PTCG-Bench: Can LLM Agents Master Pokémon Trading Card Game?

Dongdong Hua[1], Yifei Sun[1], Renhong Huang[1], Feng Gao[2], Chunping Wang[2], Yang Yang[1]*
[1] Zhejiang University, [2] FinVolution Group
{ddhua,yifeisun,renh2,yangya}@zju.edu.cn
{gaofeng02,wangchunping02}@xinye.com

Abstract

Given a strategically complex board game, human players can quickly learn to devise strategies after playing a few rounds. Autonomous agents require similar capabilities in realistic interactive environments, yet existing agent benchmarks often fail to fully capture such strategic and evolving decision-making scenarios. We present **PTCG-Bench**, a benchmark built on the Pokémon Trading Card Game (PTCG) that evaluates LLM agents at two complementary levels: (1) their decision-making performance within a single complex environment, and (2) their ability to self-evolve through accumulated experience. We further include a modular harness ablation to better interpret agent performance without conflating it with model capability. Our experiments show that, although LLM agents can achieve non-trivial gameplay performance, sustained and stable self-evolution remains challenging, and performance is sensitive to harness design. We hope that PTCG-Bench will facilitate future research on harness-aware and self-evolving agents in realistic interactive environments.[1]

1 Introduction

Games have long served as important testbeds for autonomous agents, driving progress from Atari (Mnih et al., 2015), Go (Silver et al., 2017), and StarCraft II (Vinyals et al., 2019) to recent LLM agents in Minecraft (Wang et al., 2023) and Diplomacy (Team et al., 2022). More recently, Pokémon has attracted growing attention as a challenging domain for both competitive battling (Karten et al., 2025) and open-ended RPG gameplay (Anthropic, 2025). Meanwhile, game-based benchmarks have expanded from classic environments such as ALE (Bellemare et al., 2013), TextWorld (Côté et al., 2018), ALFWorld (Shridhar et al., 2020), and MineDojo (Fan et al., 2022) to LLM-agent benchmarks such as lmgame-Bench (Hu et al., 2025), Orak (Park et al., 2025), and PokeAgent Challenge (Karten et al., 2026). However, as summarized in Table 1, existing benchmarks often emphasize isolated aspects of agent capability, provide limited support for evaluating sustained self-evolution within the same strategic environment, or leave harness-level effects insufficiently separated from model capability. This leaves open the need for a unified benchmark that can jointly assess strategic gameplay, stable self-evolution, and harness-aware agent evaluation.

To address this gap, we introduce PTCG-Bench, a benchmark built upon the Pokémon Trading Card Game (PTCG)[2], as shown in Figure 1. PTCG is a strategic two-player zero-sum game in which agents must reason over partially observable game states, make decisions under stochastic card draws, integrate textual card effects with numerical attributes, and plan over adversarial interactions. This setting allows us to evaluate whether LLM agents can combine imperfect-information reasoning, long-horizon planning, hybrid textual-numerical inference, and strategic decision-making within a single coherent game, rather than solving them as isolated subtasks.

Beyond initial performance, PTCG-Bench evaluates a capability that is increasingly central to autonomous agents: whether they can improve through experience (Ou et al., 2025). For agents deployed in realistic environments, success cannot rely solely on static instructions or one-shot task solving; instead, agents must learn from trial and error, consolidate reusable experience, and adapt their future decisions as environments change. Recent self-improvement methods (Agrawal et al.,

* Corresponding author.

[1] Code is available at https://github.com/zjunet/PTCG-Bench.

[2] For an official introduction to the Pokémon Trading Card Game, see https://tcg.pokemon.com/en-us/learn/.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 36084–36099
October 24-29, 2026 ©2026 Association for Computational Linguistics

Figure 1: Overview of the environment of PTCG-Bench. **Left:** An example Pokémon card annotated with key attributes including HP and Type, along with Ability and Attack descriptions that govern in-game decisions. **Right:** The two-player game board, where each player maintains an Active spot, up to five Bench slots, a Prize card zone, a Deck, a Discard pile, and a hand of hidden cards. Agents must reason over their own observable board state while inferring hidden information from the opponent's side.

Table 1: Comparison between PTCG-Bench and representative game-based benchmarks. Abbrev.: **IIR** = imperfect-information reasoning, **LHP** = long-horizon planning, **TNI** = text-numerical inference, **SDM** = strategic decision-making, **LSE** = long-term self-evolution, **MHS** = modular harness support, and **SGI** = single-game integration. "▲" denotes partial coverage, where the capability is present in the environment or appears in some tasks but is not explicitly evaluated as a central requirement.

Benchmark	**IIR**	**LHP**	**TNI**	**SDM**	**LSE**	**MHS**	**SGI**
ALE (Bellemare et al., 2013)	✗	▲	✗	▲	✗	✗	✗
MiniGrid (Chevalier-Boisvert et al., 2023)	▲	▲	✗	✗	✗	✗	✗
Procgen (Mohanty et al., 2021)	✗	▲	✗	▲	✗	✗	✗
TextWorld (Côté et al., 2018)	▲	▲	✗	▲	✗	✗	✗
ALFWorld (Shridhar et al., 2020)	▲	▲	▲	▲	✗	✗	✗
MineDojo (Fan et al., 2022)	▲	✓	▲	▲	▲	✗	✓
lmgame-Bench (Hu et al., 2025)	▲	▲	▲	✓	✗	✓	✗
Orak (Park et al., 2025)	▲	▲	▲	✓	▲	✓	✗
PokeAgent Challenge (Karten et al., 2026)	✓	✓	▲	✓	▲	▲	▲
PTCG-Bench (Ours)	✓	✓	✓	✓	✓	✓	✓

2025; Xu et al., 2026; Alzubi et al., 2026) and benchmarks (Jimenez et al., 2024; Zhou et al., 2024; Liu et al., 2024) have provided useful insights, but many still emphasize relatively static, short-horizon, or perfect-information settings. To better approximate the conditions faced by agents in realistic environments, PTCG-Bench evaluates self-evolving agents under a **longitudinal evaluation protocol**, in which self-evolving agents play a sequence of games, accumulate experience, and are evaluated over multiple rounds to determine whether such experience leads to improved performance in a dynamic environment.

PTCG-Bench further enables controlled analysis of **agent harness design**. Modern LLM agents depend not only on the backbone model, but also on the surrounding harness design (Yang et al., 2024). We therefore design a modular harness around three key factors: observation structure, legal-action masking, and context management. By ablating these components, PTCG-Bench allows us to quantify how harness choices affect gameplay performance and separate their effects from backbone capability.

We conduct extensive experiments and the results show that LLM agents can achieve competitive gameplay performance, but their performance varies widely across models and harness designs. PTCG-Bench produces a wide ranking of agent systems, with a 665-point Glicko-2 rating gap between the strongest and weakest LLM-based agents.[3]

[3]Under the Glicko-2 formula, a 665-point rating difference corresponds to an expected win probability of approximately 97.8% for the higher-rated player.

Moreover, PTCG-Bench exposes the limitations of existing self-evolution methods, with most agents failing to improve consistently over time.

Our main contributions are as follows:

- We introduce **PTCG-Bench**, a PTCG-based benchmark for evaluating LLM agents in a complex environment involving imperfect information, long-horizon planning, and hybrid textual-numerical reasoning.
- We propose a **longitudinal evaluation protocol** to evaluate whether self-evolving agents can accumulate cross-game experience and convert it into improved subsequent decisions.
- We develop a **modular agent harness** with independently ablatable components, enabling controlled analysis of multiple harness designs beyond backbone capability.

2 Preliminary

2.1 Pokémon Trading Card Game

PTCG is a two-player, turn-based, zero-sum card game in which each player uses a 60-card deck composed mainly of Pokémon, Energy, and Trainer cards. Players set up an Active Pokémon, optional Benched Pokémon, and six face-down Prize cards, then alternate turns to develop their board, manage resources, and attack the opponent. A player wins by taking all Prize cards, eliminating all opposing Pokémon in play, or causing the opponent to draw from an empty deck.

PTCG is challenging for LLM agents because it combines hidden information, stochastic card draws, dynamic legal actions, and long-horizon resource planning. Agents must interpret natural-language card effects while reasoning over numerical quantities such as HP, damage, Energy costs, and Prize counts, and must make sequential decisions whose consequences can compound over many turns. These properties make PTCG a suitable environment for evaluating strategic decision-making and experience-based improvement. Complete rules are included in Appendix A.

2.2 LLM Agent Formalization

By absorbing the opponent policy into the environment dynamics, a PTCG match can be modeled as a partially observable Markov decision process (POMDP), where the complete game state includes hidden information such as the opponent's hand, face-down Prize cards, and unrevealed deck order. At step t, the agent receives an observation o_t, maintains an interaction history $h_t = (o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t)$, selects an action a_t, and receives a reward r_t. Because the game state is not fully observable, the agent must act based on o_t and h_t rather than the complete state.

An LLM agent induces its policy through both a backbone model and a surrounding harness. The harness transforms the observation and history into a prompt, constrains or formats the available actions, and parses the model output into an executable action. We denote the resulting policy by

$$a_t \sim \pi_{\theta,\eta}(\cdot \mid o_t, h_t), \tag{1}$$

where θ represents the LLM backbone and η represents the harness design. The expected performance of the agent can then be written as

$$J(\theta, \eta) = \mathbb{E}_{\pi_{\theta,\eta}} \left[\sum_t r_t \right]. \tag{2}$$

3 PTCG-Bench

PTCG-Bench evaluates LLM-based agents in the Pokémon Trading Card Game (PTCG) along three dimensions: **(i)** decision-making under imperfect information and long-horizon strategic interaction, **(ii)** self-evolution through accumulated cross-game experience, and **(iii)** the contribution of harness design beyond backbone capability. This section describes the game environment, the self-evolution protocol, the modular harness, the evaluation tournament, and the evaluation metrics.

3.1 Benchmark Environment

Engine implementation. PTCG-Bench uses a custom PTCG engine[4] for reproducible rule execution and legal-action validation. The engine implements the core mechanics needed for full-game evaluation, thereby providing a consistent environment for comparing agent systems. Detailed implementation details are provided in Appendix A.

Deck pool. We use a fixed pool of 5 competitive decks covering distinct strategic archetypes, including aggressive, controlling, combo-oriented, and attrition-based plans. The deck pool is used both for controlled mirror-match evaluation and for cross-deck generalization analysis. Detailed deck statistics and strategic profiles are in Appendix B.

[4] https://github.com/gemelom/ptcg-engine

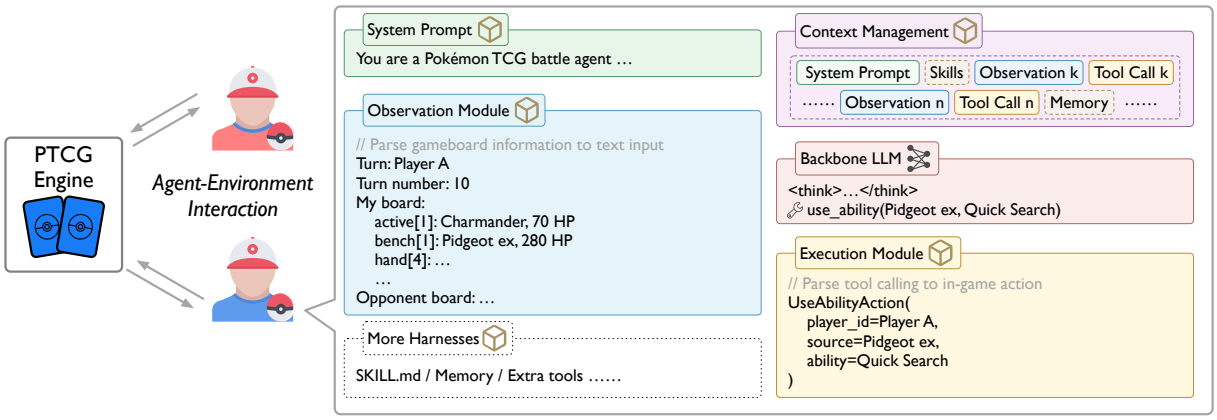

Figure 2: Agent-environment interaction loop in PTCG-Bench. At each decision point, the game engine exposes the current state and legal action set to the agent harness, which converts them into an LLM-readable prompt. The backbone model's response is parsed, validated, and executed by the engine to advance the game state.

State and action spaces. PTCG-Bench separates private, public, and global state to preserve imperfect information. Agents observe their own private zones and public board information, while the opponent's hand, deck order, and Prize-card identities remain hidden. The action space is defined as a set of parameterized game actions exposed through tool schemas, including playing or evolving Pokémon, attaching Energy, playing Trainer cards, attacking, retreating, and passing. Each action specifies the required arguments needed for execution, thereby providing a structured interface between agent decisions and game mechanics.

3.2 Agent-Environment Interface

As shown in Figure 2, the engine exposes the game state at each decision point, from which the agent receives a partial observation. The agent harness combines this observation with the interaction history and renders them into an LLM-readable prompt. The backbone LLM then returns a structured action request, typically as a tool call with an action type and the corresponding parameters. The model response is parsed by the harness and checked by the engine against the current legal action set. Valid actions are executed to advance the game state, while invalid requests trigger a retry. This interface fixes the underlying game execution while allowing agent systems to vary in how they represent information, use history, and select actions. Such flexibility, in turn, enables evaluation with multiple harness modules.

3.3 Longitudinal Evaluation Protocol

PTCG-Bench evaluates whether self-evolving agents improve through accumulated game experience. Games are organized into multiple rounds of complete PTCG matches. After each round, an evolving agent may update persistent state, such as reflections, strategic summaries, memory entries, revised prompts, or skill documents. This protocol separates within-game decision-making from cross-game experience updates. We track round-wise performance against fixed-policy anchors to analyze whether accumulated experience improves, preserves, or degrades subsequent play.

3.4 Modular Harness Architecture

In PTCG-Bench, the harness is not treated as a neutral wrapper between the game engine and backbone LLM. Prior studies show that the surrounding agent scaffold, including state perception, context management, and action execution, can substantially change the measured capability of the same backbone model.

PTCG-Bench therefore decomposes the harness into modular components, allowing key agent-environment coupling factors to be ablated independently. This modular separation helps prevent the measured performance from conflating backbone capability with the effects of state perception, context management, and action execution. The specific harness modules and ablation settings are introduced in Section 4.4.

3.5 Evaluation Tournament

As shown in Figure 3, PTCG-Bench uses two tournaments for different evaluation goals. Fixed agents are compared by round robin (Harary and Moser, 1966), where every configuration plays every other configuration to estimate overall playing

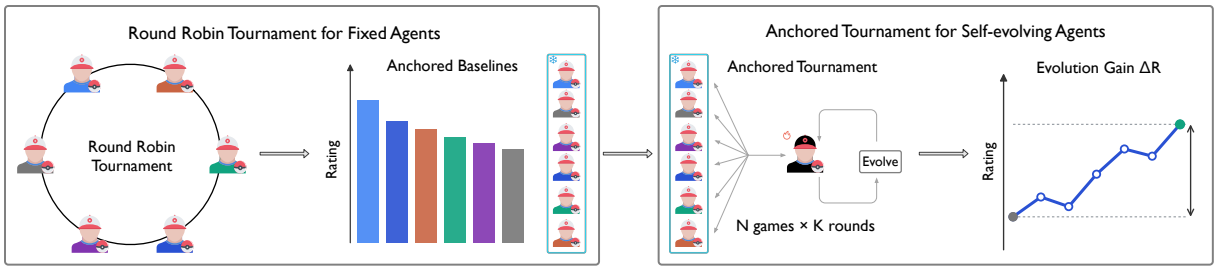

Figure 3: Evaluation tournament design in PTCG-Bench. **Left:** Fixed-policy agents are first evaluated by a round-robin tournament to obtain ratings, from which a set of fixed anchor baselines is selected. **Right:** Self-evolving agents are then evaluated through an anchored tournament, where each evolving policy plays against the frozen anchor baselines across rounds.

strength on a shared rating scale. This setting supports direct comparison among static backbones and harness configurations.

Self-evolving agents use an anchored tournament because simultaneous policy updates would make a round-robin rating scale drift across rounds. Each evolving agent plays fixed-policy anchors with calibrated ratings spanning the rating range, then may update its persistent state. Anchor matchups therefore preserve a stable reference population across agent snapshots. Since anchors do not change, round-wise rating shifts primarily reflect changes in the evolving agent.

3.6 Evaluation Metrics

PTCG-Bench reports 4 complementary metrics.

Glicko-2 rating. We use Glicko-2 (Glickman, 2012) as the primary measure of playing strength, reporting both the rating mean μ and rating deviation ϕ under stochastic match outcomes. Ratings are estimated from round-robin records for fixed agents and from anchor matchups for evolving agents, with ϕ indicating estimation uncertainty.

Head-to-head win rate. Head-to-head win rates provide an interpretable complement to Glicko-2 by exposing matchup-level advantages that aggregate ratings can obscure.

Invalid action rate. Invalid action rate measures how often an agent produces illegal or unparsable actions before retry or fallback and is especially relevant to harness ablations.

Tool calls. The number of tool calls measures interaction cost, counting action attempts and operations invoked during decision-making. We use it to contextualize configuration efficiency.

4 Experiments

4.1 Experimental Setup

In this section, we conduct comprehensive experiments to answer the following research questions: (**RQ1**) Can PTCG-Bench Resolve Strategic Capability Differences among LLM Agents? (**RQ2**) Can PTCG-Bench Measure Self-Evolution over Sequential Play? (**RQ3**) How Much Does Harness Design Affect Agent Performance?

LLM backbones. We evaluate 10 backbones from 5 model families, pairing a frontier and cost-efficient variant from each family: Gemini 3.1 Pro and Gemini 3 Flash, Claude Sonnet 4.6 and Claude Haiku 4.5, DeepSeek V4 Pro and DeepSeek V4 Flash, GPT-5.4 and GPT-5.4 Nano, and Qwen3.6 Plus and Qwen3.5 Flash. Unless otherwise specified, all agents use the same ReAct-style harness.

Evaluation configuration. Fixed-agent experiments use the round-robin tournament in Section 3.5 with ten LLM agents and two non-LLM references, Random and Heuristic, which sample legal actions with uniform and predefined weights, respectively. Each pair plays $M = 20$ games, yielding 1320 games per tournament. Self-evolution uses the anchored tournament in Section 3.5. We compute ratings with Glicko-2 and use mirror matches unless stated otherwise.

Evaluation scale. We set $M = 20$ to balance rating resolution and evaluation cost. This scale provides sufficient resolution to distinguish agents with meaningful differences in strategic capability and produces stable anchor ratings for subsequent anchored evaluations. At the same time, it keeps the evaluation computationally tractable, as each PTCG game involves long-horizon interaction and repeated model inference over many decision steps.

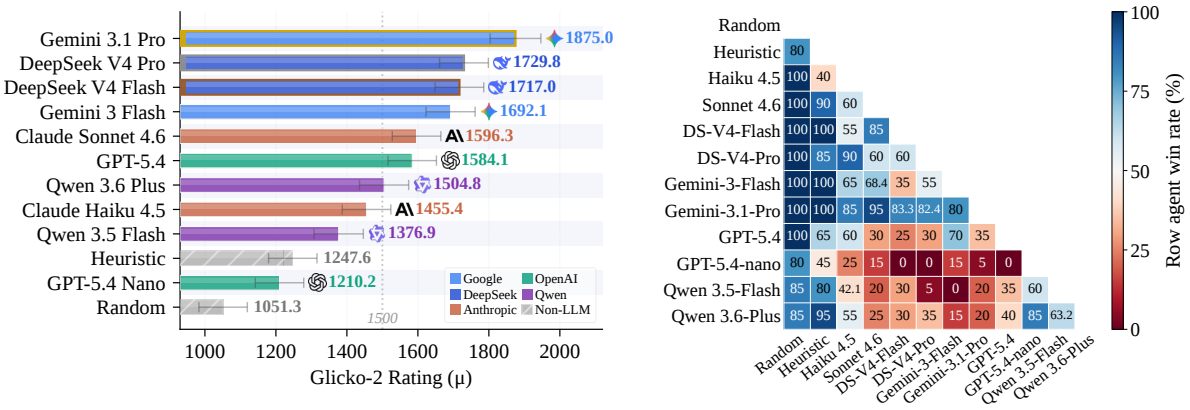

Figure 4: PTCG-Bench tournament results under a unified ReAct harness. **Left:** Glicko-2 ratings ($\mu \pm \phi$) for LLM backbones and fixed-capability anchor agents, with all agents initialized at $\mu = 1500$ and $\phi = 350$; error bars indicate rating deviation. **Right:** pairwise win-rate heatmap, where each cell gives the row agent's win rate against the column agent. Higher ratings and colder heatmap values indicate stronger play.

4.2 Backbone Strategic Capabilities

To answer **RQ1**, we test whether PTCG-Bench resolves backbone-level strategic differences under a fixed ReAct harness and mirror-match setting. Figure 4 reports Glicko-2 ratings for all 10 backbones together with the fixed-capability anchors.

PTCG-Bench produces a broad and well-resolved rating distribution. Among LLM-based agents, ratings span from GPT-5.4 Nano (1210) to Gemini 3.1 Pro (1875), corresponding to a gap of 665 Glicko-2 points. This spread substantially exceeds the rating deviations of most individual agents, suggesting that PTCG-Bench can separate LLM agents by playing strength rather than collapsing them into a narrow performance band. Head-to-head win rates in the pairwise heatmap further show that frontier variants consistently outperform their cost-efficient counterparts within the same model family, suggesting that PTCG-Bench captures meaningful capability differences rather than tournament noise.

Gameplay strength is not determined by inference cost. Figure 5 compares Glicko-2 rating with inference cost per game. The non-monotonic distribution shows that higher cost does not necessarily yield stronger play, suggesting that PTCG-Bench also reveals practical cost–performance trade-offs among LLM agents.

PTCG-Bench captures capabilities not fully reflected by general-purpose benchmarks. We further compare the PTCG-Bench ranking with model orderings reported by other LLM evaluations such as LiveBench, SWE-Bench Pro, and GPQA (White et al., 2025; Deng et al., 2025; Rein et al., 2023). As detailed in Appendix E, the resulting rankings are only partially aligned. In particular, models that are stronger on general language understanding or preference-based evaluation do not always obtain proportionally higher PTCG-Bench ratings. This divergence suggests that PTCG-Bench evaluates capabilities beyond static language understanding, particularly imperfect-information reasoning, long-horizon planning, and strategic decision-making under uncertainty.

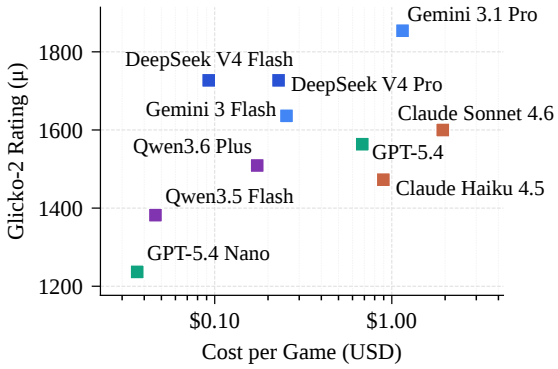

Figure 5: Cost–rating trade-off for all ten LLM backbones. The y-axis reports the Glicko-2 rating mean μ, and the x-axis reports inference cost per game in USD. Models closer to the upper-left region achieve stronger play at lower cost.

4.3 Self-Evolution

To answer **RQ2**, we evaluate whether PTCG-Bench can track *self-evolution* across sequential play using fixed anchors and a stable rating scale.

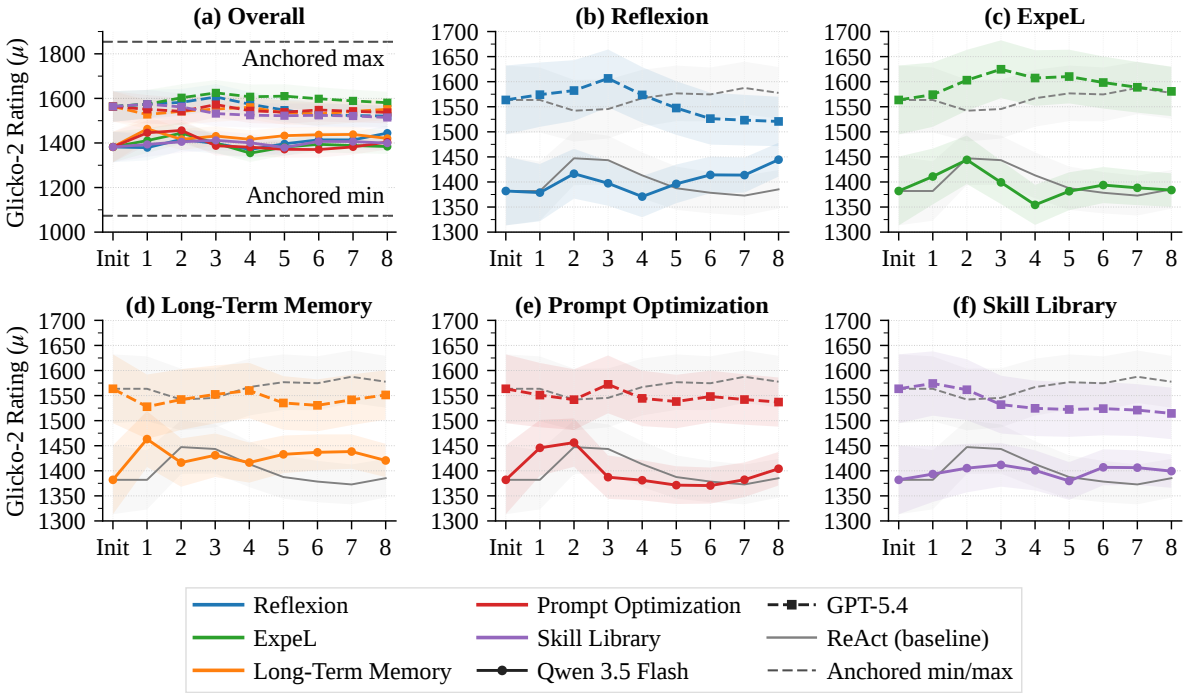

Figure 6: Glicko-2 rating trajectories across self-evolution rounds for five evolving agent configurations using Qwen3.5-Flash and GPT-5.4 as backbones. Panel (a) summarizes all evolving configurations relative to the anchored rating range, with gray dashed lines denoting anchored maximum and minimum ratings and the dotted line denoting the non-evolving same-backbone reference. Panels (b)–(f) provide method-specific comparisons against the non-evolving ReAct baseline, with shaded regions indicating rating uncertainty bands.

Evolving agent configurations. We evaluate five representative self-evolving baselines under the same PTCG-Bench harness: Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), long-term memory (Park et al., 2023; Packer et al., 2023; Tan et al., 2025; Xu et al., 2026), prompt evolution (Madaan et al., 2023; Wang et al., 2024; Yuksekgonul et al., 2025), and skill-library evolution (Anthropic, 2026; Alzubi et al., 2026). Since PTCG involves long horizons, delayed high-variance outcomes, hidden information, stochastic card order, and opponent-dependent strategy shifts, directly transplanting rollout- or search-based methods would be costly and unstable. We therefore adapt these baselines at the mechanism level. All configurations use the same backbone and harness to isolate the effect of self-evolution; details are provided in Appendix D.

Anchored self-evolution protocol. We instantiate each of the five self-evolution mechanisms with two backbones, Qwen3.5-Flash and GPT-5.4, and evaluate each resulting configuration for $R = 8$ rounds. In each round, every configuration plays 2 games against each of 12 fixed anchors: the 10 static LLM agents from RQ1 plus the Random and Heuristic agents. For each backbone, we additionally report the corresponding non-evolving ReAct agent as a static reference. This yields 24 games per configuration, $N = 120$ evolving-agent games per backbone per round, and 240 games per round across both backbones. After each round, each mechanism updates its persistent state using the accumulated trajectories before the next snapshot is evaluated against the same anchors, keeping rating changes on a stable reference scale.

4.4 Modular Harness Ablation

Figure 6 reports the resulting rating trajectories and compares them with anchored baseline ratings and the non-evolving same-backbone ReAct reference. The anchored rating range suggests substantial headroom for current self-evolving baselines, yet none of the five mechanisms shows consistent monotonic improvement across the eight rounds. Ratings instead fluctuate without forming a stable upward trend, and no evolving configuration reliably surpasses the non-evolving same-backbone reference by the end of evaluation. These results suggest that PTCG-Bench reveals the incompleteness of current self-evolution mechanisms: existing agents still struggle to extract reusable

Table 2: Main harness ablation results. All configurations use the same LLM backbone and mirror-match tournament setting. μ denotes the Glicko-2 rating mean, and $\Delta\mu$ is measured relative to the full harness. Values in parentheses normalize the rating drop by the median rating gap between adjacent LLM backbones in the RQ1 ranking (55 rating points), which is also reported in the final row as a reference scale. Inv. Rate denotes the fraction of illegal or unparsable actions before retry or fallback.

Configuration	Struct.	Mask.	Hist.	μ	$\Delta\mu$	Inv. Rate	Tool Calls
Full Harness (H1+H2+H3)	✓	✓	✓	1726	–	3.3%	78.8
w/o Structured Observation (w/o H1)	✗	✓	✓	1693	−33 (0.6×)	5.2%	68.4
w/o Legal Action Masking (w/o H2)	✓	✗	✓	1608	−118 (2.1×)	15.9%	86.7
No History Context (w/o H3)	✓	✓	✗	1611	−115 (2.1×)	19.0%	385.7
Minimal Harness (w/o H1+H2+H3)	✗	✗	✗	1575	−151 (2.7×)	27.3%	357.6
Median adjacent LLM gap	✓	✓	✓	55	1.0×	–	

strategic knowledge from long-horizon, imperfect-information gameplay without dense rewards or verifiable ground-truth solutions.

To answer RQ3, we examine whether and to what extent harness design affects measured agent performance. We hold the LLM backbone, deck pool, and tournament setting fixed, and ablate key harness choices that govern state representation, action execution, and context management.

Harness modules. We focus on these three modules because they correspond to the main stages through which the harness can influence gameplay: how the agent perceives the state, how it grounds decisions into executable actions, and how it retains temporal information across turns.

- **Observation structure (H1)**: replacing the structured state description with a minimally formatted raw state or game-log representation.
- **Legal-action masking (H2)**: removing the engine-computed legal action set and requiring the agent to infer valid actions and parameters from the current state.
- **Context management (H3)**: removing recent trajectory history while retaining the current state and legal actions.

Main harness ablation. Table 2 reports the main harness ablation results. The full harness uses structured observation, legal-action masking, and context-window-aware history truncation. We compare it against three single-component ablations and a minimal harness. The minimal harness removes all three interface supports: it uses unstructured observation, disables legal-action masking, and provides no recent history.

The ablations show that harness design affects measured capability. Removing structured observation yields a moderate 33-point rating drop and more invalid actions, whereas removing legal-action masking or recent history reduces rating by 118 and 115 points, respectively; the former increases invalid rate, and the latter increases tool calls by nearly fivefold. These drops exceed typical adjacent-backbone gaps in RQ1 and approach several within-family model-tier gaps. Harness design therefore has a substantial effect on measured gameplay strength, making it a central factor in evaluating LLM-agent systems with PTCG-Bench.

4.5 Cross-Deck Generalization

We further evaluate deck-wise mirror-match generalization across five deck archetypes to test whether the measured backbone differences persist under different strategic settings. The main agent ranking is largely preserved across decks, suggesting that the measured differences are not artifacts of a single mirror deck. Details are in Appendix F.

5 Related Work

5.1 Self-Evolving LLM Agents

Recent work has explored self-evolving LLM agents that improve through interaction histories, reflective feedback, and reusable experience. GEPA evolves prompts from reflective trajectory analysis (Agrawal et al., 2025); A-MEM organizes adaptive memories for long-term improvement (Xu et al., 2026); and EvoSkill refines reusable skills from execution failures (Alzubi et al., 2026). These studies suggest that agents can improve not only through model-level training (Da et al., 2025; Huang et al., 2026; Zhang et al., 2026), but also through evolving prompts, memories, and skills.

5.2 Benchmarks for LLM Game Agents

Games are important testbeds because they require perception, memory, planning, and long-

horizon decision-making. LMGame-Bench (Hu et al., 2025) evaluates diverse games with modular perception, memory, and reasoning components, while Orak (Park et al., 2025) covers multiple video-game genres for systematic agent training and evaluation. Other benchmarks emphasize specific challenges: TCG-Bench (Alrashed et al., 2026) is a contamination-resistant and difficulty-scalable multilingual card-game benchmark, while the PokeAgent Challenge (Karten et al., 2026) targets competitive decision-making under partial observability and long-context strategic reasoning.

6 Conclusion

We introduced **PTCG-Bench**, a comprehensive benchmark for studying LLM agents in a realistic and complex game environment. By combining long-horizon imperfect-information gameplay with a longitudinal evaluation protocol, PTCG-Bench provides a controlled setting for evaluating self-evolution under delayed feedback, stochasticity, and opponent interaction. Experiments across ten LLM backbones and multiple self-evolution mechanisms show that current agents can achieve meaningful gameplay performance, but accumulated experience does not yet reliably translate into stable performance gains. Harness ablations further show that interface design materially affects measured capability, underscoring the need to study self-evolution at the agent-system level. These findings highlight PTCG as a challenging testbed for future research on self-evolving agents.

Limitations

PTCG-Bench is designed to support controlled evaluation of LLM-agent systems in a complex trading card game, and the current study focuses on settings that make performance differences interpretable. In particular, we use a fixed deck pool and primarily rely on mirror-match evaluation to reduce asymmetric matchup effects, while studying experience-based updates to prompts, memories, and skills under a fixed LLM backbone. These choices allow us to isolate backbone capability, harness design, and self-evolution mechanisms within the same environment, but they do not exhaust the full space of strategic adaptation. Future extensions could incorporate open deck building, arbitrary cross-deck matchups, longer experience streams, or parameter-level learning while retaining the same controlled evaluation framework.

Ethical Considerations

In constructing PTCG-Bench, we utilized publicly available information from the Pokémon Trading Card Game (PTCG), including card names, mechanics, descriptions and images. All materials are used solely for non-commercial academic research purposes under the principles of fair use / research exceptions in applicable copyright laws. This benchmark is not intended for commercial applications, nor does it claim any affiliation with, sponsorship by, or endorsement from The Pokémon Company, Nintendo, or related entities. We respect the intellectual property rights of the copyright holders and have made efforts to use the data in a transformative manner for benchmarking language models. The dataset will be released only for research use, and we encourage users to comply with all relevant IP regulations.

The data used in PTCG-Bench is derived from fictional card game materials and does not involve human subjects, user-generated personal data, or information that uniquely identifies real individuals. We reviewed the collected data to ensure that it does not contain personally identifying information.

We used ChatGPT to assist with language editing and improving the clarity of the manuscript. The authors reviewed and verified all AI-assisted content and are responsible for the final paper.

Acknowledgments

This work is supported by NSFC (No. 62206056, No. 62322606, No. 62441605, No. 62606466), the CIPSC-SMP-Zhipu Large Model Cross-Disciplinary Fund, and a collaboration funding by FinVolution Group.

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. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*.

Sultan Alrashed, Jianghui Wang, and Francesco Orabona. 2026. Cards against contamination: Tcg-bench for difficulty-scalable multilingual llm reasoning. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 6710–6724.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*.

Anthropic. 2025. Claude's extended thinking. Accessed: May 2026.

Anthropic. 2026. Agent skills: Open standard specification. https://agentskills.io/home. Accessed: May 20, 2026.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research*, 47:253–279.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J K Terry. 2023. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36:73383–73394.

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. 2018. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*, pages 41–75. Springer.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. 2025. Agent-rlvr: Training software engineering agents via guidance and environment rewards. *arXiv preprint arXiv:2506.11425*.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. 2025. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362.

Mark E Glickman. 2012. Example of the glicko-2 system. *Boston University*, 28:2012.

Frank Harary and Leo Moser. 1966. The theory of round robin tournaments. *The American Mathematical Monthly*, 73(3):231–246.

Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. 2025. lmgame-bench: How good are llms at playing games? *arXiv preprint arXiv:2505.15146*.

Renhong Huang, Ning Tang, Jiarong Xu, Yuxuan Cao, Qingqian Tu, Sheng Guo, Bo Zheng, Huiyuan Liu, and Yang Yang. 2026. Policysim: An llm-based agent social simulation sandbox for proactive policy optimization. In *Proceedings of the ACM Web Conference 2026*, pages 4781–4792.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

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

Seth Karten, Andy Luu Nguyen, and Chi Jin. 2025. Pok\'echamp: an expert-level minimax language agent. *arXiv preprint arXiv:2503.04094*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2024. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Sharada Mohanty, Jyotish Poonganam, Adrien Gaidon, Andrey Kolobov, Blake Wulfe, Dipam Chakraborty, Gražvydas Šemetulskis, João Schapke, Jonas Kubilius, Jurgis Pašukonis, et al. 2021. Measuring sample efficiency and generalization in reinforcement learning benchmarks: Neurips 2020 procgen benchmark. *arXiv preprint arXiv:2103.15332*.

Yixin Ou, Wangchunshu Zhou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. 2025. Symbolic learning enables self-evolving agents. *AI Open*.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

Dongmin Park, Minkyu Kim, Beongjun Choi, Junhyuck Kim, Keon Lee, Jonghyun Lee, Inkyu Park, Byeong-Uk Lee, Jaeyoung Hwang, Jaewoo Ahn, et al. 2025. Orak: A foundational benchmark for training and evaluating llm agents on diverse video games. *arXiv preprint arXiv:2506.03610*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. 2025. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8416–8439.

Meta Fundamental AI Research Diplomacy Team et al. 2022. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074.

The Pokémon Company International. 2026. Pokémon trading card game rules. https://www.pokemon.com/static-assets/content-assets/cms2/pdf/trading-card-game/rulebook/par_rulebook_en.pdf. Accessed: May 2026.

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. 2024. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations*, volume 2024, pages 23967–24001.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, et al. 2025. Livebench: A challenging, contamination-limited llm benchmark. In *International Conference on Learning Representations*, volume 2025, pages 91595–91631.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2026. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616.

Siwei Zhang, Yun Xiong, Xi Chen, Zi'an Jia, Renhong Huang, Jiarong Xu, and Jiawei Zhang. 2026. Rapo: Expanding exploration for llm agents via retrieval-augmented policy optimization. *arXiv preprint arXiv:2603.03078*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

A PTCG Rules and Engine Details

A.1 Complete PTCG Rules

PTCG-Bench follows the official rules of the Pokémon Trading Card Game. Each game is played between two players with 60-card decks, hidden hands and decks, public boards, discard piles, and six Prize cards. A player first sets up an Active Pokémon and optional Benched Pokémon, then alternates turns consisting of card draw, legal in-turn actions, and optional attack resolution. During a

turn, players may play Basic Pokémon, evolve eligible Pokémon, attach Energy, play Trainer cards, use Abilities, retreat or switch the Active Pokémon, and attack when the required conditions are satisfied. The engine implements the main card-type mechanics for Pokémon, Energy, Items, Supporters, Tools, and Stadiums, as well as evolution constraints, retreat costs, Special Conditions, Knock Outs, Prize-taking, and standard win conditions. A player wins by taking all Prize cards, by leaving the opponent with no Pokémon in play, or when the opponent cannot draw at the beginning of their turn. For complete official rules and card-specific timing details, we refer readers to the official PTCG rulebook (The Pokémon Company International, 2026).

A.2 Engine Implementation

The PTCG-Bench engine is implemented in Python to facilitate integration with common agent research pipelines. It maintains the full game state, validates legal actions before execution, applies rule-based state transitions, and records complete game trajectories for later analysis. This unified interface allows agents with different prompting, planning, memory, or learning mechanisms to interact with the same environment under reproducible execution.

The implemented card metadata, attacks, abilities, and rule interactions follow the official Pokémon TCG card texts and game rules. Engine correctness is validated through a comprehensive `pytest` suite with approximately 95% code coverage. At the card level, every implemented card is systematically tested for metadata, attack effects, abilities, zone transitions, player-choice flows, damage and Knock Out resolution, and negative or edge-case behavior, such as invalid targets and unmet activation conditions. At the deck level, domain experts conducted complete matches using all five benchmark decks to identify implementation inconsistencies and validate complex cross-card interactions. These automated and expert-driven validations jointly ensure that the engine preserves the gameplay mechanics and strategic interactions required by the benchmark.

As shown in Figure 7, we also provide a React-based frontend for replay, debugging, and qualitative inspection. The frontend visualizes public game states, action histories, card movements, Prize-card progression, and agent decisions over time, making it easier to trace failure cases and interpret agent behavior during full-game evaluation. The frontend is used only for visualization and analysis; game execution and scoring are determined by the Python engine.

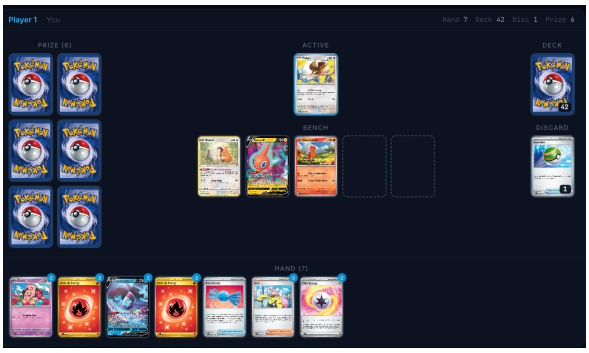

Figure 7: Screenshot of the PTCG-Bench frontend. The interface visualizes our half-field board state, including the Active Pokémon, Bench, Prize cards, Deck, Discard pile, and Hand.

B Deck Details

PTCG-Bench uses a fixed pool of five competitive decks to cover representative strategic archetypes while keeping the evaluation space controlled. Each deck induces a different pattern of board development, resource management, and win-condition execution, requiring agents to adapt across heterogeneous strategic contexts. Table 3 summarizes the composition and strategic profile of each deck.

Together, these decks contain over 100 unique cards and span several axes of strategic diversity, including aggressive versus controlling plans, fast setup versus resource-efficient play, single-core versus multi-core win conditions, and evolution-line versus Basic-Pokémon-centered board construction. This deck pool reduces the risk that evaluation results are dominated by a single play pattern, while remaining small enough to support controlled mirror-match and cross-deck analyses.

C Agent-Environment Interface Details

This appendix provides additional details on the agent-environment interface used in PTCG-Bench. All LLM agents interact with the game engine through the same structured observation format and tool-based action interface. At each decision step, the engine provides the current public and private game state, the legal action set, and any pending card-selection prompt. The field `available_actions` is treated as authoritative:

Table 3: Deck composition and strategic profiles of the five PTCG-Bench decks. #P/#T/#E denote Pokémon, Trainer, and Energy card counts, respectively.

Deck	#P	#T	#E	Plan	Board Structure	Resource Focus	Agent Challenge
Charizard ex	20	32	8	Aggro / scaling	Stage-2 evolution line	Rare Candy setup; Fire Energy acceleration	Timing evolution and prize-based damage scaling
Gardevoir ex	15	34	11	Control / recursion	Stage-2 evolution line	Psychic Energy recovery; discard management	Balancing damage, Energy recursion, and attacker rotation
Miraidon ex	13	29	18	Fast aggro	Basic-Pokémon axis	Electric Energy acceleration; early board filling	Sequencing search, acceleration, and early pressure
Gholdengo ex	19	30	11	Combo / burst	Multi-attacker box	Hand size; Energy discard and recovery	Estimating burst damage and preserving combo resources
Lugia Archeops	20	24	16	Attrition / toolbox	Multi-core box	Special Energy acceleration; Archeops setup	Choosing attackers and allocating Special Energy efficiently

agents are instructed to select only actions listed by the engine. A game turn may therefore consist of multiple LLM decision steps, since each tool call executes a single game action and returns an updated observation.

Agent Configuration. Unless otherwise specified, all agents use the same decoding and interaction settings within each experiment. We fix the system prompt template, observation format, action schema, retry/fallback policy, and context-management rule across compared agents. The default harness maintains recent trajectory context up to the model's maximum context window and truncates older interaction history when necessary. The no-history ablation removes trajectory history entirely while preserving the current state and legal action set. For self-evolving agents, the LLM backbone and action interface are held fixed across rounds; only the persistent state associated with the evolution mechanism, such as reflections, lessons, memories, revised prompts, or skills, is updated.

Tool Interface. The tool interface separates executable game actions from information-query tools. Game-action tools modify the game state and must correspond to legal actions returned by the engine. Query tools do not modify the game state and are used to inspect card text or discard-pile contents. The `activate_skill` tool is enabled only for the skill-library configuration and retrieves stored procedural guidance relevant to the current state. Table 4 summarizes the tool schema exposed to agents.

For actions involving duplicate Pokémon with the same name, the observation may provide field indices to disambiguate targets. Agents are instructed to copy card names, attack names, target names, and indices exactly from the observation or `available_actions`. Invalid, unparsable, or unavailable actions are rejected by the game engine and counted in the invalid action rate before retry or fallback.

D Self-Evolution Mechanism Details

We compare self-evolution mechanisms at the level of how prior gameplay experience is converted into persistent decision-making state. This design keeps the PTCG interaction interface fixed while varying the form of experience retained across games. In particular, all evolving configurations share the same backbone model, base harness, action tools, and evaluation anchors; they differ only in whether past trajectories are converted into reflections, distilled lessons, retrieved memories, revised prompt instructions, or structured skills.

PTCG requires mechanism-level adaptation rather than direct reuse of procedures designed for short-horizon or verifiable tasks. A complete trajectory contains many tool-mediated decisions, hidden opponent information, stochastic card draws, and a delayed game outcome. We therefore use completed games and round-level trajectory collections as the experience source for persistent updates, while later matches remain the evaluation signal for whether those updates transfer to new gameplay.

Reflexion. Following reflection-based verbal reinforcement (Shinn et al., 2023), the agent produces a free-form reflection after each game. The reflection summarizes mistakes, missed opportunities, and strategic lessons inferred from the trajectory and terminal outcome. These reflections remain episodic: they preserve game-specific feedback

Table 4: Summary of the PTCG-Bench tool interface. Required arguments are shown in parentheses; optional disambiguation indices are omitted for brevity.

Category	Tool	Purpose and Required Arguments
Attack	`attack`	Use the Active Pokémon's attack; requires `source_card`, `attack_name`.
Board setup	`play_pokemon`	Play a Basic Pokémon to the Active Spot or Bench; requires `source_card`, `position`.
Evolution	`evolve_pokemon`	Evolve a Pokémon in play; requires `source_card`, `target_card`.
Energy	`attach_energy`	Attach one Energy card from hand; requires `source_card`, `target_card`.
Trainer	`use_supporter`	Play a Supporter card; requires `source_card`.
Trainer	`use_item`	Play an Item card; requires `source_card`.
Trainer	`use_tool`	Attach a Pokémon Tool; requires `source_card`, `target_card`.
Stadium	`put_stadium`	Play a Stadium card; requires `source_card`.
Stadium	`discard_stadium`	Discard the current Stadium when legally allowed; requires `source_card`.
Stadium	`use_stadium`	Activate an in-play Stadium effect; requires `source_card`.
Ability	`use_ability`	Activate a Pokémon Ability; requires `source_card`, optionally `ability_name`.
Switching	`retreat`	Retreat the Active Pokémon; requires `source_card`.
Selection	`choose_card`	Resolve a card-selection prompt; requires `chosen_cards`.
Turn control	`pass_turn`	End the current turn without further action.
Information	`query_card`	Query exact card text and metadata; requires `card_id`.
Information	`query_discard`	Inspect a player's discard pile; requires `player`.
Skill retrieval	`activate_skill`	Load stored skill guidance; requires `name`, optionally `resource`.

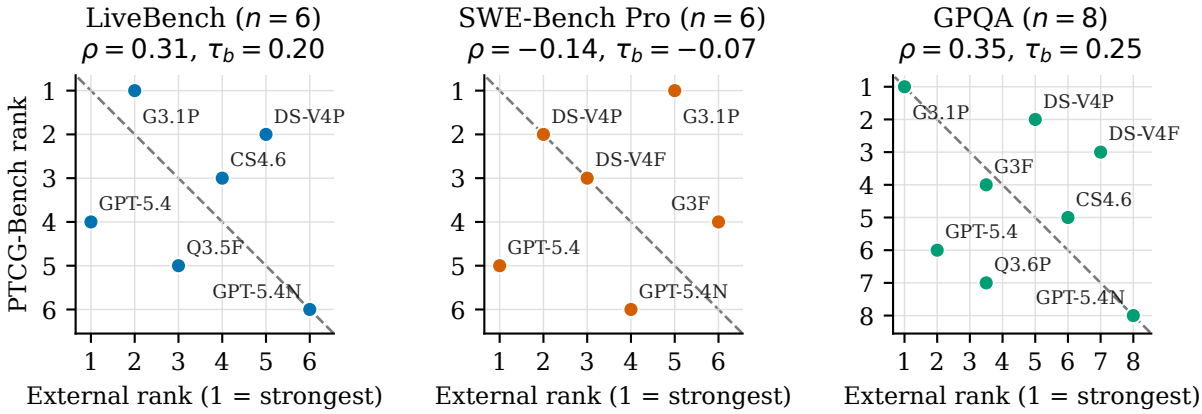

Figure 8: Rank agreement between PTCG-Bench and external LLM evaluations on available overlapping model subsets. Each point is one shared model, and the dashed diagonal marks identical orderings within the overlap subset. LiveBench and GPQA show weak positive agreement with PTCG-Bench, while the collected SWE-Bench Pro ordering shows no clear agreement.

and are prepended to later game contexts so the agent can condition subsequent decisions on recent self-critique. This configuration tests whether lightweight post-game reflection alone can convert individual failures into better future play.

ExpeL. ExpeL-style evolution (Zhao et al., 2024) aggregates experience before updating persistent state. After each evaluation round, the agent distills accumulated trajectories into reusable strategic lessons rather than storing one reflection per game. The lessons aim to abstract recurring patterns across games, such as common planning failures or generally useful decision rules, and are provided to later games as compact guidance. This configuration separates cross-game lesson distillation from the more episodic feedback used by Reflexion.

Long-Term Memory. The long-term memory configuration follows memory-based agent designs (Park et al., 2023; Packer et al., 2023; Tan et al., 2025; Xu et al., 2026). It stores prior gameplay episodes externally and retrieves memories judged relevant to the current game state during later play. To reduce accumulation of low-level trajectory detail, periodic summarization compresses past episodes into higher-level strategic notes while preserving access to experience-derived guidance. This configuration tests whether retrieval over prior experience is more useful than injecting a fixed set of lessons into every later context.

Prompt Evolution. Prompt evolution treats the strategy prompt itself as the persistent state (Madaan et al., 2023; Wang et al., 2024; Yuksekgonul et al., 2025). After each evaluation round, the agent analyzes the current strategy prompt together with recent game summaries, identifies recurring failure modes, and revises the strategy component of its instruction prompt. The revised prompt is then used in subsequent rounds under the same interaction interface. Unlike memory-based variants, this mechanism encodes experience into global decision principles that shape all later decisions rather than retrieving state-specific past episodes.

Skill Library Evolution. Skill-library evolution maintains a persistent collection of structured strategy skills (Anthropic, 2026; Alzubi et al., 2026). Skills are distilled from prior games and describe an activation condition, a strategic objective, and recommended decision principles. During later play, relevant skills are retrieved when their guidance matches the current decision context. This configuration provides a more modular form of persistent knowledge than a monolithic revised prompt: different reusable skills can target different strategic situations while remaining grounded in accumulated gameplay experience.

Across these configurations, the persistent state is updated only by the corresponding self-evolution mechanism. The environment state, legal-action interface, and fixed anchor opponents are unchanged across rounds, allowing the longitudinal results in Section 4.3 to compare how different forms of retained experience affect subsequent gameplay strength.

E External Benchmark Rank Agreement

We compare the PTCG-Bench backbone ordering with model scores reported by LiveBench, SWE-Bench Pro, and GPQA (White et al., 2025; Deng et al., 2025; Rein et al., 2023). Each comparison is computed only on the subset of evaluated PTCG-Bench backbones with an available score for that external benchmark. We rerank PTCG-Bench and the external benchmark within that overlap subset, with rank 1 denoting the strongest model, and report both Spearman ρ and Kendall τ_b rank correlations in Figure 8. Tied external scores receive average ranks; this affects the two GPQA entries with score 90.4.

On the overlapped model subsets, LiveBench yields Spearman $\rho = 0.31$ and Kendall $\tau_b = 0.20$, GPQA yields $\rho = 0.35$ and $\tau_b = 0.25$, and SWE-Bench Pro yields $\rho = -0.14$ and $\tau_b = -0.07$. This suggests that PTCG-Bench measures critical aspects of strategic reasoning in complex, long-horizon game scenarios that are insufficiently captured by these existing benchmarks.

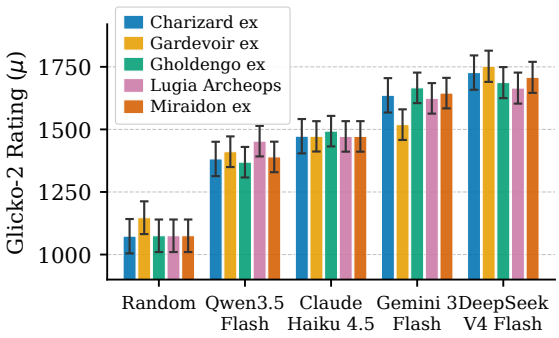

Figure 9: Cross-deck mirror-match generalization results. Each group of bars represents agents evaluated under one deck archetype, with both players using the same deck within that setting. Bar height is the Glicko-2 rating μ, and error bars denote rating deviation ϕ.

F Cross-Deck Generalization

We test cross-deck generalization with deck-wise mirror matches, varying the shared deck archetype across settings while avoiding asymmetric matchup effects. Figure 9 reports results across five representative decks: Charizard ex, Gardevoir ex, Miraidon ex, Gholdengo ex, and Lugia Archeops.

The RQ1 ranking is largely preserved across all five deck archetypes. Although individual ratings fluctuate with deck mechanics, these variations do not produce systematic rank reversals, indicating that the measured backbone differences are not artifacts of a single mirror deck. This controlled analysis tests cross-archetype robustness rather than asymmetric matchups between different decks.

G Prompt Templates

The following prompt is used for the non-evolving ReAct agent and as the base interaction template for evolving variants. The full prompt additionally includes concise reminders of PTCG-specific rules, including evolution restrictions, first-turn restrictions, retreat constraints, Trainer-card limits, damage calculation, Special Conditions, and Knock Out resolution.

Role. You are a Pokémon TCG battle agent using the ReAct pattern. At each decision step, analyze

the current game state and then call exactly one tool.

Objective. Win the game by taking all Prize cards, Knocking Out all of the opponent's Pokémon in play, or causing the opponent to be unable to draw at the beginning of their turn.

Turn Structure. Each turn consists of drawing a card, taking legal in-turn actions, and optionally attacking. Legal in-turn actions include playing Basic Pokémon, evolving eligible Pokémon, attaching one Energy card, playing Trainer cards, using Abilities, retreating the Active Pokémon, and passing the turn. Attacking ends the turn after damage, effects, Knock Outs, Prize-taking, and win checks are resolved.

Observation Reading. The current game state is provided as a structured observation. The field `available_actions` is authoritative: choose only actions listed there. If `choosing_card` is true, respond to that selection prompt before taking any other action. Use `opponent_last_turn_actions` to infer recent changes. Your hand is visible, while the opponent's hand and deck remain hidden.

Decision Pattern. Before acting, consider HP and damage, Energy attachments, available attacks, Prize counts, immediate threats, and possible win conditions. Then call the appropriate tool. A full game turn may require multiple decision steps because each game action is executed separately.

Action Discipline. Call at most one game-action tool in each response. Use exact card names, attack names, targets, and indices copied from the observation or `available_actions`. Do not invent actions, targets, card names, or attack names. If an intended action is not listed in `available_actions`, it is not currently legal. If an action fails or produces an unexpected result, inspect the updated observation and query relevant cards before proceeding.

Tools. Use query tools, such as `query_card` and `query_discard`, when card text, discard contents, attack effects, retreat costs, or other details are uncertain. Use game-action tools, such as `attack`, `play_pokemon`, `evolve_pokemon`, `attach_energy`, `use_supporter`, `use_item`, `use_tool`, `put_stadium`, `retreat`, `use_ability`, `choose_card`, and `pass_turn`, only when the corresponding action is legal in the current observation.

H Implementation Details

We used the OpenRouter API for model inference. Specifically, we set temperature to 0.7 and maximum output length to 2048 tokens. Unless otherwise specified, other API parameters were kept at their default values.