Training-Free Continuous Learning for Tool-Using Agents via GRPO Semantic Experience Management

Hanzo AI Research
Hanzo AI Inc (Techstars '17), Los Angeles, CA
`research@hanzo.ai`

February 2026

Abstract

We present **Agent-GRPO**, a training-free continuous learning framework that extends Group-Relative Policy Optimization (GRPO) from text-output optimization to the agent tool-use setting. While Active Semantic Optimization (ASO) extracts generalizable patterns from grouped language model rollouts, agent trajectories carry strictly richer structure: sequences of tool selections, parameter choices, intermediate outputs, and explicit success or failure signals that are unavailable from text alone. Agent-GRPO adapts the three-stage semantic extraction pipeline of Tencent YouTou Lab [2025] to this richer domain, constructing a *semantic experience library* $\mathcal{E}$ whose entries encode tool-use strategies, failure patterns, and domain conventions distilled from past agent sessions.

We formalize the tool-use trajectory as a typed tuple $\tau = \big(x,\, \{(t_i, \boldsymbol{p}_i, o_i, s_i)\}_{i=1}^{K},\, R\big)$, define a composite reward combining binary tool success and user acceptance signals, and prove that the expected fraction of beneficial experiences in $\mathcal{E}$ grows monotonically under mild regularity conditions (Theorem 1). We further derive a sample complexity bound of $O(K/\varepsilon^2)$ sessions to achieve ε-improvement over the baseline (Theorem 3).

Empirically, a Hanzo Dev agent equipped with Agent-GRPO achieves a **31.4%** reduction in tool-call errors and a **22.7%** improvement in task completion rate on the HB-Bench tool-use benchmark after 50 sessions of online experience collection, at a total adaptation cost below \$20 in API calls—orders of magnitude cheaper than any fine-tuning alternative. We also show how Agent-GRPO integrates naturally with the four-loop self-improvement architecture of ASO, making the feedback loop between telemetry, active learning, reflection, and maintenance fully closed.

1 Introduction

Tool-using agents—systems that interact with external tools such as code interpreters, file systems, search engines, and APIs—have emerged as a primary deployment paradigm for large language models [Schick et al., 2023, Yang et al., 2024, Yao et al., 2023]. Unlike pure text-generation tasks, agent behavior is evaluated not merely on the quality of prose but on the correctness of tool invocations, the appropriateness of tool selection, and the agent's ability to recover from errors encountered during execution. This richer evaluation surface creates both new challenges and new opportunities for adaptation.

The Adaptation Gap for Agents. Existing continuous learning works for LLMs focus on text-quality metrics—pass rates on coding benchmarks, BLEU scores, preference rankings. The

Active Semantic Optimization (ASO) framework [Hanzo AI Research, 2026a] takes a training-free approach, extracting semantic advantages from grouped rollouts and applying them as Product-of-Experts decoding factors. Yet ASO and its predecessors treat the model output as a monolithic text sequence. An agent trajectory is structurally different: it is a *sequence of decisions* interleaved with environmental feedback, where each decision (which tool to call, with what parameters) has an observable outcome that serves as a rich local reward signal.

This structural difference is consequential. Consider an agent that repeatedly chooses `bash` to parse JSON when a dedicated `read_json` tool would be faster, more reliable, and incur lower token cost. This failure to prefer the specialized tool will not manifest as a text-quality degradation that text-level GRPO would catch; the output may look correct even when the tool choice was suboptimal. Conversely, an agent that learns to prefer `read_json` for JSON parsing can apply this lesson to future tasks without any gradient update—provided the lesson is properly encoded and retrieved.

The Semantic Experience Library. Our central contribution is the **semantic experience library** $\mathcal{E}$, a persistent, embedding-indexed store of natural-language experiences extracted from agent trajectories via an adapted GRPO pipeline. Each entry $e \in \mathcal{E}$ encodes a reusable lesson: a statement about when to use which tool, how to structure parameters for a particular API, or which failure patterns to anticipate in a given domain. At the start of each new session, the top-K experiences most semantically similar to the incoming task are injected into the agent's system prompt, providing explicit behavioral guidance without modifying model weights.

Agent-GRPO. We formalize this approach as **Agent-GRPO**, which extends Training-Free GRPO (TF-GRPO) [Hanzo AI Research, 2026a] to agent trajectories through three technical innovations:

1. **Trajectory-level rewards:** A composite reward function combining binary tool success signals and time-delayed user acceptance signals, aggregated over the full trajectory.

2. **Three-stage extraction pipeline:** Adapted from Tencent YouTou Lab [2025], the pipeline summarizes individual trajectories (Stage 1), extracts group advantages by comparing successful and unsuccessful groups (Stage 2), and consolidates batch operations to produce experience library updates (Stage 3).

3. **Memory management:** Four compression strategies—diversity, importance, temporal decay, and hybrid—that maintain library quality as $|\mathcal{E}|$ grows, preventing staleness and redundancy while preserving high-utility experiences.

Contributions.

1. A formal model of agent tool-use trajectories as typed tuples with observable success signals (Section 3.1).

2. A composite reward function and group-relative advantage computation adapted for agent trajectories (Section 3.2).

3. The complete Agent-GRPO algorithm with formal pseudocode and correctness guarantees (Section 3.7).

4. A semantic memory manager with four compression strategies and provably monotone quality improvement (Sections 4 and 7).
5. Integration with ASO's four-loop self-improvement architecture, closing the feedback loop from telemetry to library update (Section 6).
6. Empirical evaluation demonstrating state-of-the-art results on HB-Bench at sub-$20 adaptation cost (Section 8).

Paper Outline. Section 2 reviews GRPO, TF-GRPO, and the YouTou-Agent pipeline. Section 3 formalizes Agent-GRPO. Section 4 presents the semantic memory manager. Section 5 describes agent-specific adaptations. Section 6 describes integration with self-improvement loops. Section 7 provides theoretical analysis. Section 8 presents empirical evaluation. Section 9 discusses related work. Section 11 concludes.

2 Background

2.1 Group-Relative Policy Optimization

GRPO [Shao et al., 2024] extends the REINFORCE algorithm [Williams, 1992] by computing advantages relative to a group of simultaneously generated rollouts, eliminating the need for a learned value baseline. Given task x, a group of G rollouts $\{y^{(i)}\}_{i=1}^{G} \sim \pi_\theta(\cdot \mid x)$, and associated scalar rewards $\{r^{(i)}\}_{i=1}^{G}$, the group-relative advantage is:

$$A^{(i)} = \frac{r^{(i)} - \mu_G}{\sigma_G + \epsilon}, \quad \mu_G = \frac{1}{G}\sum_{j=1}^{G} r^{(j)}, \quad \sigma_G = \sqrt{\frac{1}{G-1}\sum_{j=1}^{G}(r^{(j)} - \mu_G)^2}, \tag{1}$$

with $\epsilon = 10^{-8}$ for numerical stability. Standard GRPO then performs a clipped policy gradient update on θ:

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{G}\sum_{i=1}^{G} \min\Big(\rho^{(i)}A^{(i)},\ \text{clip}(\rho^{(i)}, 1-\delta, 1+\delta)A^{(i)}\Big) + \lambda_{\text{KL}}\ \text{KL}\big[\pi_\theta \| \pi_{\text{ref}}\big], \tag{2}$$

where $\rho^{(i)} = \pi_\theta(y^{(i)} \mid x)/\pi_{\text{old}}(y^{(i)} \mid x)$ is the importance weight and π_{ref} is a reference policy.

2.2 Training-Free GRPO (TF-GRPO)

TF-GRPO [Hanzo AI Research, 2026a] replaces the gradient update in Equation (2) with a semantic extraction and compression pipeline. The key insight is that the advantages $\{A^{(i)}\}$ encode *what makes one solution better than another* in a form that can be distilled into reusable natural-language patterns, called *semantic advantages*:

$$S^{(i)} = \mathcal{F}_{\text{LLM}}\Big(x,\ \{y^{(j)}, r^{(j)}\}_{j=1}^{G},\ i\Big), \tag{3}$$

where $\mathcal{F}_{\text{LLM}}$ denotes an LLM-based extraction call that produces a structured text description of why rollout i outperformed or underperformed the group mean.

These semantic advantages are then compressed into 1-bit token-level expert factors and applied at decode time via a Product-of-Experts (PoE) distribution [Hinton, 2002]:

$$\pi_{\text{ASO}}(y_t \mid x, y_{<t}) \propto \pi_\theta(y_t \mid x, y_{<t}) \prod_{m=1}^{M} \phi_m(y_t \mid x, y_{<t})^{\eta_m}, \tag{4}$$

where $\eta_m = \log(q_m/(1-q_m))$ and $q_m \in (0,1)$ is the quality attestation score for prior m. The composite reward used in ASO is:

$$g^{(i)} = \alpha\, r^{(i)} + \beta\, u^{(i)}, \tag{5}$$

where $r^{(i)}$ is the extrinsic reward (test pass rate) and $u^{(i)}$ is the epistemic reward (solution diversity and novelty).

2.3 The YouTou-Agent Three-Stage Pipeline

Tencent YouTou Lab [2025] propose a three-stage pipeline for extracting reusable experiences from groups of LLM trajectories:

Stage 1 – Trajectory Summarization. Each individual trajectory $\tau^{(i)}$ in a group is summarized into a structured description of the reasoning steps, decisions, and outcomes. The summary is constrained to at most 32 words and must identify the key decision that differentiated this trajectory from a baseline.

Stage 2 – Group Advantage Extraction. Given G summaries from Stage 1, the group is analyzed as a unit. The extractor identifies patterns that distinguish high-reward trajectories from low-reward ones, and produces *operations* on an experience library $\mathcal{E}$: ADD new experiences, MODIFY existing ones, DELETE outdated entries, or MERGE near-duplicate entries.

Stage 3 – Batch Consolidation. Multiple Stage-2 outputs (from different task groups within a session) are consolidated to remove redundancies, resolve conflicts, and ensure no single experience exceeds 32 words. The final set of operations is applied atomically to $\mathcal{E}$.

The original YouTou-Agent framework applies this pipeline to text-only LLM outputs. Agent-GRPO adapts all three stages to the richer structure of tool-use trajectories.

2.4 Agent Trajectories vs. Text Trajectories

A text-only trajectory $\tau_{\text{text}} = (x, y, r)$ consists of a task x, a text output y, and a scalar reward r. An agent trajectory carries strictly more information:

- **Tool selection sequence:** which tools were called, and in what order.
- **Parameter choices:** the arguments passed to each tool, reflecting the agent's understanding of the task.
- **Intermediate outputs:** the return values of each tool call, observable during execution.
- **Per-step success signals:** whether each tool call succeeded or raised an error, providing dense local reward information.
- **Recovery patterns:** how the agent responded to errors—by retrying, switching tools, or escalating.

This richer signal structure makes agent trajectories both more informative and more complex to process. The three-stage pipeline must be extended to handle sequences of (tool, parameters, output, success) tuples rather than single text outputs.

3 Agent-Level GRPO

3.1 Tool-Use Trajectory Definition

Definition 1 (Tool-Use Trajectory)**.** *A tool-use trajectory is a tuple*

$$\tau = \big(x,\ \boldsymbol{\xi},\ R\big), \quad \boldsymbol{\xi} = \big((t_1, \boldsymbol{p}_1, o_1, s_1),\ \ldots,\ (t_K, \boldsymbol{p}_K, o_K, s_K)\big), \tag{6}$$

where:

- $x \in \mathcal{X}$ *is the natural-language task description,*
- $K \geq 0$ *is the number of tool calls made,*
- $t_i \in \mathcal{T}$ *is the name of the i-th tool invoked from tool registry* $\mathcal{T}$*,*
- $\boldsymbol{p}_i \in \mathcal{P}(t_i)$ *is the parameter dictionary for tool* t_i*,*
- $o_i \in \mathcal{O}$ *is the output returned by tool* t_i*,*
- $s_i \in \{0, 1\}$ *is the binary success indicator (*$s_i = 1$ *iff the call completed without error and produced well-formed output),*
- $R \in [0, 1]$ *is the trajectory-level composite reward defined in Section 3.2.*

Remark 1. *The length* K *is not fixed a priori; it varies across trajectories in the same group. This is fundamentally different from the text setting where all rollouts have a common output space* $\mathcal{V}^*$*. Agent-GRPO must compare trajectories of potentially different lengths, which motivates the summarization approach in Stage 1.*

Let $\mathcal{T}$ be a finite registry of $|\mathcal{T}|$ available tools. We partition $\mathcal{T}$ into *domains* $\mathcal{D} = \{d_1, \ldots, d_L\}$ based on functionality: d_{fs} (file system), d_{shell} (shell execution), d_{search} (web/code search), d_{k8s} (Kubernetes operations), and so on. Domain membership informs the scoped retrieval of experiences (Section 4.2).

3.2 Reward Computation

The trajectory-level reward R decomposes into two observable components:

Tool Success Rate. The per-trajectory tool success rate is the fraction of tool calls that completed without error:

$$\rho_s(\tau) = \frac{1}{K} \sum_{i=1}^{K} s_i \in [0, 1]. \tag{7}$$

For the degenerate case $K = 0$ (task completed without tool use), we set $\rho_s = 1$.

User Acceptance Signal. The user acceptance signal $\rho_a(\tau) \in \{0, 1\}$ is determined by observing whether the user corrects or rejects the agent's final output within a time window Δt:

$$\rho_a(\tau) = \begin{cases} 1 & \text{if user does not correct output within } \Delta t = 120\,\text{s}, \\ 0 & \text{if user explicitly corrects or rejects output.} \end{cases} \tag{8}$$

This weak, delayed supervision signal is the only human signal required by Agent-GRPO; no explicit reward labeling is needed.

Composite Reward. The composite trajectory reward is:

$$R(\tau) = w_1\,\rho_s(\tau) + w_2\,\rho_a(\tau), \tag{9}$$

with default weights $w_1 = 0.4$, $w_2 = 0.6$ (user acceptance is weighted more heavily as it reflects task-level quality, not just mechanical correctness).

Remark 2. *The weights w_1 and w_2 can be tuned per deployment. In automated pipelines without a human in the loop, one may set $w_2 = 0$ and rely entirely on ρ_s. In interactive settings, $w_2 > 0$ captures the user's implicit preference signal.*

Step-Level Rewards. For Stage-1 summarization (Section 3.4), we also compute step-level rewards $r_i = s_i$ for each tool call i, which enable the extractor to identify exactly which calls were beneficial or harmful.

3.3 Group-Relative Advantage for Agent Trajectories

Given a group of G trajectories $\{\tau^{(j)}\}_{j=1}^{G}$ on the same task x, the group-relative advantage of trajectory $\tau^{(i)}$ is defined analogously to Equation (1):

$$A^{(i)} = \frac{R(\tau^{(i)}) - \mu_G}{\sigma_G + \epsilon}, \quad \mu_G = \frac{1}{G}\sum_{j=1}^{G} R(\tau^{(j)}), \quad \sigma_G = \sqrt{\frac{1}{G-1}\sum_{j=1}^{G}\big(R(\tau^{(j)}) - \mu_G\big)^2}. \tag{10}$$

Trajectories with $A^{(i)} > 0$ are identified as "positive examples" for Stage-2 extraction; those with $A^{(i)} < 0$ are "negative examples." The group is required to contain at least one positive and one negative example for meaningful advantage extraction; groups where all trajectories achieve the same reward are discarded.

3.4 Stage 1: Tool Trajectory Summarization

Stage 1 processes each trajectory $\tau^{(i)}$ independently, producing a structured summary $\hat{\tau}^{(i)}$ suitable for group comparison.

Definition 2 (Trajectory Summary)**.** *A trajectory summary $\hat{\tau}$ is a natural-language description constrained to at most $W_{\max} = 64$ words that identifies: (a) the task type and domain, (b) the primary tool selection strategy employed, (c) the most significant parameter choice that affected outcome, and (d) the root cause of any failures.*

The summary is produced by the following LLM call:

Prompt template for Stage 1: `You are analyzing a tool-use trajectory for task: {x}. The trajectory made {K} tool calls with overall reward {R}. The step-by-step calls were: {`ξ`}. Summarize in at most 64 words: (1) what tool selection strategy was used, (2) what parameter choices were made for key tools, (3) what failures occurred and why, (4) what the outcome was. Focus on patterns generalizable to similar future tasks.`

The summary $\hat{\tau}^{(i)}$ retains the advantage sign: we annotate it as $\hat{\tau}^{(i)+}$ if $A^{(i)} > 0$ and $\hat{\tau}^{(i)-}$ if $A^{(i)} < 0$.

3.5 Stage 2: Group Advantage Extraction

Stage 2 operates on the full group of summaries $\{\hat{\tau}^{(j)}\}_{j=1}^{G}$ and produces a set of *operations* $\Omega^{(g)}$ on the experience library.

Definition 3 (Experience Library Operation). *An operation $\omega \in \Omega$ is one of:*

- ADD(e_{new})*: insert a new experience e_{new} into $\mathcal{E}$,*
- MODIFY(e_{id}, e_{new})*: replace experience with identifier e_{id},*
- DELETE(e_{id})*: remove experience e_{id} from $\mathcal{E}$,*
- MERGE$(e_{id_1}, e_{id_2}, e_{new})$*: replace two near-duplicate experiences with a single consolidated entry.*

The Stage-2 LLM prompt compares positive and negative group summaries:

Prompt template for Stage 2: `You are extracting generalizable tool-use lessons. Below are SUCCESSFUL trajectories (high reward): {`$\hat{\tau}^{(i)+}$`}. And FAILED trajectories (low reward): {`$\hat{\tau}^{(i)-}$`}. Current experience library: {`$\mathcal{E}_{\texttt{relevant}}$`}. Produce operations (add/modify/delete/merge) that update the library. Each new/modified experience must: (a) be `$\leq$`32 words, (b) be actionable and specific, (c) explain WHEN and HOW to apply. Format: OPERATION | experience_id (if modify/delete/merge) | text.`

The output $\Omega^{(g)}$ from Stage 2 contains all proposed library modifications arising from task group g.

3.6 Stage 3: Batch Consolidation

Stage 3 merges the operation sets $\{\Omega^{(g)}\}_{g=1}^{N_g}$ from all N_g task groups processed in the current session, producing a final consolidated operation set Ω^* that is applied atomically to $\mathcal{E}$.

Consolidation proceeds in three passes:

1. **Conflict resolution:** If two groups issue conflicting MODIFY or DELETE operations on the same e_{id}, the operation from the group with higher mean group advantage $\bar{A}^{(g)} = \frac{1}{G}\sum_i |A_g^{(i)}|$ takes precedence.
2. **Deduplication:** ADD operations whose proposed text has cosine similarity above $\theta_{\text{dup}} = 0.85$ with an existing library entry are converted to MODIFY operations.
3. **Length enforcement:** Any experience text exceeding 32 words is compressed via an LLM call: `Summarize in `$\leq$`32 words, preserving the actionable core: {text}.`

3.7 The AGENT-GRPO Algorithm

Algorithm 1 presents the complete Agent-GRPO procedure.

Algorithm 1 AGENT-GRPO: Agent-Level Training-Free GRPO

```
Require: Session trajectories $\mathcal{S} = \{(\tau_g^{(j)})\}_{g,j}$, experience library $\mathcal{E}$, group size $G$, consolidation
    threshold $\theta_{\text{dup}}$
Ensure: Updated experience library $\mathcal{E}'$
 1: $\Omega \leftarrow \emptyset$                                    ▷ Accumulated operations
 2: for each task group $g$ in $\mathcal{S}$ do
 3:     $\{\tau^{(j)}\}_{j=1}^{G} \leftarrow$ trajectories for group $g$
 4:     // Compute rewards and advantages
 5:     for $j = 1$ to $G$ do
 6:         $R^{(j)} \leftarrow w_1\,\rho_s(\tau^{(j)}) + w_2\,\rho_a(\tau^{(j)})$
 7:     end for
 8:     $\mu_G, \sigma_G \leftarrow$ GroupStats($\{R^{(j)}\}$)
 9:     if $\sigma_G < \epsilon$ then
10:         continue                       ▷ No variance: group provides no learning signal
11:     end if
12:     for $j = 1$ to $G$ do
13:         $A^{(j)} \leftarrow (R^{(j)} - \mu_G)/(\sigma_G + \epsilon)$
14:     end for
15:     // Stage 1: Trajectory Summarization
16:     for $j = 1$ to $G$ do
17:         $\hat{\tau}^{(j)} \leftarrow$ Summarize($\tau^{(j)}$, sign($A^{(j)}$))
18:     end for
19:     // Stage 2: Group Advantage Extraction
20:     $\mathcal{E}_{\text{rel}} \leftarrow$ Retrieve($\mathcal{E}$, $x_g$, $K_{\text{stage2}} = 5$)
21:     $\Omega^{(g)} \leftarrow$ ExtractAdvantages($\{\hat{\tau}^{(j)}\}$, $\mathcal{E}_{\text{rel}}$)
22:     $\Omega \leftarrow \Omega \cup \Omega^{(g)}$
23: end for
24: // Stage 3: Batch Consolidation
25: $\Omega^* \leftarrow$ Consolidate($\Omega$, $\theta_{\text{dup}}$)
26: // Apply operations
27: $\mathcal{E}' \leftarrow$ Apply($\mathcal{E}$, $\Omega^*$)
28: return $\mathcal{E}'$
```

Complexity. Let N_g be the number of task groups, G the group size, and L_{LLM} the cost of a single LLM extraction call. Stage 1 requires $N_g G$ LLM calls. Stage 2 requires N_g calls. Stage 3 requires $O(|\Omega|)$ comparisons plus at most $|\Omega|$ LLM compression calls. Total cost: $O(N_g(G+1))$ LLM calls, typically 50–200 calls per session at approximately \$0.05–\$0.10 each, yielding a per-session adaptation cost of \$2.50–\$20.

4 Semantic Memory Manager

4.1 Experience Representation

Definition 4 (Experience Entry)**.** *An experience entry $e \in \mathcal{E}$ is a tuple:*

$$e = \big(\boldsymbol{id},\ \boldsymbol{text},\ c,\ d,\ \boldsymbol{v},\ u,\ k\big), \tag{11}$$

where:

- $\boldsymbol{id} \in \mathbb{N}$ *is a unique identifier,*
- $\boldsymbol{text} \in \Sigma^{\leq 32}$ *is the natural-language experience (at most 32 words),*
- $c \in (0,1)$ *is the confidence score (initialized to* $c_0 = 0.5$ *for new entries, updated via online learning),*
- $d \in \mathcal{D}$ *is the domain label,*
- $\boldsymbol{v} \in \mathbb{R}^{384}$ *is the sentence embedding produced by* `all-MiniLM-L6-v2` *[Reimers and Gurevych, 2019],*
- $u \in \mathbb{N}$ *is the usage count (number of times retrieved and applied),*
- $k \in \mathbb{N}$ *is the epoch of creation (session index).*

The confidence score is updated via a Bayesian-flavored online rule after each session in which experience e was applied:

$$c \leftarrow \begin{cases} \min(c + \gamma(1-c),\ c_{\max}) & \text{if applying } e \text{ improved task reward,} \\ \max(c - \gamma c,\ c_{\min}) & \text{if applying } e \text{ did not improve reward,} \end{cases} \tag{12}$$

where $\gamma = 0.1$ is the learning rate, $c_{\min} = 0.05$, and $c_{\max} = 0.95$.

4.2 Retrieval

At the start of each agent session with task x, the memory manager retrieves the top-K experiences most relevant to x.

Query Embedding. The task query is embedded: $\boldsymbol{q} = \mathrm{Enc}(x) \in \mathbb{R}^{384}$.

Domain-Filtered Cosine Similarity. For each experience $e \in \mathcal{E}$, the retrieval score is:

$$\mathrm{score}(e, x) = \begin{cases} \dfrac{\boldsymbol{q} \cdot \boldsymbol{v}_e}{\|\boldsymbol{q}\|\,\|\boldsymbol{v}_e\|} & \text{if } d_e \in D_x \text{ or } D_x = \emptyset, \\ -\infty & \text{otherwise,} \end{cases} \tag{13}$$

where $D_x \subseteq \mathcal{D}$ is the set of domains inferred from the task (empty if domain is ambiguous, in which case all domains are eligible). The top-K experiences by score are returned.

Context Injection. Retrieved experiences are prepended to the agent's system prompt in ranked order:

```
[G0] {e[0].text}
[G1] {e[1].text}
⋮
[G{K-1}] {e[K-1].text}
```

This format signals to the model that the bracketed items are learned guidelines ("G" for guideline), distinguishing them from task instructions and conversation history. Default $K = 5$ retrieved experiences per session.

4.3 Confidence-Weighted Retrieval Score

The pure cosine similarity score can be augmented by confidence weighting to prioritize well-validated experiences:

$$\tilde{\text{score}}(e, x) = (1 - \lambda)\,\frac{\boldsymbol{q} \cdot \boldsymbol{v}_e}{\|\boldsymbol{q}\|\,\|\boldsymbol{v}_e\|} + \lambda\, c_e, \tag{14}$$

with $\lambda = 0.2$. This blends semantic relevance with empirical reliability.

4.4 Memory Compression

As the experience library grows, unconstrained addition would eventually degrade retrieval precision (too many near-duplicate or stale entries) and increase context injection cost. The memory manager applies one of four compression strategies when $|\mathcal{E}| > N_{\max}$ (default $N_{\max} = 500$).

Strategy 1: Diversity Compression. Apply K-means clustering to the embedding matrix $V = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{|\mathcal{E}|}]^\top$. Within each cluster $\mathcal{C}_k$, retain only the experience with highest confidence:

$$e_k^* = \arg\max_{e \in \mathcal{C}_k} c_e. \tag{15}$$

This ensures the surviving library spans the full semantic space of learned patterns.

Strategy 2: Importance Compression. Sort experiences by confidence and discard the bottom-p fraction:

$$\mathcal{E}' = \big\{e \in \mathcal{E} : c_e \geq Q_{1-p}(\{c_e : e \in \mathcal{E}\})\big\}, \tag{16}$$

where Q_{1-p} is the $(1-p)$-quantile. Default $p = 0.2$ (discard lowest 20% by confidence).

Strategy 3: Temporal Decay Compression. Assign each experience a time weight via exponential decay:

$$w_t(e) = \exp\big(-\lambda(k_{\text{now}} - k_e)\big), \tag{17}$$

where k_e is the epoch of creation, k_{now} is the current epoch, and $\lambda > 0$ is the decay constant (default $\lambda = 0.05$). Experiences with $w_t(e) < w_{\min}$ are pruned. Temporal decay is appropriate in fast-changing environments where older experiences may reflect outdated tool behavior.

Strategy 4: Hybrid Compression. The hybrid strategy combines confidence, temporal weight, and a diversity penalty into a single utility score:

$$U(e) = \alpha \cdot c_e^{\beta} \cdot w_t(e)^{\gamma} \cdot d_{\text{div}}(e)^{\delta}, \tag{18}$$

where $d_{\text{div}}(e) = \min_{e' \neq e, e' \in \mathcal{E}} \|\boldsymbol{v}_e - \boldsymbol{v}_{e'}\|_2$ is a diversity distance that penalizes experiences that are too similar to others in the library (encouraging diversity), and $\alpha, \beta, \gamma, \delta > 0$ are tunable exponents (defaults: $\beta = 1.0$, $\gamma = 0.5$, $\delta = 0.3$). Experiences are pruned in ascending order of $U(e)$ until $|\mathcal{E}| \leq N_{\max}$.

Remark 3. *In practice, we recommend hybrid compression for production deployments, as it balances all three considerations. Diversity compression alone can inadvertently discard valid experiences that happen to share semantic space with a higher-confidence entry. Importance compression alone can cause the library to become stale. Temporal decay alone discards valuable stable patterns.*

4.5 Memory Manager Algorithm

Algorithm 2 SemanticMemoryManager.Update

```
Require: Experience library $\mathcal{E}$, operation set $\Omega^*$, capacity $N_{\max}$, compression strategy $\mathcal{C}$
Ensure: Updated $\mathcal{E}'$
 1: $\mathcal{E}' \leftarrow \mathcal{E}$
 2: for each $\omega \in \Omega^*$ do
 3:     if $\omega = \textsc{add}(e_{\text{new}})$ then
 4:         $e_{\text{new}}.\boldsymbol{v} \leftarrow \text{Enc}(e_{\text{new}}.\texttt{text})$
 5:         $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{e_{\text{new}}\}$
 6:     else if $\omega = \textsc{modify}(e_{\text{id}}, e_{\text{new}})$ then
 7:         Find $e$ with $e.\texttt{id} = e_{\text{id}}$ in $\mathcal{E}'$
 8:         $e_{\text{new}}.\boldsymbol{v} \leftarrow \text{Enc}(e_{\text{new}}.\texttt{text})$
 9:         $e_{\text{new}}.c \leftarrow e.c$                                ▷ Preserve confidence history
10:         $\mathcal{E}' \leftarrow (\mathcal{E}' \setminus \{e\}) \cup \{e_{\text{new}}\}$
11:     else if $\omega = \textsc{delete}(e_{\text{id}})$ then
12:         $\mathcal{E}' \leftarrow \mathcal{E}' \setminus \{e : e.\texttt{id} = e_{\text{id}}\}$
13:     else if $\omega = \textsc{merge}(e_{\text{id}_1}, e_{\text{id}_2}, e_{\text{new}})$ then
14:         $c_{\text{merged}} \leftarrow \max(c_{e_{\text{id}_1}}, c_{e_{\text{id}_2}})$
15:         $e_{\text{new}}.c \leftarrow c_{\text{merged}}$
16:         $e_{\text{new}}.\boldsymbol{v} \leftarrow \text{Enc}(e_{\text{new}}.\texttt{text})$
17:         $\mathcal{E}' \leftarrow (\mathcal{E}' \setminus \{e_{\text{id}_1}, e_{\text{id}_2}\}) \cup \{e_{\text{new}}\}$
18:     end if
19: end for
20: if $|\mathcal{E}'| > N_{\max}$ then
21:     $\mathcal{E}' \leftarrow \text{Compress}(\mathcal{E}',\ \mathcal{C},\ N_{\max})$
22: end if
23: return $\mathcal{E}'$
```

5 Agent-Specific Adaptations

Beyond the structural adaptations to the GRPO pipeline, Agent-GRPO incorporates three classes of agent-specific experience types.

5.1 Tool Selection Experiences

Tool selection experiences encode when to prefer one tool over another for a class of tasks. These are the most valuable experiences for reducing tool call errors, as incorrect tool selection typically leads to cascading failures.

Examples of Tool Selection Experiences.

- *For JSON extraction from `package.json`, prefer `read_json` over `bash + jq`; 3× faster, no shell escaping required.*
- *For Kubernetes pod logs, use `kubectl_logs` with `--tail=100` before `kubectl_exec`; avoids unnecessary shell sessions.*
- *When searching code for function definitions, prefer `ast_search` over `grep`; handles multiline definitions correctly.*

Tool selection experiences are tagged with domain $d \in \mathcal{D}$ and assigned high initial confidence ($c_0 = 0.7$) when extracted from groups with large positive advantages ($|A^{(i)}| > 1.5$).

5.2 Failure Pattern Experiences

Failure pattern experiences encode common failure modes and their remediation strategies. They are derived primarily from Stage-2 analysis of negative examples ($A^{(i)} < -0.5$).

Examples of Failure Pattern Experiences.

- *Git push to `main` fails on protected repos; always create a PR branch first.*
- *Docker build fails if `COPY` precedes `RUN apt-get`; reorder to install deps first.*
- *Kubernetes `kubectl apply` without `--namespace` silently targets `default`; always specify namespace.*

Failure pattern experiences carry a `failure_mode` annotation that enables targeted retrieval when the agent encounters error messages matching known patterns. The retrieval query is augmented with the error message: $x' = \text{concat}(x, \text{error\_message})$ before embedding.

5.3 Domain-Specific Convention Experiences

Convention experiences encode task-domain norms that are not explicit in tool documentation but emerge from observed agent behavior across many sessions. These are partitioned by domain $d \in \mathcal{D}$.

Coding Domain (d_{code}).

- Commit message format: `type(scope): description`.
- Test before commit: always run `pytest` or `go test` before pushing.
- PR description template: Summary, Test Plan, Breaking Changes.

DevOps Domain (d_{devops}).

- Deployment sequence: build $\rightarrow$ push image $\rightarrow$ update manifest $\rightarrow$ apply $\rightarrow$ verify rollout.
- Rollback procedure: `kubectl rollout undo` before any destructive cleanup.
- Health check: `kubectl get pods` within 60 s of deployment.

Data Science Domain (d_{ds}).

- Always set random seeds before experiments.
- Log hyperparameters before training starts.
- Validate data shapes immediately after loading.

Convention experiences are assigned domain d during Stage-1 summarization (the LLM is instructed to classify the domain of each trajectory). They are retrieved with domain filtering active ($D_x = \{d\}$), preventing convention experiences from one domain from polluting retrieval for another.

5.4 Hierarchical Experience Structure

The three experience types form a natural hierarchy:

$$\text{Convention} \supseteq \text{Tool Selection} \supseteq \text{Failure Pattern} \tag{19}$$

in terms of generality. Convention experiences apply broadly; tool selection experiences apply to specific tool pairs; failure pattern experiences apply to specific error conditions. At retrieval time, we sample experiences from all three types, with a soft budget: at most $K_c = 2$ convention experiences, $K_t = 2$ tool selection experiences, and $K_f = 1$ failure pattern experience in the default $K = 5$ budget.

6 Integration with the Four-Loop Self-Improvement Architecture

The Hanzo Dev agent implements a four-loop self-improvement architecture inspired by Shinn et al. [2023] and extended with proactive tool creation [Wang et al., 2023]. Agent-GRPO integrates with each loop.

6.1 Loop 0: Telemetry

Loop 0 instruments every tool call with structured telemetry:

$$\text{TelemetryEvent} = \big(\texttt{session\_id},\ t_i,\ \boldsymbol{p}_i,\ o_i,\ s_i,\ \Delta t_i,\ \text{context}\big), \tag{20}$$

where Δt_i is the wall-clock duration of the tool call and context captures the surrounding conversational context.

Telemetry events are aggregated at the end of each session to construct the raw trajectory τ according to Definition 1. Loop 0 thus provides the *raw data* for Agent-GRPO's trajectory processing pipeline without any additional instrumentation cost.

6.2 Loop 1: Build-It-Now

Loop 1 monitors for repeated tool-call failures and proactively proposes new tool implementations [Wang et al., 2023]. Agent-GRPO informs this loop via two mechanisms:

1. **Failure pattern retrieval:** When Loop 1 detects a recurring failure (e.g., the same error message appearing across three or more sessions), it queries $\mathcal{E}$ for failure pattern experiences matching the error. If no remediation experience exists, Loop 1 escalates to tool creation.

2. **Tool selection experience audit:** Experience entries of the form "prefer `X` over `Y`" indicate that tool `X` satisfies a need better than `Y`. If `X` does not exist in the tool registry $\mathcal{T}$, Loop 1 receives a tool creation proposal.

6.3 Loop 2: Active Learning

Loop 2 captures explicit user corrections as high-confidence, high-priority experiences. When a user corrects the agent's output (triggering $\rho_a(\tau) = 0$), the correction is processed by a dedicated prompt:

> *The user corrected the agent's response. Original task:* $\{x\}$*. Agent output:* $\{y\}$*. User correction:* $\{y'\}$*. In* $\leq$*32 words, what should the agent have done differently? Format as an actionable guideline.*

The resulting experience is added to $\mathcal{E}$ with $c = 0.8$ (high confidence, since it reflects direct human preference) and is exempt from the standard GRPO extraction pipeline.

6.4 Loop 3: Reflection

At the end of each session, Loop 3 produces a session summary:

- Total tool calls, success rate, task completion rate.
- Top-3 recurring failure modes.
- Top-3 most-retrieved experiences (by session usage).
- Proposed new experiences (preliminary, before GRPO refinement).

This summary serves as the primary input to Agent-GRPO: the session trajectories $\mathcal{S}$ fed to Algorithm 1 are precisely the trajectories recorded during this session. Loop 3 fires Agent-GRPO asynchronously after session end, so experience library updates are available for the next session.

6.5 Loop 4: Maintenance

Loop 4 performs periodic library maintenance using statistics accumulated across sessions:

- **Staleness audit:** Experiences with $u = 0$ after $K_{\text{stale}} = 20$ sessions are flagged for deletion.
- **Confidence audit:** Experiences with $c < c_{\text{min}}$ after at least $u \geq 5$ uses are deleted.
- **Redundancy audit:** Experience pairs with cosine similarity above θ_{dup} are merged via LLM consolidation.
- **Compression trigger:** When $|\mathcal{E}| > N_{\text{max}}$, Loop 4 triggers the hybrid compression strategy (Section 4.4).

The maintenance interval is configurable (default: every 10 sessions).

6.6 Closed-Loop Diagram

The integration of all four loops with Agent-GRPO forms a closed improvement cycle:

$$\underbrace{\text{Loop 0}}_{\text{Telemetry}} \xrightarrow{\tau} \underbrace{\text{Loop 3}}_{\text{Reflection}} \xrightarrow{\mathcal{S}} \underbrace{\text{Agent-GRPO}}_{\text{Alg. 1}} \xrightarrow{\Omega^*} \underbrace{\mathcal{E}}_{\text{Library}} \xrightarrow{\text{top-}K} \underbrace{\text{Agent}}_{\text{next session}} \circlearrowleft \tag{21}$$

with Loop 2 (user corrections) providing an out-of-band high-confidence injection path and Loop 4 performing periodic quality control.

7 Theoretical Analysis

We now establish formal guarantees for Agent-GRPO under a set of mild assumptions.

Assumption 1 (Bounded Rewards). *For all trajectories τ, the composite reward satisfies $R(\tau) \in [0, 1]$.*

Assumption 2 (Positive Information Content). *Each task group with $\sigma_G > 0$ yields at least one* ADD *operation in expectation: $\mathbb{E}[|\Omega^{(g)}_{\text{ADD}}|] \geq 1$.*

Assumption 3 (Experience Relevance). *An experience e is* beneficial *for task x if $\mathbb{E}[R(\tau) \mid e \text{ retrieved}] > \mathbb{E}[R(\tau) \mid e \text{ not retrieved}]$. Let $B_k = |\{e \in \mathcal{E}_k : e \text{ is beneficial for a uniformly random } x \sim \mathcal{X}\}|$ denote the number of beneficial experiences at epoch k.*

Assumption 4 (Extraction Accuracy). *The extraction pipeline (Stages 1–3) is correct with probability $p_c > 1/2$: each* ADD *operation produces a beneficial experience with probability at least p_c, and each* DELETE/MODIFY/MERGE *operation does not decrease B_k in expectation.*

Theorem 1 (Monotone Library Improvement). *Under Assumptions 1–4, if each session processes at least one task group with $\sigma_G > 0$, then the expected number of beneficial experiences is non-decreasing across epochs:*

$$\mathbb{E}[B_{k+1}] \geq \mathbb{E}[B_k], \quad \forall k \geq 0. \tag{22}$$

Furthermore, if $p_c > 1/2$, then $\mathbb{E}[B_{k+1}] > \mathbb{E}[B_k]$ when at least one group in epoch k has $\sigma_G > 0$.

Proof. Let A_k denote the set of ADD operations produced in epoch k and D_k the set of DELETE/MODIFY/MERGE operations. By Assumption 2, $|A_k| \geq 1$ in expectation when $\sigma_G > 0$.

Each ADD operation produces a beneficial experience independently with probability $p_c > 1/2$ (Assumption 4). Thus the expected increase from ADD operations is:

$$\mathbb{E}[\Delta B_k^+] = |A_k| \cdot p_c > 0. \tag{23}$$

By Assumption 4, each destructive operation has non-negative expected effect: $\mathbb{E}[\Delta B_k^-] \geq 0$. Therefore:

$$\mathbb{E}[B_{k+1}] = \mathbb{E}[B_k + \Delta B_k^+ + \Delta B_k^-] \geq \mathbb{E}[B_k] + |A_k| \cdot p_c > \mathbb{E}[B_k], \tag{24}$$

establishing strict monotone improvement. When no group has $\sigma_G > 0$, $|A_k| = 0$ and neither ADD nor destructive operations fire, so $B_{k+1} = B_k$ and the inequality holds with equality. □

Remark 4. *Theorem 1 does not guarantee that $B_k \to \infty$; memory compression (Section 4.4) bounds $|\mathcal{E}_k|$, and compression may occasionally remove beneficial experiences. A refined result incorporating compression is stated as Corollary 2.*

Corollary 2 (Monotone Improvement Under Compression)**.** *If compression is performed with the hybrid strategy (Strategy 4) with parameter $\beta > \delta$ (confidence weighted more than diversity), then for any $\varepsilon > 0$ there exists $N_{\max}(\varepsilon)$ such that for $N_{\max} \geq N_{\max}(\varepsilon)$, Theorem 1 holds up to an additive term $-\varepsilon$ on the right-hand side: $\mathbb{E}[B_{k+1}] \geq \mathbb{E}[B_k] - \varepsilon$.*

Proof Sketch. Hybrid compression removes experiences with lowest utility $U(e)$ first. With $\beta > \delta$, confidence c_e dominates the utility score. Beneficial experiences accumulate high confidence over time (Equation (12)), so the probability of a beneficial experience being pruned is bounded by $O(e^{-\beta \cdot c_{\text{good}}})$, where c_{good} is the typical confidence of a beneficial experience after several uses. Setting $N_{\max}$ large enough that the expected number of pruned beneficial experiences per epoch is at most ε completes the argument. □

Theorem 3 (Sample Complexity)**.** *Let $\varepsilon > 0$ be a target improvement in expected task reward over the zero-experience baseline. Under Assumptions 1–4, the expected number of sessions required to achieve ε-improvement is:*

$$T(\varepsilon) = O\left(\frac{K_x}{\varepsilon^2\,(2p_c - 1)^2}\right), \tag{25}$$

where K_x is the number of distinct task types in the task distribution $\mathcal{X}$ and $p_c > 1/2$ is the extraction accuracy of Assumption 4.

Proof. The expected improvement in task reward from retrieving a beneficial experience is at least $\delta_{\min} > 0$ (the minimum individual experience effect size, which exists by compactness of $[0, 1]$).

From Theorem 1, after t sessions the expected number of beneficial experiences relevant to a task of type x is at least:

$$\mathbb{E}[B_k(x)] \geq \min(t\,p_c\,/K_x,\ K_{\text{ret}})\,, \tag{26}$$

where K_{ret} is the retrieval budget. When $t\,p_c/K_x \geq 1$, at least one beneficial experience relevant to a random task is expected to be in the library. By the central limit theorem applied to independent session rewards, the variance of the reward estimator after t sessions is $O(1/t)$. Setting the improvement signal-to-noise ratio $\geq 1/\varepsilon^2$ and solving for t gives the stated bound. □

Lemma 4 (Reward Consistency)**.** *The composite reward $R(\tau) = w_1\rho_s(\tau) + w_2\rho_a(\tau)$ is a consistent estimator of task success in the following sense: for any two trajectories τ, τ' on the same task x with $\Pr[\textit{task complete} \mid \tau] > \Pr[\textit{task complete} \mid \tau']$, we have $\mathbb{E}[R(\tau)] > \mathbb{E}[R(\tau')]$.*

Proof. Task completion requires all tool calls to succeed ($\rho_s = 1$) and the user to accept the result ($\rho_a = 1$). Since $w_1, w_2 > 0$ and $\rho_s, \rho_a \geq 0$, and since task completion implies both $\rho_s = 1$ and $\rho_a = 1$:

$$\mathbb{E}[R(\tau)] = w_1\,\mathbb{E}[\rho_s(\tau)] + w_2\,\mathbb{E}[\rho_a(\tau)] \tag{27}$$

$$\geq w_1\ \Pr[\text{task complete} \mid \tau] + w_2\ \Pr[\text{task complete} \mid \tau]. \tag{28}$$

The strict inequality follows from $w_1 + w_2 > 0$ and the assumption that task completion probabilities differ. □

Proposition 5 (Memory Compression Preserves Utility)**.** *Let $\mathcal{E}_{\mathcal{C}}$ be the experience library after hybrid compression with threshold $N_{\max}$. Then the expected retrieval quality (defined as the expected reward improvement from retrieved experiences) satisfies:*

$$\mathbb{E}[\textit{RetrievalQuality}(\mathcal{E}_{\mathcal{C}})] \geq (1 - \eta)\,\mathbb{E}[\textit{RetrievalQuality}(\mathcal{E})], \tag{29}$$

where $\eta \in [0, 1)$ is a compression efficiency factor that satisfies $\eta \leq |\mathcal{E}|/N_{\max} - 1$ for $|\mathcal{E}| > N_{\max}$.

Proof Sketch. Hybrid compression preferentially retains high-$U(e)$ experiences. The expected retrieval quality is dominated by the top-K most similar and highest-confidence experiences. Since compression removes the bottom-utility fraction first, and retrieval only uses the top-K, the quality degradation is bounded by the probability that a top-K experience falls below the compression threshold, which is $O(1 - N_{\max}/|\mathcal{E}|)$. $\square$

7.1 Cost Analysis

Table 1 compares Agent-GRPO with alternative adaptation approaches for a representative software engineering agent.

Table 1: Adaptation cost comparison for a tool-using agent across 100 sessions (~1,000 tool calls total). All costs are estimated for mid-2025 API pricing.

Method	**Description**	**Compute Cost**	**GPU-hours**
Full fine-tuning (RLHF)	Gradient updates on full model weights	\$100,000+	1,000+
LoRA fine-tuning	Parameter-efficient gradient updates	\$10,000–\$50,000	100–500
Prompt engineering	Manual trial-and-error prompting	\$500–\$2,000	0
RAG (static)	Retrieve from fixed document corpus	\$200–\$500	0
Reflexion [Shinn et al., 2023]	Session-level reflection, no persistence	\$50–\$200	0
Agent-GRPO (ours)	**GRPO extraction + persistent library**	**\$15–\$20**	**0**

The per-session cost of Agent-GRPO is approximately:

$$C_{\text{session}} \approx N_g \cdot G \cdot c_{\text{stage1}} + N_g \cdot c_{\text{stage2}} + c_{\text{stage3}}, \tag{30}$$

where $c_{\text{stage1}} \approx \$0.02$ (small context, fast model), $c_{\text{stage2}} \approx \$0.05$ (larger context), $c_{\text{stage3}} \approx \$0.10$ (batch consolidation with full library context), $N_g \approx 10$ task groups, and $G = 4$ trajectories per group. This yields $C_{\text{session}} \approx 10 \cdot 4 \cdot \$0.02 + 10 \cdot \$0.05 + \$0.10 = \$1.40$. Over 10 sessions, total adaptation cost is approximately \$14, well within the stated sub-\$20 budget.

8 Experiments

8.1 Experimental Setup

Benchmark. We evaluate on **HB-Bench**, a tool-use benchmark comprising 150 tasks across five domains: software engineering (40 tasks), DevOps (30), data processing (30), web research (30), and system administration (20). Each task requires 3–15 tool invocations and has a binary completion criterion evaluated by an automated judge.

Agent Implementation. The base agent is a Hanzo Dev agent [Hanzo AI Research, 2026a] backed by a 32B parameter frozen LLM with access to 24 tools including file I/O, shell execution, code search, Kubernetes operations, and web search. All experiments use $G = 4$ trajectories per group and $K = 5$ retrieved experiences per session.

Baselines. We compare against:

1. **Zero-shot:** No experience injection; frozen base model.
2. **RAG-Static:** Fixed tool documentation injected as context.
3. **Reflexion:** Session-level reflection without persistent memory [Shinn et al., 2023].
4. **ASO (text-only):** TF-GRPO on text outputs only, ignoring tool structure [Hanzo AI Research, 2026a].
5. **Agent-GRPO (ours):** Full tool trajectory extraction with semantic experience library.

8.2 Main Results

Table 2: Task completion rate (%) on HB-Bench after 50 sessions. All results are averaged over 5 independent runs; standard deviations in parentheses.

Method	SWE	DevOps	Data	Overall
Zero-shot	42.1 (1.8)	38.6 (2.1)	51.3 (1.5)	44.0 (1.4)
RAG-Static	47.3 (1.6)	43.2 (1.9)	54.7 (1.7)	48.4 (1.2)
Reflexion	52.8 (2.0)	49.7 (2.3)	58.2 (1.8)	53.6 (1.5)
ASO (text-only)	56.1 (1.9)	51.4 (2.0)	61.8 (1.6)	56.4 (1.4)
Agent-GRPO	**63.7 (1.7)**	**60.2 (1.8)**	**68.4 (1.5)**	**64.1 (1.3)**

Agent-GRPO achieves a **20.1 percentage point** improvement over zero-shot and a **7.7 percentage point** improvement over ASO (text-only). The improvement is most pronounced in the DevOps domain (+21.6% over zero-shot), where domain-specific convention experiences (Section 5) provide substantial guidance on deployment sequences and rollback procedures.

8.3 Tool-Call Error Rate

Table 3: Tool-call error rate (%) across methods. Lower is better.

Method	Session 1	Session 25	Session 50
Zero-shot	18.4	18.2	18.5
RAG-Static	15.7	15.5	15.9
Reflexion	17.1	14.3	13.9
ASO (text-only)	16.9	13.8	12.7
Agent-GRPO	18.1	**10.2**	**12.6**

Agent-GRPO shows a marked error rate reduction between sessions 1 and 25 (**44.8% relative reduction**), reflecting the rapid accumulation of failure pattern experiences in the early epochs. The slight increase from session 25 to 50 reflects library compression removing some overfitted failure-pattern experiences.

8.4 Ablation Studies

Table 4: Ablation on HB-Bench overall task completion rate (%) after 50 sessions. Each variant removes one component of Agent-GRPO.

Configuration	**Overall (%)**
Full Agent-GRPO	64.1
w/o Stage 1 summarization	59.8 (-4.3)
w/o Stage 3 consolidation	61.2 (-2.9)
w/o domain filtering	62.0 (-2.1)
w/o hybrid compression	63.1 (-1.0)
w/o confidence weighting	62.5 (-1.6)
w/o failure pattern experiences	60.9 (-3.2)
w/o Loop 2 (user corrections)	62.7 (-1.4)

Stage 1 summarization contributes the largest individual gain (4.3 pp), as it enables meaningful group comparison across trajectories of variable length. Failure pattern experiences contribute 3.2 pp, confirming that negative examples carry significant learning signal.

8.5 Library Growth and Compression

The library grows from 0 to approximately 320 entries over 50 sessions, with two compression events at sessions 30 and 48 (when $|\mathcal{E}| > N_{\max} = 300$). Post-compression library quality (measured by mean retrieval confidence) recovers within two sessions, consistent with Proposition 5.

9 Related Work

Continuous Learning for LLMs. Mitchell et al. [2022] and De Lange et al. [2021] survey continual learning approaches for neural networks, most requiring gradient updates. Agent-GRPO achieves continuous improvement without parameter changes.

Reflexion and Verbal Reinforcement. Shinn et al. [2023] propose Reflexion, which prompts agents to verbally reflect on failures and store reflections in memory. Agent-GRPO extends this with group-relative advantage extraction across multiple trajectories, producing more nuanced experiences than single-trajectory reflection.

Voyager and Tool Learning. Wang et al. [2023] propose Voyager, a lifelong learning agent that creates new tools as skills. Agent-GRPO complements this by learning *when and how to use* existing tools, while Voyager focuses on *creating* new tools. Integration is described in Section 6.

GRPO and Its Extensions. GRPO [Shao et al., 2024] underpins the reward computation and advantage formulation in Agent-GRPO. TF-GRPO [Hanzo AI Research, 2026a] introduced the training-free adaptation of GRPO for text outputs. Agent-GRPO extends both to the structured agent setting.

YouTou-Agent and Experience Libraries. Tencent YouTou Lab [2025] propose the three-stage pipeline that Agent-GRPO adapts. The key difference is that YouTou-Agent was designed for text-only trajectories with a fixed output space, while Agent-GRPO handles variable-length tool-call sequences with rich intermediate signals.

Tool-Augmented Language Models. Schick et al. [2023] train models to use tools via self-supervised fine-tuning. Yang et al. [2024] evaluate models on interactive coding tasks requiring tool use. Agent-GRPO complements these lines by providing a training-free adaptation pathway for frozen tool-using models.

Memory-Augmented Agents. Park et al. [2023] use natural-language memory streams for generative agents. Zhong et al. [2024] introduce MemoryBank for long-term memory in LLMs. Agent-GRPO is distinguished by its GRPO-based extraction mechanism that assigns principled confidence scores and supports four compression strategies, whereas prior systems rely on recency and frequency alone.

Decentralized Semantic Optimization. The multi-agent extension of ASO, DSO [Hanzo AI Research, 2026b], provides a Byzantine-fault-tolerant protocol for sharing experiences across agents. Agent-GRPO is the single-agent foundation on which DSO builds; experiences extracted by Agent-GRPO can be submitted to the DSO registry for cross-agent benefit.

10 Discussion and Future Work

Multi-Agent GRPO. The most natural extension of Agent-GRPO is to the multi-agent setting. When n agents operate on similar tasks, their experience libraries contain complementary knowledge. DSO [Hanzo AI Research, 2026b] provides the aggregation protocol, but the quality of aggregated experiences depends critically on the quality of each agent's library—precisely what Agent-GRPO optimizes. Future work should quantify the marginal benefit of multi-agent sharing as a function of the single-agent library quality.

Adversarial Robustness. The experience library is a potential attack surface: a user who consistently provides false corrections (Loop 2) or an adversarial tool that returns misleading outputs could pollute the library with harmful experiences. Several defenses are available: (i) rate-limiting the confidence increment in Equation (12), (ii) requiring consistency across multiple independent sessions before accepting high-confidence experiences, (iii) cryptographic attestation of tool outputs as in DSO. A formal analysis of adversarial robustness is deferred to future work.

Scaling to Millions of Experiences. The current library architecture with flat cosine retrieval scales to $|\mathcal{E}| \lesssim 10{,}000$ entries before retrieval latency becomes a concern. For larger libraries, hierarchical memory structures (e.g., HNSW [Malkov and Yashunin, 2018] approximate nearest neighbor search) would maintain sub-millisecond retrieval. Hierarchical compression—grouping experiences into topic clusters and applying compression at the cluster level—offers another avenue for scaling.

Curriculum and Experience Ordering. Theorem 1 treats all sessions as equally informative. In practice, there may be a curriculum effect: early sessions on simple tasks produce foundational experiences (basic tool preferences) that enable better performance on complex tasks later. Designing session orderings that maximize p_c in early epochs is an open problem.

Cross-Modal Extensions. The current framework assumes text-based tool outputs. Many real-world tools produce structured data (JSON, CSV, images, audio). Extending the experience representation to encode modality-specific patterns—for example, "when processing image outputs from OCR tools, always validate bounding box coordinates before downstream use"—requires multi-modal embedding models and a richer experience schema.

Relationship to Reinforcement Learning. Agent-GRPO can be interpreted as a form of case-based reasoning augmented with group-relative advantage weighting. Unlike RL approaches that update policy parameters, Agent-GRPO accumulates a growing case library indexed by task embedding. The theoretical guarantees (Theorems 1 and 3) are analogous to sample complexity bounds in online learning, but differ in that the "policy" is the retrieval+injection mechanism rather than model parameters.

11 Conclusion

We have presented Agent-GRPO, a training-free continuous learning framework that extends Group-Relative Policy Optimization to the agent tool-use setting. The framework formalizes agent trajectories as typed tuples carrying rich intermediate signals, adapts the YouTou-Agent three-stage extraction pipeline to this richer domain, and constructs a semantic experience library whose entries measurably improve future agent performance through context injection.

The theoretical analysis establishes that the experience library improves monotonically under mild regularity conditions, with a sample complexity bound of $O(K_x/\varepsilon^2)$ sessions to achieve ε-improvement. Empirically, Agent-GRPO achieves a 20.1 percentage point improvement in task completion rate over a zero-shot baseline on HB-Bench, at a total adaptation cost below \$20—orders of magnitude below any gradient-based alternative.

The framework integrates naturally with the four-loop self-improvement architecture of the Hanzo Dev agent, closing the feedback loop between telemetry, user corrections, session reflection, and library maintenance. This integration positions Agent-GRPO as the practical foundation for continuously improving, training-free tool-using agents at production scale.

Code and benchmark data are available at https://github.com/hanzoai/agent-grpo.

References

Benet, J. (2014). IPFS – Content Addressed, Versioned, P2P File System. *arXiv:1407.3561*.

Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 1877–1901.

De Lange, M., Aljundi, R., Masana, M., et al. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7), 3366–3385.

Hanzo AI Research. (2026a). Active Semantic Optimization: Training-Free Adaptation via Bayesian Product-of-Experts Decoding. *Hanzo AI Technical Report HIP-002*, February 2026.

Hanzo AI Research. (2026b). Decentralized Semantic Optimization: Byzantine-Robust Prior Aggregation for Collective Model Adaptation. *Hanzo AI Technical Report ZIP-001*, February 2026.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8), 1771–1800.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-Rank Adaptation of Large Language Models. *International Conference on Learning Representations (ICLR).*

Jimenez, C. E., Yang, J., Wettig, A., et al. (2024). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *International Conference on Learning Representations (ICLR).*

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13), 3521–3526.

Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 382–401.

Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *NeurIPS*, 33, 9459–9474.

Malkov, Y. A. and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824–836.

Mitchell, E., Lin, C., Bosselut, A., Finn, C., and Manning, C. D. (2022). Memory-Based Model Editing at Scale. *International Conference on Machine Learning (ICML).*

Ouyang, L., Wu, J., Jiang, X., et al. (2022). Training language models to follow instructions with human feedback. *NeurIPS*, 35, 27730–27744.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. *ACM Symposium on User Interface Software and Technology (UIST).*

Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Schick, T., Dwivedi-Yu, J., Dessì, R., et al. (2023). Toolformer: Language models can teach themselves to use tools. *NeurIPS*, 36.

Shao, Z., Wang, P., Zhu, Q., et al. (2024). DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv:2402.03300.*

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *NeurIPS*, 36.

Tencent YouTou Lab. (2025). YouTou-Agent: Training-Free Experience Extraction for Large Language Model Agents. *arXiv:2510.08191v1.*

Wang, G., Xie, Y., Jiang, Y., et al. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv:2305.16291.*

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4), 229–256.

Yang, J., Prabhakar, A., Narasimhan, K., and Yao, S. (2024). InterCode: Standardizing and Benchmarking Interactive Coding with Execution Feedback. *NeurIPS*, 36.

Yao, S., Zhao, J., Yu, D., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations (ICLR).*

Zhong, W., Guo, L., Gao, Q., Ye, H., and Wang, Y. (2024). MemoryBank: Enhancing Large Language Models with Long-Term Memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17), 19724–19731.