interface. In NetHack+, we replace the agent harness in BALROG with the Claude Code (Anthropic, 2025) and Prime Agent (Karten et al., 2026b) command-line interfaces (CLIs), where the game is exposed through the Model Context Protocol (MCP). We define and default to using the NetCore set of skills, which are a reduced set of skills taken from Jeurissen et al. (2024). NetCore retains the expressiveness of the original vi controls while compressing up to 50 actions into a single LLM turn.

Capability	NLE	MiniHack	NetHack+
Full NetHack	yes	no (sub-tasks)	yes
Environments per process	1	1	thousands
Save, restore, branch game state	no	no	yes, $\mathcal{O}(1)$
Edit hero stats and depth at run time	no	task setup only	yes
Difficulty knobs (generation / character / live)	0	task-authored	8 / 4 / 30
Serialize and transplant a level	no	no	yes
Observation formats	1	1	12
Action surfaces	vi	vi	vi, NetPlay, NetCore, code

Table 1: **Infrastructure Comparison.** Capabilities of NLE, MiniHack, and NetHack+ game engines. Table 7 and Table 8 give the full comparison and knob list.

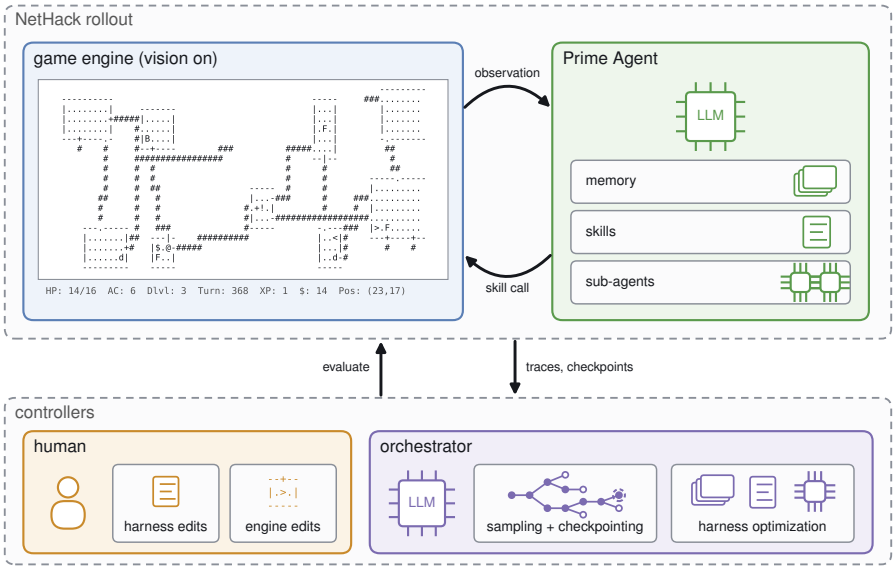


Figure 1: **NetHack Agent Benchmark Diagram.** Architecture to optimize the Prime Agent + NetCore harness on NetHack. **NetHack Rollout:** An agent interacts with the game engine while writing to and reading from its memory, skills, and sub-agent configurations. **Human:** Manual edits and ablations used to measure the difficulties caused by various game mechanics. **Orchestrator:** LLM-based algorithms for optimizing rollouts: Prime-agent-explore and Continual Harness.

NetCore consists of eight skills, each a small program over raw NetHack keys. np_move_to() navigates to a target tile using A* and emits movement keys in a single call. Other actions not covered by dedicated skills remain accessible by calling np_press_key() with NetHack's key binds (Table 9).

Our implementation differs from NetPlay (Jeurissen et al., 2024) in that it allows interrupts. In NetPlay, game prompts are dismissed by a skill internally, hiding the message from the agent. This substitutes the deterministic code's judgment for the model's, instilling a prior towards action. We instead abort at the first game message, ensuring the agent retains full knowledge and control.

Evaluation Metric. We introduce BALROG-min ($\mathrm{BAL}_{\min}$), defined as $\mathrm{BAL}_{\min}(d, x) = \min(p_d(d), p_x(x))$, where d and x are the maximum dungeon level (Dlvl) and experience level (XL) attained across a rollout. We use the human-derived progression mappings from Paglieri et al. (2025), where $p_d(d)$ and $p_x(x)$ estimate the probability of eventual ascension among human games that reach dungeon level d or experience level x, respectively. The standard BALROG metric, which we denote $\mathrm{BAL}_{\max}$, takes the maximum of these two scores instead.

This metric rewards balanced progression in dungeon depth and experience, as motivated by the joint progression observed in successful human runs (Figure 5). Under $\mathrm{BAL}_{\max}$, an agent can increase its score by descending while its experience remains low. Under $\mathrm{BAL}_{\min}$, once the experience-based score is the smaller component, further descent alone will not improve the metric.

Prime-Agent-Explore. We introduce Prime-Agent-Explore, a checkpointed exploration algorithm inspired by Go-Explore (Ecoffet et al., 2021; Lu et al., 2025). During gameplay, we save checkpoints containing both the game state and the sub-agent's conversation context. Between rollouts, an orchestrator receives a summary of the previous rollout and a list of all checkpoints, including their dungeon depth, experience level, health, hunger status, and game score. The orchestrator then selects a checkpoint to resume and provides a high-level plan to append to the sub-agent's context.

After each rollout, subsequent agents do not have access to the prior traces, tile locations, death causes, or item positions from that rollout beyond the context already stored in a selected checkpoint. Each sub-agent resumes with a summary of their checkpoint and the orchestrator's high-level plan. We restrict information sharing across attempts to the rollout summaries, checkpoint statistics, and high-level guidance available through the orchestrator. We also do not restore the RNG stream, so that an orchestrator cannot accumulate knowledge about their seed.

4 Experimental Setup

Standard Setup. We evaluate GPT-5.6 Sol and GPT-5.6 Luna, Gemini-3.7 Flash, Qwen-3.8 Max, and GLM-5.2 with Prime Agent + NetCore. We use GLM-5.2 for further ablation experiments. We compare our experimental results to previously published $\mathrm{BAL}_{\max}$ (and derived $\mathrm{BAL}_{\min}$) scores obtained with fog-of-war, but our default evaluation uses the full-vision setting. Figure 1 presents a diagram of our experiments.

Failure Mode Probes. We probe agents' planning, control, and perception skills by supplying elements of the agent scaffold that embody each of the capabilities and removing obstacles from the environment (Section 5.1). We emphasize that these results are a small subset of the experiments that are now possible with our new game engine. For direct comparability, each ablation is evaluated on 15 traces, consisting of 5 seeds with 3 repetitions each (Table 3).

Continual Harness Protocol. Each of the harness components described so far is designed and tested by hand. Next, we ask how the model can optimize its own harness by reflecting on its previous gameplay (Shinn et al., 2023; Zhao et al., 2024). We test three configurations of a GEPA-style (Khattab et al., 2024; Agrawal et al., 2026) train–reflect–evaluate wrapper over the Continual Harness functionality within Prime Agent. The first is a *memory-only* run, in which only memory entries are persisted between iterations (as opposed to editing NetCore). For the *wiki* run, the orchestrator edits the same memories, but is additionally given two NetHackWiki strategy pages on the early game. The *skills+wiki* run is given the wiki and can edit memory entries and NetCore skills. In each iteration, subagents play the game and propose memory edits, which an orchestrator then approves or modifies. Full details for each variation can be found in Appendix F.

Prime-Agent-Explore Evaluation. For each rollout, we save checkpoints every 20 LLM calls, at each change in Dlvl or XL, and upon 'save' skill calls. For our evaluations, we allot 30 rollout attempts, with a scaling study on one of the evaluation seeds using up to 100 rollout attempts. To better understand the algorithm, we ablate Prime-Agent-Explore variants that prevent the orchestrator model from steering agents, selecting checkpoints, and both. These ablations use 15 attempts (Table 5). To test generalization, we apply Prime-agent-explore to Crafter (Hafner, 2022), TextWorld (Côté et al., 2018), and MiniHack (Samvelyan et al., 2021).

5 Results

Overall Performance. We report our main evaluations in Table 2 and Figure 2. The evaluated models reach dungeon levels of up to 8 on average but only experience levels of 2–3. Qwen-3.8 Max achieves the highest baseline median $\mathrm{BAL}_{\min}$ of 2.12%, matching the human median. Our traces suggest the agents understand the objective and glyphs of the game, identifying monsters easily. However, this does not prevent them from catastrophic and irrevocable mistakes (e.g., taking losing

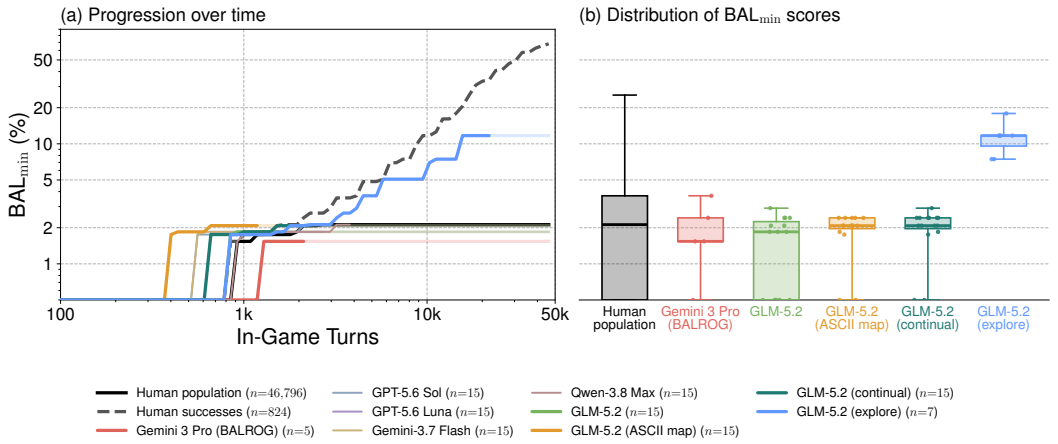


Figure 2: **Progress in NetHack. a)** $\mathrm{BAL}_{\mathrm{min}}$ against game turns for recent LLMs using the Prime Agent + NetCore, compared to human population, human successes, and prior SOTA (Gemini 3 Pro with BALROG harness). Each curve represents the running maximum of the median $\mathrm{BAL}_{\mathrm{min}}$ of living rollouts (i.e., we measure the highest median score that can be attained by the model within some number of in-game turns). **b)** Boxplots show the distribution of $\mathrm{BAL}_{\mathrm{min}}$ scores for human players, the BALROG SOTA, and our optimized harnesses.

		$\mathrm{BAL}_{\mathrm{min}}$		beats human	
Agent	n	mean	median	mean/median	pctile
Human	46,796	5.41 ± 0.06	2.12	—	52.4%
Gemini-3 Pro (BALROG)	5	1.84 ± 0.60	1.54	no / no	30.9%
Prime Agent + NetCore (GLM-5.2)	15	1.32 ± 0.30	1.85	no / no	41.6%
GPT-5.6 Sol	15	1.65 ± 0.23	2.08	no / no	45.3%
GPT-5.6 Luna	15	1.50 ± 0.25	1.85	no / no	41.6%
Gemini-3.7 Flash	15	1.47 ± 0.24	1.85	no / no	41.6%
Qwen-3.8 Max	15	1.74 ± 0.31	2.12	no / no	52.4%
Claude Code + NetCore (GLM-5.2)	15	1.57 ± 0.31	2.08	no / no	45.3%
+ harness fixes (GLM-5.2)*	15	1.88 ± 0.20	2.08	no / no	45.3%
+ continual/wiki (GLM-5.2)	15	1.96 ± 0.22	2.08	no / no	45.3%
+ explore (GLM-5.2, 30 attempts)	7	$\mathbf{11.38 \pm 1.33}$	**11.70**	yes / yes	92.0%
+ explore (GLM-5.2, 100 attempts)	3	$\mathbf{20.43 \pm 2.52}$	**17.91**	yes / yes	94.3%
+ explore (fog, GLM-5.2, 30 attempts)	5	6.03 ± 0.58	5.08	yes / yes	85.8%

Table 2: **Main Results.** We compare evaluations of the Prime Agent + NetCore with prior state-of-the-art evaluation results from BALROG, as well as human gameplay from NetHack Alt.org (NAO, Hambro et al. (2022b)) games that ran to a terminal state. We evaluate GLM-5.2, GPT-5.6 Sol, GPT-5.6 Luna, Gemini-3.7 Flash, and Qwen-3.8 Max with the Prime Agent + NetCore. *We report the best harness intervention that leaves the game itself unchanged, the ASCII map ablation of Section 5.1. 'Forced revive' performs better ($\mathrm{BAL}_{\mathrm{min}}$ at 2.64 ± 0.20), but fundamentally changes the game rather than the harness[1]. Fog denotes evaluation with fog-of-war enabled.

fights and descending on low health). Although we default to Prime Agent, we compare to Claude Code in Appendix G.

5.1 Environment and Interface Ablations

We consider three main capabilities for evaluating behaviors of LLMs. *Planning* focuses on formulating and pursuing goals beyond the next action, *control* is the ability to reliably interact with the

[1]Forced revive differs from Prime-agent-explore since revived agents retain information about their death, allowing them to memorize locations of mobs and simulate fighting mobs. Prime-agent-explore avoids this contamination.

game, and *perception* concerns identification of the game state from observations. We test model weaknesses by removing various obstacles representative of each capability.

Intervention	n	Dlvl	XL	BAL_{max}	BAL_{min}
Prime Agent + NetCore	15	8.0 ± 0.9	2.33	9.96 ± 2.19	1.32 ± 0.30
Advisory descent gate	15	6.5 ± 0.7	2.93	5.73 ± 1.17	1.79 ± 0.25
Forced revive on death	15	$\mathbf{10.3 \pm 0.8}$	**4.20**	$\mathbf{15.48 \pm 2.58}$	$\mathbf{2.64 \pm 0.20}$
Locked doors removed	15	7.3 ± 0.7	2.73	7.49 ± 1.67	1.76 ± 0.25
ASCII map every turn	15	7.7 ± 0.7	2.93	8.50 ± 1.66	1.88 ± 0.20
JSON map every turn	15	6.4 ± 0.7	2.13	5.70 ± 1.30	1.43 ± 0.23

Table 3: **Harness Interventions.** Using Prime Agent + NetCore with GLM-5.2 as the control, we show results with modifications to the harness and engine. Each row shows evaluations on a fixed set of 5 seeds $\times 3$ repetitions. BAL_{max} and BAL_{min} are aggregations of the human-derived Dlvl and XL progression scores. Directionally, ablations improve BAL_{min}, while four of the five lower BAL_{max}.

Decision-making Gates (planning). We add heuristic-based gates that interrupt tool calls and request confirmation when an agent attempts to descend while underleveled relative to successful human games, or attack an enemy while below one-third of maximum health. After receiving a warning, the agent declines the descent in 44% of cases, but gains an experience level before retrying in only 2% of cases. Overall, mean experience level rises from 2.33 to 2.93, leading BAL_{min} to increase from 1.32% to 1.79% (+36%).

Revival + Rollback (planning). Instead of making a gate prior to potential mistakes, we test out how a model responds to and recognizes the mistakes after they occur by providing an n-step rollback skill using state checkpointing. We also provide automated rollbacks upon death, where a run is now declared 'finished' upon the third death of a model in a single dungeon level. This leads to the best results under all metrics at depth 10.3 ± 0.8 and BAL_{min} 2.64 (+100%). Due to the success of these attempts, we extend this strategy in Section 5.3.

Unlocking Doors (control). To test the marginal benefit of improving control, we initialize all doors as unlocked[2]. This reduces one source of control friction, and we find that `np_kick()` calls fall from 4.9% to 3.9% of all tool calls. Affecting 1% of tool calls corresponds to an improvement to 1.76 BAL_{min} (+33%), suggesting that improving control is a very promising area of NetHack+.

Streamed Observations (perception). Our harness only refreshes dungeon maps when requested, to reduce redundant tokens[3]. However, we observed 9.7% of `melee_attack(x,y)` tool calls occurring on empty tiles (where a monster used to be). We test out providing the map on every turn. In this case, misfires fall to 2.4% when the map is formatted as ASCII and to 0% when a JSON map is provided. We give examples of map formats in Appendix B. JSON slightly improves BAL_{min} (+0.11%) while ASCII majorly improves BAL_{min} (+0.56%) respectively, suggesting that the observation format is also a large factor in NetHack+ performance.

5.2 Continual Harness Optimization

Table 4 reports the held-out evaluations for each configuration. In both reflection-only configurations BAL_{min} rises, but skill-editing capability decreases both BAL_{min} and BAL_{max}.

Continual Harness For both Memory-Only and Wiki-Based Reflection variants, the model focuses on correcting factual errors in its traces. The most frequent edits adjust numerical boundaries for guidance (e.g., modifying 'never descend 2 or more levels without XP gain' to 'never descend unless XL increased since last descent'). The model empirically only makes changes addressing one issue at a time to isolate the effect of its changes. Continual harness with memory results in BAL_{min} 1.55 ± 0.26. With wiki access during continual harness, BAL_{min} increases to 1.96.

[2]Standard doors are all marked as unlocked, but secret doors and doors on special floors remain randomized.

[3]The number of ASCII glyphs that change on any given turn is typically small (~ 5).

Condition	n	Dlvl	XL	BAL_{max}	BAL_{min}
Prime Agent + NetCore	15	8.00 ± 0.88	2.33 ± 0.36	9.96 ± 2.19	1.32 ± 0.30
Continual, memory-only	15	7.27 ± 0.87	2.47 ± 0.31	8.01 ± 1.98	1.55 ± 0.26
Continual, + wiki	15	5.27 ± 0.64	3.13 ± 0.31	3.73 ± 0.86	1.96 ± 0.22
Continual, + skills	5	3.20 ± 0.58	2.40 ± 0.68	2.00 ± 0.23	1.26 ± 0.52

Table 4: **Continual Harness Evaluations.** Each row denotes a specific setup for running Continual Harness for 10 iterations. Each cell reports the average as well as standard error of the mean. Dlvl and XL represent the maximum dungeon level and experience level attained during the run, and BAL_{max} and BAL_{min} represent the maximum and minimum BALROG progression metrics.

When allowed to edit NetCore, the orchestrator creates longer-horizon tools, including a skill intended to descend an entire dungeon level. Rollout agents abandon these skills after failures, limiting the feedback available for subsequent edits (Appendix D). This configuration achieves 1.26% BAL_{min}, compared with 1.96 for memory and wiki access without skill editing.

5.3 PRIME-AGENT-EXPLORE

a) Averaged over Five Seeds	n	BAL_{min}	BAL_{max}	Max Dlvl	Max XL
GLM-5.2, Prime Agent + NetCore	15	1.32 ± 0.30	9.96 ± 2.19	8.0 ± 0.9	2.3 ± 0.4
Prime-Agent-Explore, 30 attempts	7	$\mathbf{11.38 \pm 1.33}$	$\mathbf{34.66 \pm 3.19}$	$\mathbf{18.4 \pm 1.5}$	$\mathbf{8.6 \pm 0.3}$
b) Single-Seed Scaling					
GLM-5.2, single life	3	1.50 ± 0.76	16.31 ± 6.98	10.0 ± 2.5	2.7 ± 0.9
Prime-Agent-Explore, 30 attempts	3	12.36 ± 3.04	38.43 ± 1.26	20.3 ± 0.9	9.0 ± 0.6
Prime-Agent-Explore, 100 attempts	3	$\mathbf{20.43 \pm 2.52}$	$\mathbf{44.18 \pm 3.38}$	$\mathbf{23.3 \pm 1.5}$	$\mathbf{10.3 \pm 0.3}$
c) Prime-Agent-Explore Ablations					
No Selection or Guidance	5	2.45 ± 0.30	5.28 ± 1.86	6.0 ± 1.2	4.6 ± 0.4
No Selection	5	2.81 ± 0.99	5.50 ± 1.77	5.2 ± 1.6	5.2 ± 1.0
No Guidance	5	4.21 ± 0.35	15.88 ± 4.39	10.6 ± 1.4	6.4 ± 0.2
Prime-Agent-Explore, 15 attempts	5	$\mathbf{9.53 \pm 1.38}$	$\mathbf{21.56 \pm 4.35}$	$\mathbf{13.0 \pm 1.8}$	$\mathbf{8.4 \pm 0.4}$

Table 5: **Prime-Agent-Explore Results. a)** Averaged Prime-agent-explore results with 30 attempts as compared to single-life GLM-5.2 evaluations, on a fixed set of five evaluation seeds; **b)** We scale Prime-agent-explore to 100 attempts on one seed, showing the effect of scaling test-time compute. **c)** Ablations on the evaluation seeds without an orchestrator to provide guidance and select a checkpoint, with 15 attempts. All values are means $\pm$ standard error over runs.

In Table 5, Prime-agent-explore with GLM-5.2 beats the Prime Agent + NetCore by $8.6\times$ on BAL_{min} and $3.48\times$ on BAL_{max} using 30 attempts. It reaches 34.66 BAL_{max} and 11.38 BAL_{min} on average, corresponding to Dlvl 18.4 ± 1.5 with XL 8.6 ± 0.3. In Figure 2, we find that progress matches human progress through 10k in-game turns on BAL_{min} but still trails human performance as turns increase. The orchestrator observes hunger and health statistics when selecting checkpoints and guiding rollouts. An example can be found in Appendix E.

We scale the experiments to 100 rollouts on one of the evaluation seeds in Table 5, finding that performance improves with the larger budget in this single-seed comparison. At 100 attempts, dungeon level increases by 3.0 and XL by 1.3. These levels are consequential in the game, so both metrics increase significantly. The best run by BAL_{min} reaches 25.48 (Dlvl 23 at XL 11), while the deepest run reaches Dlvl 26 at XL 10, for a BAL_{max} of 50.68.

Suite	Crafter	TextWorld	MiniHack
n	10	30	40
Base	0.545	0.420	0.150
PAE	0.659	0.933	0.325
Δ	$+0.114$	$+0.514$	$+0.175$
p	0.004	$< 10^{-5}$	0.008

Table 6: **Prime-agent-explore Outside NetHack.** Evaluations on Crafter, TextWorld, and MiniHack.

Section c of Table 5 shows that removing guidance reduces BAL_{min} from 9.53 to 4.21, while removing selection and defaulting to resuming the last checkpoint reduces BAL_{min} to 2.81. Removing both yields 2.45. The highest mean performance

is attained when both are combined, but the ablation suggests that guidance is most effective when an orchestrator can already select a starting checkpoint.

We verify these results transfer to the fog-of-war version of NetHack, and attain an average of $30.93\% \pm 4.93$ BAL_{max} and $6.03\% \pm 0.58$ BAL_{min} across five seeds. For comparison, the highest published results stand at 6.77% BAL_{max} and 1.84% BAL_{min} as reported in (Paglieri et al., 2025). We report results on the three additional environments in Table 6. Compared to a single life, Prime-agent-explore statistically improves across all games.

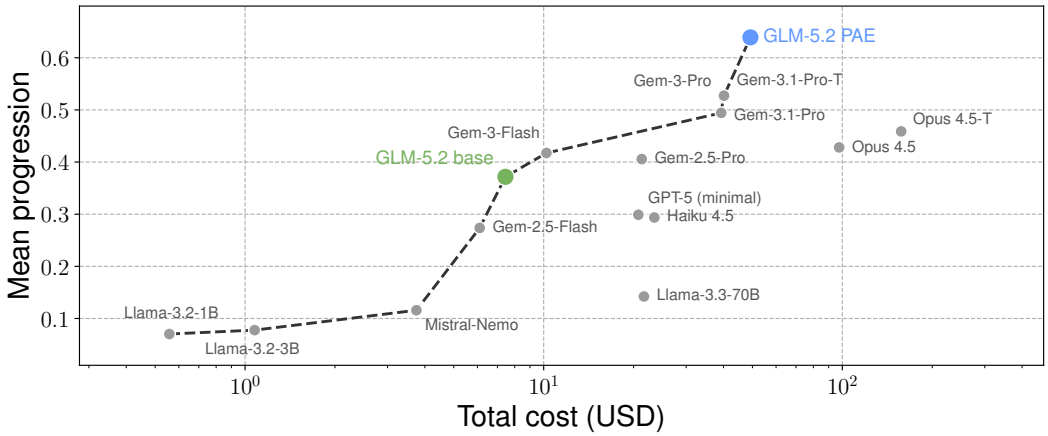


Figure 3: **Cost Against Performance.** Mean progression over MiniHack, TextWorld, and Crafter against total cost summed over the three games. The dashed line is the Pareto frontier.

6 Discussion

Both point-ablations and continual harness results suggest that the behavior of agents can be steered by inserting domain knowledge into the agent's context. Across harness experiments, we find that behavioral changes allow BAL_{min} to increase, yet decrease BAL_{max} (Table 3), validating the hypothesis that BAL_{max} over-indexes on short-term strategies while BAL_{min} does not. Figure 5 also shows that the models we evaluate overprioritize increasing Dlvl and diverge from the IQR of successful human traces early on, unlike Prime-agent-explore traces.

Future work may consider integrating a further optimized set of skills or skill-editing capabilities into the Prime-Agent-Explore algorithm to further improve performance. Another promising direction of work is to use Prime-agent-explore to generate high-quality data for imitation learning. In the past, imitation learning has been limited due to the low quality of AutoAscend data (Tuyls et al., 2024). With Prime-agent-explore, stronger traces could lead to much better models.

7 Conclusion

We introduce NetHack+, a configurable NetHack engine that supports checkpointing, custom observation and action interfaces, and modern agent harnesses. These capabilities enable controlled interventions on the game and checkpointed exploration through Prime-Agent-Explore. We also define BAL_{min}, a metric which rewards balanced dungeon depth and experience level. With Prime-Agent-Explore, GLM-5.2 reaches 11.38% BAL_{min} with full vision and 6.03% with fog-of-war, compared with the previously reported LLM baseline of 1.84%. We observe a 71% score increase when applying Prime-Agent-Explore to MiniHack, Crafter, and TextWorld, suggesting that checkpointed exploration generalizes across game environments. Together, NetHack+ and Prime-Agent-Explore provide infrastructure and a search method for continuing to study and improve long-horizon agent behavior on NetHack.

Ethics Statement

This paper presents work whose goal is to advance the field of Machine Learning. We contribute evaluation infrastructure and a search-based agent algorithm for NetHack, a single-player terminal game. All experiments run in a sandboxed game engine with no network access, no tool use that affects anything outside the harness or the game, and no interaction with people or external systems, so the direct risks of this research are minimal. Human reference statistics come from the public NLD (Hambro et al., 2022b).

Reproducibility Statement

The engine, the harness, the NetCore skills, the Prime-agent-explore orchestrator, and the evaluation scripts will be released after blind review.

AI Use Statement

In this work, we used generative AI tools for coding assistance. We have not used generative AI tools for drafting claims in this writing, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for generating figures, proofreading, and verifying citations. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *ICLR*, 2026.

Anthropic. Claude Code: An agentic command-line coding tool. https://www.anthropic.com/claude-code, 2025. URL https://claude.com/product/claude-code. Accessed: 2026-09-01.

Anysphere. Cursor CLI: Cursor's coding agent in the terminal. https://cursor.com/cli, 2025. Accessed: 2026-09-01.

ARC Prize Foundation. ARC-AGI-3: A new challenge for frontier agentic intelligence. *arXiv preprint arXiv:2603.24621*, 2026.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep Blue. *Artificial Intelligence*, 2002.

Nathan Cloos, Meagan Jens, Michelangelo Naim, Yen-Ling Kuo, Ignacio Cases, Andrei Barbu, and Christopher J Cueva. Baba is AI: Break the rules to beat the benchmark. *arXiv preprint arXiv:2407.13729*, 2024.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. TextWorld: A learning environment for text-based games. In *Computer Games Workshop at IJCAI*, 2018.

DeepSeek-AI. DeepSeek Harness: An open-source plugin-based agent runtime. https://github.com/deepseek-ai/deepseek-harness, 2026. Developer preview v0.1, MIT License. Accessed: 2026-09-01.

Yunpeng Dong, Jingkai He, Shiqi Liu, Yuze Hou, Dong Du, Zhonghu Xu, Si Yu, Baochuan Yang, Yubin Xia, and Haibo Chen. Deltabox: Scaling stateful AI agents with millisecond-level sandbox checkpoint/rollback. *arXiv preprint arXiv:2605.22781*, 2026.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 2021.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *CoRL*, 2017.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *ICLR*, 2022.

Eric Hambro, Sharada Mohanty, Dmitrii Babaev, Minwoo Byeon, Dipam Chakraborty, Edward Grefenstette, Minqi Jiang, Jo Daejin, Anssi Kanervisto, Jongmin Kim, et al. Insights from the NeurIPS 2021 NetHack challenge. In *NeurIPS Competition Track*, 2022a.

Eric Hambro, Roberta Raileanu, Danielle Rothermel, Vegard Mella, Tim Rocktäschel, Heinrich Küttler, and Naila Murray. Dungeons and data: A large-scale NetHack dataset. In *NeurIPS*, 2022b.

Mikael Henaff, Scott Fujimoto, Michael Matthews, and Michael Rabbat. Scalable option learning in high-throughput environments. In *ICML*, 2026.

Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P. Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. lmgame-bench: How good are llms at playing games?, 2025. URL https://arxiv.org/abs/2505.15146.

Dominik Jeurissen, Diego Perez-Liebana, Jeremy Gow, Duygu Cakmak, and James Kwan. Playing nethack with LLMs: Potential and limitations as zero-shot agents. In *CoG*, 2024.

Seth Karten, Andy Luu Nguyen, and Chi Jin. Pokéchamp: an expert-level minimax language agent. In *ICML*, 2025.

Seth Karten, Jake Grigsby, Tersoo Upaa Jr, Junik Bae, Seonghun Hong, Hyunyoung Jeong, Jaeyoon Jung, Kun Kerdthaisong, Gyungbo Kim, Hyeokgi Kim, et al. The pokeagent challenge: Competitive and long-context learning at scale. *arXiv preprint arXiv:2603.15563*, 2026a.

Seth Karten, Alex L. Zhang, Kevin Thomas, Sebastian Müller, Elie Bakouch, Daniel Auras, Mika Senghaas, Fares Obeid, Konstantin Dunas, Johannes Hagemann, and Sami Jaghouar. Prime agent: A self-improving RLM harness. *arXiv preprint arXiv:2608.23552*, 2026b.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026c.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. In *ICLR*, 2024.

Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. In *ICLR*, 2024.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The NetHack learning environment. In *NeurIPS*, 2020.

Cong Lu, Shengran Hu, and Jeff Clune. Intelligent Go-Explore: Standing on the shoulders of giant foundation models. In *ICLR*, 2025.

Michael Matthews, Mikayel Samvelyan, Jack Parker-Holder, Edward Grefenstette, and Tim Rocktäschel. Hierarchical kickstarting for skill transfer in reinforcement learning. In *CoLLAs*, 2022.

Michael Matthews, Pierluca D'Oro, Anssi Kanervisto, Scott Fujimoto, Jakob Foerster, and Mikael Henaff. Revisiting the nethack learning environment. In *ICLR Blog Track*, 2026a. URL https://iclr-blogposts.github.io/2026/blog/2026/revisiting-the-nle/.

Michael Tryfan Matthews, Anssi Kanervisto, Jakob Foerster, Pierluca D'Oro, Scott Fujimoto, and Mikael Henaff. Hierarchical behaviour spaces. *arXiv preprint arXiv:2604.24558*, 2026b.

Bogdan Mazoure, Jake Bruce, Doina Precup, Rob Fergus, and Ankit Anand. Accelerating exploration and representation learning with offline pre-training. *arXiv preprint arXiv:2304.00046*, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

OpenAI. Codex CLI: A lightweight coding agent for the terminal. https://github.com/openai/codex, 2025. Apache-2.0 License. Accessed: 2026-09-01.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wołczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *ICLR*, 2025.

Ulyana Piterbarg, Lerrel Pinto, and Rob Fergus. NetHack is hard to hack. In *NeurIPS*, 2023.

Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alexander Peysakhovich, Kyunghyun Cho, and Joan Bruna. Backplay: "man muss immer umkehren". *arXiv preprint arXiv:1807.06919*, 2018. AAAI-19 Workshop on Reinforcement Learning in Games.

Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. *arXiv preprint arXiv:1812.03381*, 2018.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. MiniHack the planet: A sandbox for open-ended reinforcement learning research. In *NeurIPS*, 2021.

Finlay Sanders. Beating AutoAscend's NetHack challenge score with RL, 2026. URL https://x.com/finlay_sanders/status/2099518352536305850. Posted September 14, 2026.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, go, chess and shogi by planning with a learned model. *Nature*, 2020.

Mika Senghaas, Florian Brand, Will Brown, and Prime Intellect Team. verifiers v1: Decomposing tasksets and harnesses for agentic rl & evaluations. *Prime Intellect Blog*, July 2026. https://www.primeintellect.ai/blog/verifiers-v1.

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

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

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*, 2018.

Joseph Suarez. Pufferlib: Making reinforcement learning libraries and environments play nice. *arXiv preprint arXiv:2406.12905*, 2024.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

Maciej Sypetkowski and Michał Sypetkowski. Autoascend – 1st place nethack agent for the nethack challenge at neurips 2021. https://github.com/maciej-sypetkowski/autoascend, 2021. GitHub repository.

Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 1994.

Jens Tuyls, Dhruv Madeka, Kari Torkkola, Dean Foster, Karthik Narasimhan, and Sham Kakade. Scaling laws for imitation learning in single-agent games. *TMLR*, 2024.

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. *TMLR*, 2024. URL https://openreview.net/forum?id=nfx5IutEed.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *NeurIPS*, 2023.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE TPAMI*, 2025.

Mario Zechner and Earendil Works. Pi: A minimal, self-extensible terminal coding harness. https://github.com/earendil-works/pi, 2026. MIT License. Accessed: 2026-09-01.

Alex L. Zhang, Thomas L. Griffiths, Karthik R. Narasimhan, and Ofir Press. Videogamebench: Can vision-language models complete popular video games?, 2026a. URL https://arxiv.org/abs/2505.18134.

Jenny Zhang, Joel Lehman, Kenneth Stanley, and Jeff Clune. Omni: Open-endedness via models of human notions of interestingness. In *ICLR*, 2024.

Zhisong Zhang, Tianqing Fang, Kaixin Ma, Wenhao Yu, Hongming Zhang, Haitao Mi, and Dong Yu. Webrollback: Enhancing web agents with explicit rollback mechanisms. In *EACL*, 2026b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *AAAI*, 2024.

APPENDIX

A ENVIRONMENT COMPARISONS

Table 7 compares the capabilities of our environment versus NLE. Table 8 lists all the hyperparameters supported by the game-engine. *generation* knobs are consumed when a floor is built and therefore take effect at `reset()` time. *character* knobs are consumed at character creation, and *live* knobs are read each turn and may be changed mid-episode. Unless noted, values are multiplicative scalars in which 1.0 is vanilla behavior, 0.5 halves and 2.0 doubles; absolute-valued knobs are marked. Table 9 reports our default skills.

Capability	NLE	MiniHack	Ours
Full NetHack	yes	no (sub-tasks)	yes
Environments per process	1	1	thousands
Save / restore game state	no	no	yes (in $O(1)$ turns)
Branch a state into n futures	no	no	yes
Edit hero stats / depth at run time	no	partial (task setup)	yes
Tune generation parameters	no	task-authored	8 knobs
Tune character creation	no	partial	4 knobs
Tune combat / vision / economy	no	partial	30 live knobs
Serialize and transplant a level	no	no	yes
Observation formats	1	1	12
Action surfaces	1 (keys)	1 (keys)	3 (keys / NetPlay / NetCore)
Code-mode actions	no	no	yes (sandboxed Python)
Introspectable obs & action specs	no	no	yes
Seed-paired ablation harness	no	no	yes

Table 7: **Environment Comparisons.** Capability comparison of environment infrastructure (not task difficulty). MiniHack (Samvelyan et al., 2021) composes simplified sub-tasks from NetHack's level language; every configuration in our environment is the full game with a parameter changed.

Knob	Class	Default	Effect
`room_density`	generation	1.0	Number of rooms attempted per floor
`room_size`	generation	1.0	Dimensions of generated rooms
`corridor_connectivity`	generation	1.0	Density of the corridor graph between rooms
`trap_density`	generation	1.0	Number of traps generated
`locked_door`	generation	1.0	Probability that a door is generated locked
`mob_spawn`	generation	1.0	Initial monster population on a new floor
`item_spawn_chance`	generation	1.0	Item placement rate on a new floor
`artifact_spawn_chance`	generation	1.0	Artifact generation rate
`player_starting_hp`	character	— *abs*	Starting hit points
`player_starting_str`	character	— *abs*	Starting strength
`player_starting_dex`	character	— *abs*	Starting dexterity
`starting_pet_friendliness`	character	1.0	Starting pet disposition
`dmg_to_player_scale`	live	1.0	Damage dealt by monsters to the hero
`dmg_by_player_scale`	live	1.0	Damage dealt by the hero to monsters
`weapon_damage_scale`	live	1.0	Weapon damage
`armor_protection_scale`	live	1.0	Armor protection
`player_hp_scale`	live	1.0	HP gained per level-up
`hp_regen_scale`	live	1.0	Natural HP regeneration rate
`hunger_rate_scale`	live	1.0	Nutrition consumed per turn
`food_decay_rate`	live	1.0	Rate at which food spoils
`vision_radius`	live	0.0 *abs*	Sets vision radius
`reveal_map`	live	0.0	If 1.0, removes fog of war
`ongoing_spawn_scale`	live	1.0	Rate of monsters spawned over time
`monster_spawn_chance`	live	1.0	Per-turn monster spawn probability
`monster_difficulty_scale`	live	1.0	Level and stats of newly spawned monsters
`max_monster_difficulty`	live	— *abs*	Hard cap on generated monster difficulty
`monster_speed_scale`	live	1.0	Monster movement and attack frequency
`monster_aggression`	live	1.0	Monster hostility and pursuit
`player_speed_scale`	live	1.0	Hero movement rate
`xp_gain_scale`	live	1.0	Experience gained from kills
`luck_scale`	live	1.0	Hero luck
`spell_failure_rate`	live	1.0	Spell-casting failure chance
`teleport_chance`	live	1.0	Random teleportation probability
`shop_price_scale`	live	1.0	Shop prices
`quest_difficulty_scale`	live	1.0	Role quest difficulty

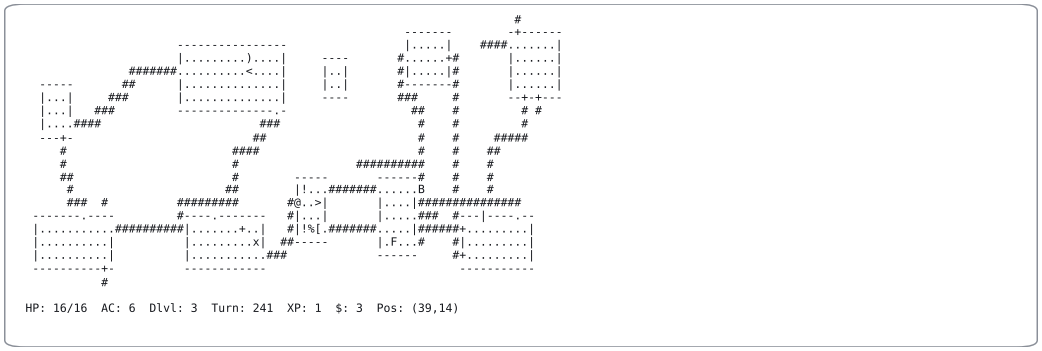
Table 8: **Difficulty tuning knobs.** Generation knobs apply at level creation, character knobs at character creation, and live knobs may be changed at any point during an episode. Values are multiplicative scalars unless marked *abs*.

Skill	Arguments	Skill	Arguments
`np_explore_level`	(none)	`np_move_to`	`x`, `y`
`np_melee_attack`	`x`, `y`	`np_kick`	`x`, `y`
`np_press_key`	`key`	`np_apply`	`item_letter` (optional)
`np_rest`	`count` (optional, default 5)	`np_pray`	(none)

Table 9: NetCore skill set and arguments for the NetHack agent.

B Observation Formats

(a) ASCII map



(b) JSON entities

```
{
  "player": [39, 14],
  "entities": [
    {"kind":"item","x":32,"y":3,"desc":"tool"},
    {"kind":"item","x":40,"y":13,"desc":"potion"},
    {"kind":"monster","x":57,"y":13,"desc":"bat"},
    {"kind":"monster","x":39,"y":14,"desc":"valkyrie"},
    {"kind":"item","x":31,"y":16,"desc":"door (closed)"},
    {"kind":"item","x":40,"y":16,"desc":"potion"},
    {"kind":"item","x":41,"y":16,"desc":"food/corpse"},
    {"kind":"item","x":42,"y":16,"desc":"armor"},
    {"kind":"monster","x":33,"y":17,"desc":"grid bug"},
    {"kind":"monster","x":53,"y":17,"desc":"brown mold"},
    {"kind":"door","x":71,"y":1,"desc":"door (closed)"},
    {"kind":"door","x":61,"y":3,"desc":"door (closed)"},
    {"kind":"stair","x":32,"y":4,"desc":"stairs UP"},
    {"kind":"door","x":72,"y":6,"desc":"door (closed)"},
    {"kind":"door","x":74,"y":6,"desc":"door (closed)"},
    {"kind":"door","x":5,"y":9,"desc":"door (closed)"},
    {"kind":"stair","x":42,"y":14,"desc":"stairs DOWN"},
    {"kind":"door","x":31,"y":16,"desc":"door (closed)"},
    {"kind":"door","x":63,"y":16,"desc":"door (closed)"},
    {"kind":"door","x":63,"y":18,"desc":"door (closed)"},
    {"kind":"door","x":11,"y":19,"desc":"door (closed)"}
  ]
}
```

(c) Lists, map hidden

```
=== MAP (hidden; call reveal(x1,y1,x2,y2) to view a region) ===

=== STATUS ===
HP: 16/16  AC: 6  Dlvl: 3  Turn: 241  XP: 1  $: 3  Pos: (39,14)
Character: valkyrie (human, neutral)

=== VISIBLE FEATURES ===
stairs DOWN at (42,14); stairs UP at (32,4); door (open/gap) at
(70,2), (55,3), (22,4), (36,7), (6,8), (43,13), (51,13), (56,13),
(8,15), (27,15), (56,15), (66,15), (71,15), (12,16), (43,16), (51,16),
(56,17), (34,18); door (closed) at (71,1), (61,3), (72,6), (74,6),
(5,9), (31,16), (63,16), (63,18), (11,19); food/corpse at (41,16);
potion at (40,13), (40,16); armor at (42,16); weapon at (32,3)

=== VISIBLE MONSTERS ===
grid bug at (33,17), 6 steps SW; brown mold at (53,17), 14 steps E;
bat at (57,13), 18 steps E

=== MESSAGES ===
  There is a doorway here.
```

Figure 4: **Observation Encodings.** (a) The ASCII map from NetHack. (b) JSON entities in the grid. (c) Standard output when the map is not refreshed.

C Expanded General Results

		Gemini-3	Prime Agent + NetCore					GLM-5.2 with modifications			
	Human	Pro	GLM-5.2	Sol	Luna	Flash	Qwen	harness*	cont./wiki	explore @ 30	explore @ 100
n	46,796	5	15	15	15	15	15	15	15	7	3
Dlvl	6.48	5.60 ± 1.94	8.00 ± 0.88	6.47 ± 0.54	5.53 ± 0.55	8.33 ± 0.72	4.73 ± 0.88	7.73 ± 0.74	5.27 ± 0.64	18.43 ± 1.53	23.33 ± 1.45
XL	5.04	3.40 ± 0.81	2.33 ± 0.36	2.67 ± 0.30	2.27 ± 0.30	2.20 ± 0.26	3.40 ± 0.41	2.93 ± 0.27	3.13 ± 0.31	8.57 ± 0.30	10.33 ± 0.33
BAL_{max} mean	8.05	6.77 ± 3.59	9.96 ± 2.19	5.28 ± 1.00	3.79 ± 0.61	9.80 ± 1.80	4.56 ± 1.10	8.50 ± 1.66	3.73 ± 0.86	34.66 ± 3.19	44.18 ± 3.38
BAL_{max} median	2.91	2.12	6.96	3.54	3.54	9.77	2.42	9.77	2.65	37.90	42.58
BAL_{min} mean	5.41	1.84 ± 0.60	1.32 ± 0.30	1.65 ± 0.23	1.50 ± 0.25	1.47 ± 0.24	1.74 ± 0.31	1.88 ± 0.20	1.96 ± 0.22	11.38 ± 1.33	20.43 ± 2.52
BAL_{min} median	2.12	1.54	1.85	2.08	1.85	1.85	2.12	2.08	2.08	11.70	17.91

Table 10: **Expanded Main Results.** We compare evaluations of the Prime Agent + NetCore with prior state-of-the-art evaluation results from BALROG, as well as human gameplay from NetHack Alt.org (NAO, Hambro et al. (2022b)) games that ran to a terminal state. We evaluate GLM-5.2, GPT-5.6 Sol, GPT-5.6 Luna, Gemini-3.7 Flash, and Qwen-3.8 Max with the Prime Agent + NetCore. *We report the best harness intervention that leaves the game itself unchanged, the ASCII map ablation of Section 5.1. 'Forced revive' performs better (BAL_{min} at 2.64 ± 0.20), but fundamentally changes the game rather than the harness.

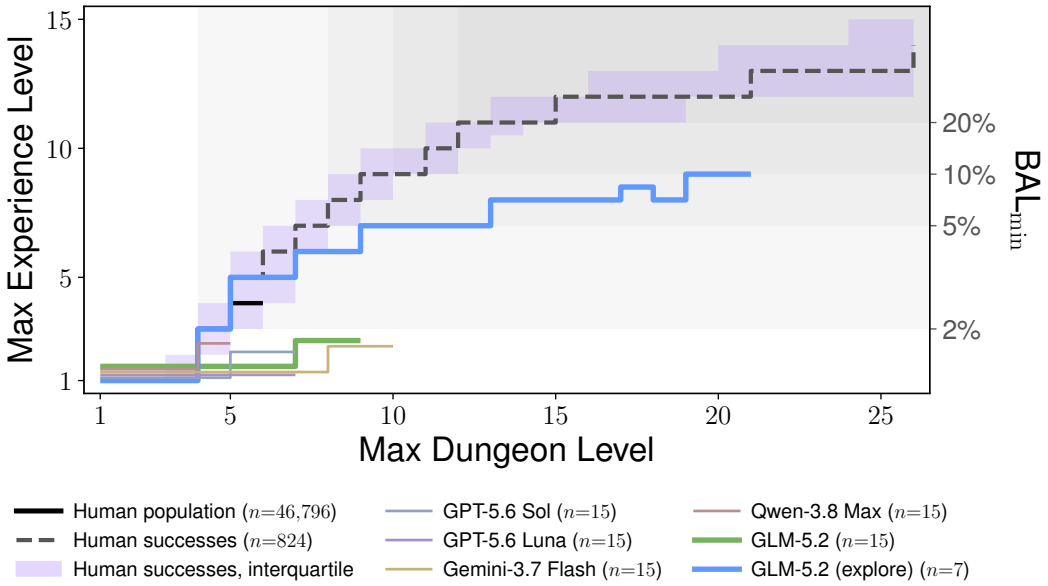


Figure 5: **Leveling vs Descending Strategy Profile.** For each model, we plot the median BAL_{min} trajectory's progress through the trace. Models disproportionately prioritize Dungeon Level over increasing experience. Prime-agent-explore strays from the human interquartile range (IQR) later.

D Continual Harness Progression

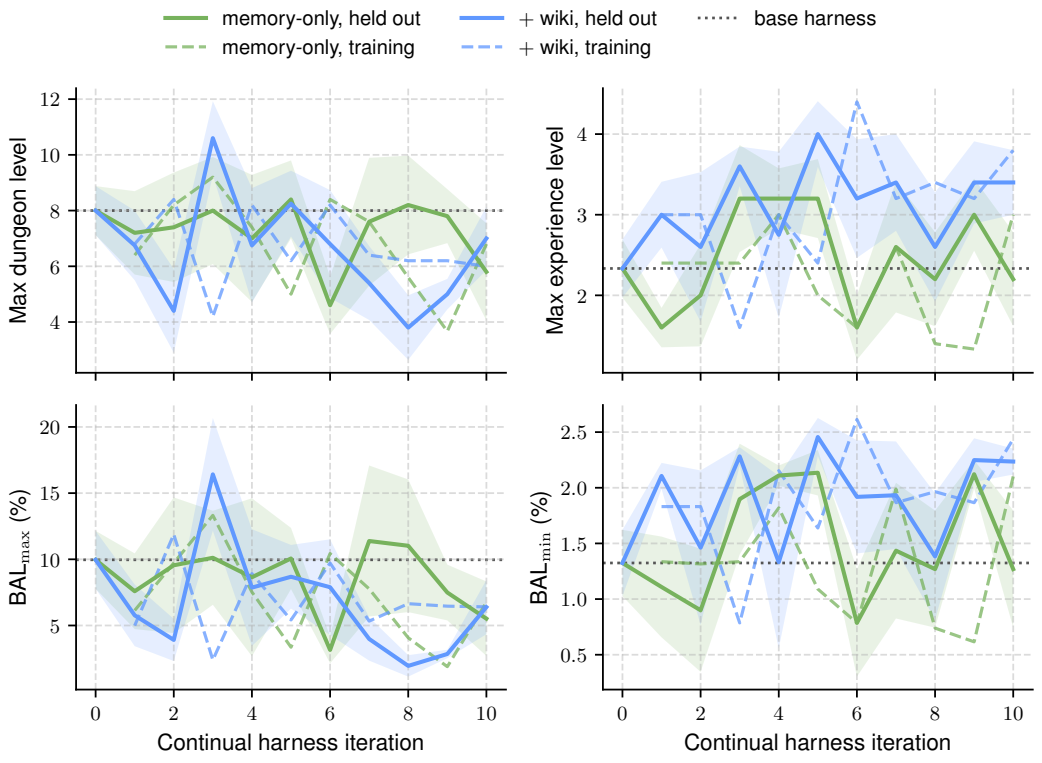


Figure 6: **Continual Harness Metrics.** Max DLvl, XL, BAL_{max}, and BAL_{min} for the *memory* and *memory+wiki* Continual Harness runs. Held out denotes seeds 0-4 and training denotes seeds 6-10.

E PRIME-AGENT-EXPLORE TRACE

Figure 7 shows an attempt tree for Prime-agent-explore. Below, we provide one round of the loop including 1) the orchestrator's analysis and directive, 2) the prompt served to the rollout agent, and 3) the rollout agent's own reasoning under that directive.

Orchestrator Analysis. The orchestrator reads attempt 9's outcome and the archive, rejects the deepest checkpoint, and resumes from an intermediate checkpoint during attempt 9's rollout.

```
=== ORCHESTRATOR: round 10 selection ===

**Analysis:** Attempt 9 was hugely productive -- reached dlvl 10, up
from dlvl 7. But the hero is now under-levelled by 2 (XL 7 at dlvl 10,
winners are XL 9). The death spiral on dlvl 10 (c120->c122 !Weak->c123
!Weak 14HP) shows the hero went too deep too fast without enough
experience. The right move is NOT to resume from the deepest point.
c120 is Hungry at 86% HP on a floor that killed the hero. Instead,
I'll step back to c112 -- dlvl 8, XL 7, 70/70 full HP, Normal food.
This is exactly the "slow is not failing" scenario -- the hero needs
to catch up on experience.

{"checkpoint": "c112",
 "directive": "You are under-levelled for your depth -- stay on this
   floor and kill monsters one at a time in corridors until you reach
   XL 8 or 9 before descending further. [...] Do not go down the
   stairs until you have gained at least one experience level.",
 "rationale": "the hero rushed to dlvl 10 at XL 7 and died -- farming
   XP here to reach XL 8-9 before descending is the correction."}
```

Prompt for the Rollout Agent. The rollout agent is provided the directive as well as checkpoint contextualization, in order to help diversify.

```
=== ROLLOUT AGENT: first observation, attempt 10 ===

[ORCHESTRATOR DIRECTIVE for this attempt: You are under-levelled for
 your depth -- stay on this floor and kill monsters one at a time in
```

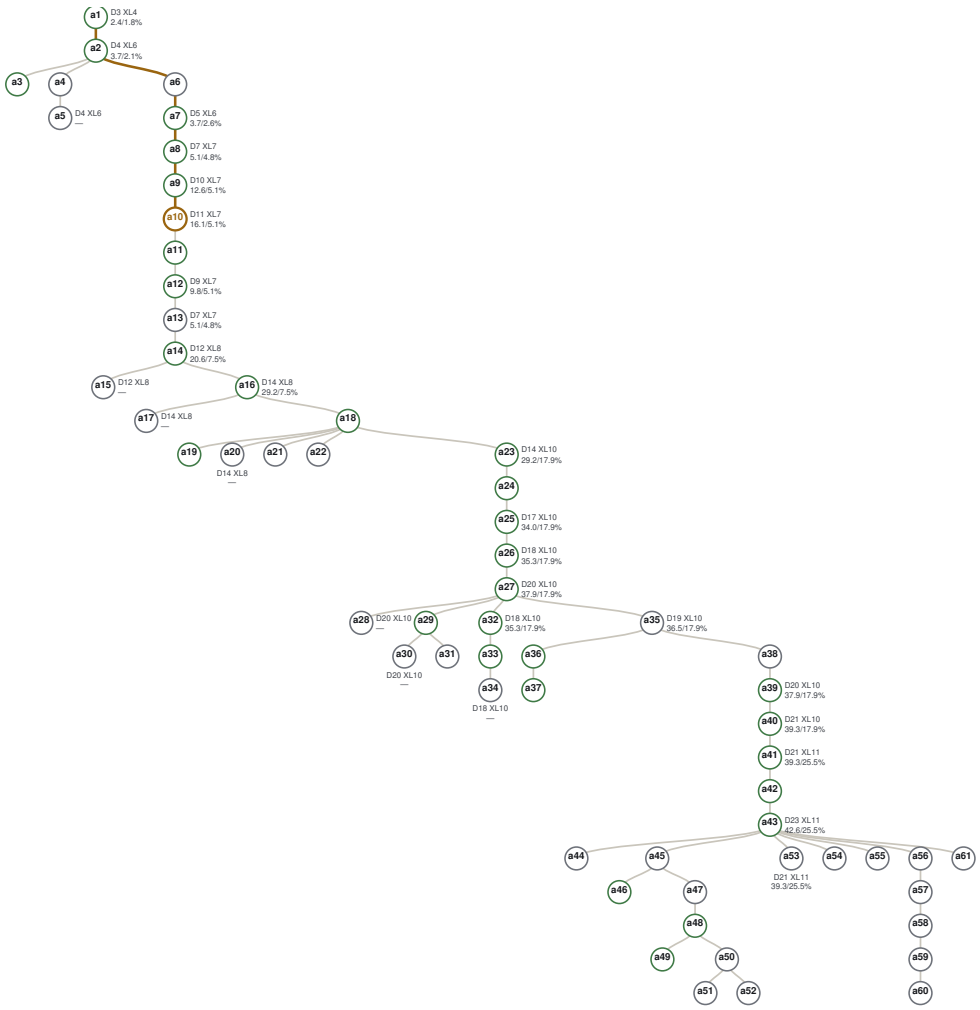


Figure 7: **Prime-Agent-Explore Attempt Tree.** Each node is one attempt and is marked with (Dlvl, XL on top and $\mathrm{BAL}_{\max}$/ $\mathrm{BAL}_{\min}$ below). Edges point from resumed checkpoints to their ending state. The golden node marks the node selected in the text examples.

```
 corridors until you reach XL 8 or 9 before descending further. [...]]

[RESUMED FROM CHECKPOINT 112 ('auto call_421') -- you are NOT starting
 fresh. Dlvl 8, XL 7, HP 70/70, game turn 9104.
 Why it was saved: automatic checkpoint: 20 LM calls elapsed]

CHECKPOINT ARCHIVE: 123 saved state(s) in 120 distinct positions.
   id    from  Dlvl  XL  dXL    HP    food    turn  tried
   c1    root     1   1    .  16/16  Normal     1  1: died@D3
   c8      c7     1   1    .  13/16  Normal   1143  -
   [...]
```

Rollout Agent Reasoning. The agent considers how to implement the directive given to it.

```
=== ROLLOUT AGENT: reasoning, turn 156, Dlvl 8 ===

I found the down stairs! '>' at (33,17)! But the directive says I
should stay on this floor and gain XP before descending. I'm at XL 7
and need to reach XL 8 or 9.

However, I've been spending a LOT of turns on this floor without
gaining any XP. Let me think about my options:
1. Stay and fight monsters for XP - I've had trouble catching monsters
2. Descend now - the directive says to gain at least one level
```

```
Actually, let me try to fight some monsters. I've seen various
monsters appear (housecat, lynx, neanderthal, Mordor orc) but they
keep moving before I can attack them. Let me be more aggressive
about hunting monsters. [...]
```

F CONTINUAL HARNESS PROCEDURE

In each iteration t, five subagents independently play a game of NetHack, with each rollout having its own seed, work directory, and private copy of the memory store from the conclusion of iteration $t-1$. Once the runs complete, the orchestrator session, a separate Prime Agent process, reads the finished traces and issues create/update/delete edits against the merged store. The orchestrator is additionally given a seed-matched scoreboard of every previous iteration together with its own past rationales, so it can attribute a change in score to an entry it wrote. The runtime's prompt renderer caps entries at six per kind (memory, prompt note, skill) to avoid prompt bloat.

In the Wiki variant, we instruct the orchestrator agent to consult two NetHack strategy pages[4] prior to making changes. After every iteration, we freeze the store and evaluate it, mounted read-only, on a fixed set of five held-out evaluation seeds, disjoint from the training seeds. For the *skills+wiki* run, we reuse a single training seed and use one subagent per iteration.

We report evaluations on five held-out seeds after each round of the train–reflect–evaluate loop for ten iterations. Held-out scores at a single iteration are noisy due to large behavioral shifts iteration-to-iteration, so we report evaluations pooled from each optimization run's last three iterations.

G PRIME AGENT VERSUS CLAUDE CODE

Table 11 compares Prime Agent with Claude Code (Anthropic, 2025). Results suggest that Claude Code performs better on metrics, but uses 2.7× the uncached input tokens and nearly 2× the output tokens.

Metric ($n = 15$ each)	Prime Agent + NetCore	Claude Code + NetCore
BAL_{min}	1.32 ± 0.30	1.57 ± 0.31
BAL_{max}	9.96 ± 2.19	11.05 ± 2.50
Max Dlvl	8.0 ± 0.9	8.40 ± 0.99
Max XL	2.3 ± 0.4	2.80 ± 0.39
Game calls per life	113.6	141.4
Uncached input tokens	1.28M	3.40M
Cached input tokens	57.3M	96.0M
Output tokens	0.20M	0.39M

Table 11: **Harness Comparison.** Each cell reports mean ± standard error over 15 rollouts (5 seeds ×3 repetitions).

[4] https://nethackwiki.com/wiki/Standard_strategy, https://nethackwiki.com/wiki/Why_do_I_keep_dying