ABSTRACT-STATE GROUPING POLICY OPTIMIZATION FOR CLI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent methods for step-level credit assignment in agentic reinforcement learning, such as Group-in-Group Policy Optimization (GiGPO), compare actions taken from common states across trajectories. However, states of command-line interface (CLI) agents are represented by terminal transcripts that vary across runs. Run-specific paths, hashes, and logs make these transcripts differ even when they describe similar execution situations. Because common-state comparisons require identical transcripts, these differences make common states rare across trajectories. In this paper, we observe that up to 98% of the groups formed by common states contain only one sampled step, and such groups yield zero step-level advantage. To address this problem, we propose Abstract-State Grouping Policy Optimization (ASGPO), a critic-free algorithm that estimates step-level advantages by grouping steps according to *abstract states*, characterized by recent execution feedback and interaction progress. Specifically, ASGPO maps each pre-action transcript to an abstract state, and then groups steps sharing this abstract state and computes their relative advantages using bounded progress-aware returns. These returns combine the final task reward with execution-feedback scores accumulated from each step onward. Positive and negative progress contributions are capped separately to control their scale. Comprehensive experimental results demonstrate the effectiveness of our proposed ASGPO.

1 INTRODUCTION

Large language models (LLMs) generate text and code (OpenAI et al., 2023; Qwen et al., 2024; DeepSeek-AI et al., 2024). LLM agents use these capabilities to interact with tools and environments (Yao et al., 2023; Schick et al., 2023). In command-line interfaces (CLIs), they inspect files and run tests to guide code edits. These interactions form a *trajectory* of actions and observations (Yang et al., 2023; Merrill et al., 2026). At each step, the available interaction transcript serves as the agent's *concrete (or raw) state*. Task verifiers typically provide a reward only for the final outcome of a trajectory. Training CLI agents with reinforcement learning (RL) therefore requires assigning credit to individual actions using this outcome-level feedback (Sutton & Barto, 2018).

REINFORCE Leave-One-Out (RLOO) and Group Relative Policy Optimization (GRPO) compare sampled trajectory rewards without training a value network (Kool et al., 2019; Ahmadian et al., 2024; Shao et al., 2024). Standard trajectory-level GRPO assigns the same relative score, or *episode-level advantage*, to every action in a rollout. This score does not distinguish individual action contributions. GiGPO compares actions from common states across trajectories for finer-grained credit (Feng et al., 2025). Its default rule groups steps with identical observed states. We call this rule *exact-state grouping* and the resulting sets *exact-state groups*. Each step's return summarizes rewards from that step onward. GiGPO uses the group mean as a baseline and centers each return to obtain a step-level advantage for the corresponding action.

In CLI environments, different transcripts can describe similar execution situations, such as tracebacks during early debugging. However, run-specific paths, hashes, line numbers and logs make these transcripts differ. Exact-state grouping requires identical observed states, so these text differences prevent otherwise comparable steps from sharing a group. This leaves few samples per group. A group with only one sampled step is a *singleton*. Its return equals the group mean, so its centered step advantage is zero. Our preliminary experiments show that singleton-group rates are 73% for

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

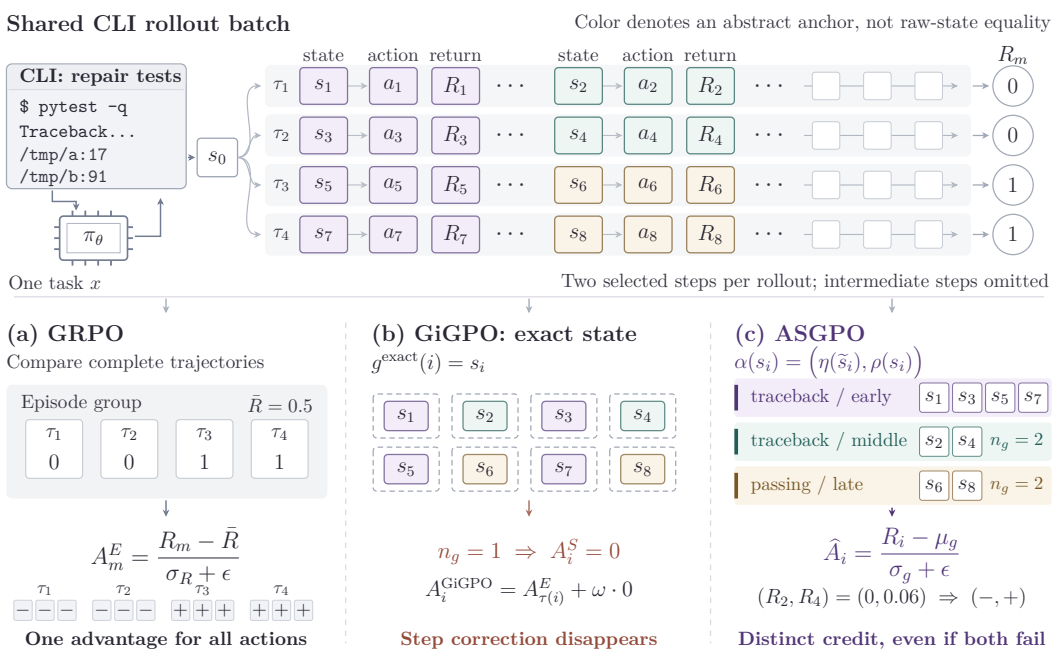

Figure 1: Comparison of credit-assignment schemes. (a) GRPO assigns one episode advantage to each rollout. (b) GiGPO's step correction vanishes for singleton groups, but its episode term remains. (c) ASGPO compares steps sharing feedback–progress anchors, indicated by color. Two steps are shown per trajectory with illustrative returns.

Qwen3-4B and 92% for Qwen2.5-7B on the OpenThoughts split, and 98% for Nemotron-Terminal-8B on the SETA split. These results motivate grouping steps by shared execution feedback and interaction stage rather than requiring their full transcripts to match.

In this paper, we propose Abstract-State Grouping Policy Optimization (ASGPO), which groups steps by two pre-action transcript features. The *feedback mode* distinguishes execution conditions such as tracebacks, missing resources, and passing checks. To separate steps with the same feedback at different stages, the *progress stage* labels the number of completed interactions as start, early, middle or late. The pair of features defines an *abstract state*, also called an *anchor*. Steps with this pair form an *abstract group*, even if their concrete transcripts differ. For transcript s, let $\tilde{s}$ denote its recent segment after removing run-specific text. The map $\alpha(s) = (\eta(\tilde{s}), \rho(s))$ assigns feedback mode η and progress stage ρ. Abstract states select comparison peers, while the policy still receives the full transcript. Abstract groups provide peers, but their returns can still coincide. Failed rollouts can have the same final verifier reward despite different intermediate command outcomes. If all steps in a group share this verifier-only return, their centered advantages are zero. ASGPO therefore constructs *progress-aware returns* by combining the final verifier reward with execution-feedback scores accumulated from each step onward. Separate caps control the scale of positive and negative contributions. Groupwise centering and scaling then yield step-level advantages.

Figure 1 illustrates these comparisons. Bars and μ_g denote return means, and σ denotes standard deviation. In panel (b), singletons remove GiGPO's step term weighted by ω, but its episode advantage remains. In panel (c), ASGPO forms groups of sizes $n_g = 4, 2, 2$. The middle-progress traceback group contains returns $R_2 = 0$ and $R_4 = 0.06$. Centering them around their group mean gives opposite advantage signs, distinguishing the two steps despite both rollouts failing. These comparisons depend on the state information retained by the grouping rule. Finer grouping can separate states with different expected returns but leave fewer samples for estimating group means. Our theorem relates group size and within-group value differences to baseline estimation error. It covers a family of comparison rules with exact-state grouping as the finest partition. Under matched training settings, ASGPO improves mean held-out success over exact-state grouping across 4B–8B agents. The final 8B model reaches 0.127 on Terminal-Bench 2.0, compared with 0.092 for its released Nemotron base and 0.089 for matched exact-state grouping.

This paper makes three contributions:

- We introduce **ASGPO**, which groups terminal transcripts by feedback mode and progress stage and constructs a bounded progress-aware return for step-level optimization.
- We quantify the need for **abstract comparison states** in CLI agents, linking rare repeated states to singleton groups and zero centered step advantages.
- We derive an **abstraction-refinement theorem** that formalizes the trade-off between comparison-group size, within-group value differences, and baseline estimation error.

2 Related Work

Learning from agent interaction. Benchmarks evaluate web and application control (Yao et al., 2022; Trivedi et al., 2024; Xie et al., 2024), as well as software and terminal tasks (Jimenez et al., 2024; Yang et al., 2023; Merrill et al., 2026). Training agents for these settings requires both task experience and policy learning. Software-training work scales data and environments (Wei et al., 2025; Gandhi et al., 2026). Policy-learning approaches include ArCHer's hierarchical RL and AgentQ's search-based preferences (Zhou et al., 2024; Putta et al., 2024). LOOP and RAGEN adapt reward-based policy updates to multi-turn rollouts (Chen et al., 2025; Wang et al., 2025b).

Group-relative credit assignment. Advantage estimation guides policy updates for preference alignment (Ziegler et al., 2019; Ouyang et al., 2022) and reasoning (DeepSeek-AI et al., 2025). RLOO and GRPO use sample-based baselines instead of a learned critic (Kool et al., 2019; Ahmadian et al., 2024; Shao et al., 2024). GRPO variants address objective bias and efficiency (Liu et al., 2025; Yu et al., 2025; Lin et al., 2025). GiGPO combines episode and within-state step advantages (Feng et al., 2025). Its step baseline depends on the comparison peers.

Comparison-peer selection. Comparison rules use different state information. HGPO refines identical-observation groups by shared history (He et al., 2026). GiGPO's similarity-based variant and ProxMO's TF–IDF weighting relax exact text matching (Feng et al., 2025; Fang et al., 2026). Bi-PACE uses actor-hidden-state clusters and action-conditioned estimates (Wang et al., 2026a). RTMC builds rollout trees from software state–action signatures (Wang et al., 2026b). ASGPO instead groups by feedback mode and interaction depth. Both coordinates come from the pre-action transcript without full-transcript matching or hidden-state extraction.

State abstraction and process feedback. State abstraction groups concrete states by value-relevant properties (Li et al., 2006; Ferns et al., 2004). Grouping selects peers but does not specify their training rewards. Process supervision evaluates reasoning steps (Uesato et al., 2022; Lightman et al., 2024), while tool-use rewards evaluate intermediate interactions (Qian et al., 2025; Wang et al., 2025a). ASGPO combines pre-action grouping with bounded progress-aware returns. The resulting group-relative advantages enter a clipped policy objective (Schulman et al., 2017).

3 Problem Formulation

For task x, the rollout policy $\pi_{\theta_{\text{old}}}$ samples M trajectories $\tau_1, \ldots, \tau_M$. At step t of τ_m, it samples action $a_{m,t}$ given the full pre-action transcript $s_{m,t}$. Execution feedback enters the next transcript. For step-level comparison, we index the N sampled steps by i, with $(s_i, a_i) = (s_{m,t}, a_{m,t})$ for the corresponding (m, t). Let R_i denote the training return used to evaluate action a_i.

Group-relative step credit. A grouping rule assigns each step a key $g(i)$ to select its comparison peers. The group G_g contains all step indices with key g and has size $n_g = |G_g|$. Let μ_g and σ_g denote its return mean and standard deviation. We compute the step-level advantage by centering R_i on its group mean and scaling the difference (Shao et al., 2024; Feng et al., 2025):

$$A_i^S(g) = \frac{R_i - \text{mean}_{j \in G_{g(i)}} R_j}{F_{\text{norm}}(\{R_j : j \in G_{g(i)}\})}, \tag{1}$$

where F_{norm} is a positive groupwise scale. Setting $F_{\text{norm}} = 1$ omits scaling. We use $\sigma_{g(i)} + \epsilon$ with $\epsilon > 0$ to avoid division by zero when group returns are equal.

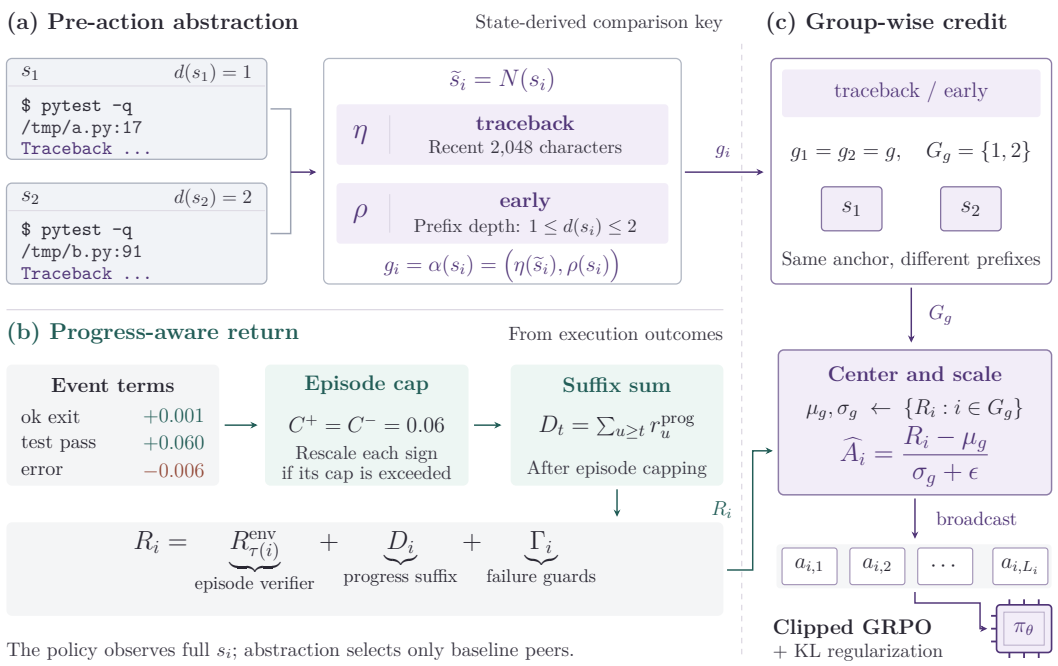

Figure 2: ASGPO training mechanism. (a) Pre-action feedback and interaction depth define abstract anchors. (b) Capped execution signals form progress-aware returns. (c) Group-relative advantages drive the policy update. Abstraction selects comparison peers, while the policy observes the full prefix.

4 METHOD: ASGPO

ASGPO constructs comparison peers and progress-aware returns from the same rollout batch. Figure 2 shows the three stages: extract pre-action anchors, construct returns from execution outcomes, then compute group-relative advantages for policy optimization.

4.1 Constructing Abstract Comparison States

We compare steps with the same feedback mode and interaction stage, even when their transcripts differ. Let $\widetilde{s}_i$ denote the normalized recent transcript. The pair $(\eta(\widetilde{s}_i), \rho(s_i))$ defines the abstract anchor $g_i = g(i)$ for step i. Its comparison group G_{g_i} contains all sampled step indices j with that anchor:

$$g_i = \alpha(s_i) = \big(\eta(\widetilde{s}_i), \rho(s_i)\big), \qquad G_{g_i} = \{j : \alpha(s_j) = g_i\}. \tag{2}$$

Feedback mode η. Feedback modes separate conditions that require different next actions. After removing run-specific text, η checks the last 2,048 characters of the normalized prefix. It selects the first matching mode in the order `traceback`, `permission`, `missing`, `error`, and `passing`. These rules identify exception traces, denied access, missing resources, generic failures, and passing checks, respectively. If no rule matches, the mode is `normal`. Priority resolves overlapping signals: a traceback overrides a generic error.

Progress stage ρ. The same feedback can occur at different interaction depths. Let $d(s_i)$ count completed agent–environment interactions in the prefix. We assign four progress buckets:

$$\rho(s_i) = \begin{cases} \texttt{start}, & d(s_i) = 0, \\ \texttt{early}, & 1 \le d(s_i) \le 2, \\ \texttt{middle}, & 3 \le d(s_i) \le 6, \\ \texttt{late}, & d(s_i) \ge 7 \end{cases}. \tag{3}$$

The implementation reads the equivalent stored step index. Thus, ρ measures interaction depth rather than verified task completion. Both coordinates use only the pre-action prefix. The current action and future verifier outcomes are excluded from the key.

In Figure 2(a), two prefixes differ in temporary paths and traceback line numbers. Both report an early-stage traceback and map to the same anchor:

$$\alpha(s_i) = \alpha(s_j) = (\texttt{traceback}, \texttt{early}), \quad s_i \neq s_j. \tag{4}$$

Appendix B specifies canonicalization, priority rules, and pseudocode.

4.2 Constructing Progress-Aware Returns

Abstract groups supply comparison peers, but equal returns still give zero centered advantages. Terminal tasks typically provide sparse final verifier rewards despite useful execution feedback appearing earlier (Yang et al., 2023; Jimenez et al., 2024; Merrill et al., 2026; Gandhi et al., 2026). A zero final reward gives every step a zero undiscounted verifier return. ASGPO therefore includes execution scores in its training returns to distinguish outcomes with equal verifier rewards.

Execution scores. For one rollout, let r_t^{prog} denote the signed execution score at step t. Successful commands and passing local tests contribute positive terms. Errors and repeated commands contribute penalties. Timeouts and detected benchmark-specific shortcuts incur additional penalties. Scores use command strings and visible tool outputs, not hidden verifier labels. This design draws on process supervision and tool-use rewards (Uesato et al., 2022; Lightman et al., 2024; Qian et al., 2025; Wang et al., 2025a). Appendix B defines the events and weights in Eq. (16).

Bounded accumulation. Long rollouts can accumulate large positive or negative progress totals. We therefore cap each sign's total magnitude separately. If either total exceeds its cap, all scores of that sign are rescaled uniformly. The capped scores, still denoted r_t^{prog}, satisfy

$$\sum_{t:r_t^{\text{prog}}>0} r_t^{\text{prog}} \leq C^+, \qquad -\sum_{t:r_t^{\text{prog}}<0} r_t^{\text{prog}} \leq C^-. \tag{5}$$

Here $C^+ = C^- = 0.060$ limits each sign's contribution relative to a full task-completion reward.

Return assembly. To credit earlier actions for later execution outcomes, we form a suffix sum of the capped scores (Figure 2(b)). Let D_t denote this sum from step t onward and G_t the sparse environment return. For the sample i at that step,

$$D_t = \sum_{u \geq t} r_u^{\text{prog}}, \qquad R_i = G_t + D_t + \Gamma_t, \tag{6}$$

where G_t equals the rollout's final verifier score under undiscounted returns. It is a scalar return, distinct from the comparison group G_g. The term Γ_t contains penalties for short failed episodes and failed episodes whose prompts exceed the context budget. The short-episode penalty discourages terminating early without success. Successful short solutions incur no such penalty. The coordinate ρ selects comparison stages, while D_t contributes subsequent execution scores to the return. Progress scores guide training, while evaluation uses the environment verifier.

4.3 Group-Relative Advantages And Policy Update

ASGPO compares the assembled returns within groups defined by pre-action anchors. For each anchor g, we compute μ_g and σ_g from $\{R_j : j \in G_g\}$. Applying Eq. (1) with $g(i) = \alpha(s_i)$ gives $\widehat{A}_i = (R_i - \mu_{g(i)})/(\sigma_{g(i)} + \epsilon)$.

Illustrative comparison. Consider a group containing only the two early traceback steps in Eq. (4). Suppose both rollouts fail with zero verifier rewards and no penalties Γ_t. Their capped progress suffixes are 0 and 0.06. Their verifier returns are equal, but their training returns are 0 and 0.06. The group mean is 0.03, giving centered returns of -0.03 and $+0.03$. Division by the same positive scale preserves these opposite signs. The step with greater subsequent progress therefore receives higher credit despite the shared failure outcome.

Policy optimization. We broadcast each step advantage to all tokens of its sampled action (Figure 2(c)) and optimize the clipped GRPO objective with Kullback–Leibler (KL) regularization. The policy observes the full transcript, not the abstract key. Appendix B gives the objective and complete pseudocode.

The matched EXACT baseline replaces only the key with $g^{\text{exact}}(i) = s_i$. It shares the returns, normalization, and optimizer. Both estimators include the final verifier score in R_i before normalization. Neither adds a separately normalized episode advantage. EXACT tests GiGPO's step grouping rule rather than its full episode-plus-step estimator, which appears in Appendix B.1.

Complexity. Extracting anchors requires one pass over each transcript, $O(\sum_i |s_i|)$ time. Hash-based grouping and return statistics take $O(N)$ time and $O(N)$ worst-case storage for N sampled steps. No learned encoder or critic is required. Grouping adds no clustering pass or extra model call.

5 ANALYSIS: ABSTRACT COMPARISON STATES

How much execution detail should an abstract state retain? Let Φ contain deterministic pre-action maps $\phi : \mathcal{S} \to \mathcal{Z}_\phi$, where $\phi(s)$ is an *abstract comparison state* used only to select baseline peers. Here $\mathcal{S}$ and $\mathcal{Z}_\phi$ denote the concrete and abstract state spaces. Each map induces the group-relative estimator

$$\widehat{A}_i(\phi) = \frac{R_i - \mu_{\phi(s_i)}}{\sigma_{\phi(s_i)} + \epsilon}. \tag{7}$$

ASGPO uses $\phi = \alpha = (\eta, \rho)$. The identity map $\phi_{\text{id}}(s) = s$ recovers the matched exact-state estimator associated with GiGPO's step component. These maps differ in the state information they retain for comparison. Identity retains all concrete-state distinctions.

To examine how group size affects centered credit, let $u_i = \nabla_\theta \log \pi_\theta(a_i \mid s_i)$ and temporarily omit the positive scale denominator. Let $\mathcal{F}$ fix an action-independent partition in the independent-branch model of Appendix C. *Leave-in centering* includes the target step's own return in the group mean. It gives

$$\mathbb{E}\big[u_i(R_i - \mu_{g(i)}) \mid \mathcal{F}\big] = \left(1 - \frac{1}{n_{g(i)}}\right) \mathbb{E}[u_i R_i \mid \mathcal{F}]. \tag{8}$$

Singletons therefore contribute no centered step-level policy-gradient signal. For N steps in $|\mathcal{G}|$ nonempty groups, the average retention factor is

$$\frac{1}{N} \sum_{i=1}^{N} \left(1 - \frac{1}{n_{g(i)}}\right) = 1 - \frac{|\mathcal{G}|}{N}. \tag{9}$$

More nonempty groups therefore lower the average leave-in retention factor.

Coarser grouping increases this factor but can mix different continuation values. To analyze this trade-off, say that ϕ_2 *refines* ϕ_1 when $\phi_1 = h \circ \phi_2$ for some deterministic h. Define the realized leave-in signal retention $\mathcal{S}_N(\phi) = 1 - |\mathcal{G}_\phi|/N$ and the population value aliasing $\mathcal{H}(\phi) = \mathbb{E}[\mathrm{Var}(V(S) \mid \phi(S))]$, where $V(s) = \mathbb{E}[R \mid S = s]$. Value aliasing measures variation in continuation value among states assigned the same abstract label. We call the set of states sharing a label z a *cell*. Its sampled steps form the corresponding group.

Theorem 1 (Abstract comparison-state trade-off) *Assume square-integrable returns and let ϕ_2 refine ϕ_1. Then the first inequality below holds for every realized batch, while the second is a population statement:*

$$\mathcal{S}_N(\phi_2) \le \mathcal{S}_N(\phi_1), \qquad \mathcal{H}(\phi_2) \le \mathcal{H}(\phi_1). \tag{10}$$

Write $R = V(S) + \xi$ with $\mathbb{E}[\xi \mid S] = 0$. Inside a cell $z = \phi(S)$, let $H_z = \mathrm{Var}(V(S) \mid z)$ and $\bar{\sigma}_z^2 = \mathbb{E}[\mathrm{Var}(\xi \mid S) \mid z]$. If $\widehat{V}_z^{(m)}$ is the mean return of m conditionally independent peers from that cell and S_0 is an independent target state in the same cell, then

$$\mathbb{E}[(\widehat{V}_z^{(m)} - V(S_0))^2 \mid z] = H_z + \frac{H_z + \bar{\sigma}_z^2}{m}. \tag{11}$$

The theorem separates the two effects of comparison granularity. Finer maps reduce value aliasing but can lower the leave-in retention factor. Coarser maps provide more peers per group. They improve baseline mean squared error (MSE) only when the reduction in sampling error outweighs the added heterogeneity in Eq. (11). Identity has zero value aliasing but the smallest retention factor. This trade-off motivates ASGPO's two-coordinate design: retain feedback and stage distinctions while pooling other transcript variation. Appendix C provides the proofs, assumptions on sample dependence, and comparison with concrete-state identity.

6 EXPERIMENTS

6.1 Setup

All RL runs start from existing supervised fine-tuning (SFT) checkpoints, without additional large-scale SFT. Within each scale, ASGPO and EXACT share the initialization and progress signal. They use the same optimizer, schedule, and rollout budget. Decoding and checkpoint windows are also matched. The 4B/7B agents use OpenThoughts Terminal-Bench-format tasks (655 training, 73 held-out). The 8B agent uses *Nemotron-Terminal-8B* (Pi et al., 2026) and SETA tasks (156 training, 70 held-out). Official Terminal-Bench 2.0 (Merrill et al., 2026) is reserved for post-training zero-shot evaluation.

De-noised plateau evaluation. For the primary controlled comparisons, we predefine a checkpoint window. Each checkpoint is evaluated with four validation rollouts per task (n=4). We average verifier pass rates over tasks, rollouts, and checkpoints in that window. This metric uses no pass@4 aggregation or post-hoc best-checkpoint selection. Evaluation uses the benchmark verifier rather than the training progress signal or grouping metadata. Baselines with incomplete evaluation coverage are marked and reported using completed checkpoints only.

6.2 Task Success And Grouping Diagnostics

Tables 1–4 summarize the data splits and success rates. They also report collapse diagnostics, zero-shot transfer, and ablations. OT-73 denotes the OpenThoughts held-out split. Singleton rates count groups. Zero-advantage rates count steps. Variant and diagnostic terms are defined in the respective table captions.

Table 1: Controlled splits, de-noised plateau success (n=4), and singleton diagnostics.

Scale	Split	Tasks	Exact sing.	ASGPO sing.	GRPO	Exact	BiPACE	**ASGPO**
4B	OT-73	655/73	0.73	0.08	0.167	0.205	0.139*	**0.314**
7B	OT-73	655/73	0.92	—	0.196†	0.257	—	**0.346**
8B	SETA-70	156/70	0.98	0.10	0.414	0.334	—	**0.460**

†Completed checkpoints only. *Paper-based reference implementation, not the official code. Training used thinking, while evaluation used the common no-thinking protocol. This row is not a matched baseline.

Table 2: Terminal-Bench 2.0.

Method	Source	Protocol	Score
PPO	Endless Terminals	reported	0.067
Nemotron base	ours	matched	0.092
Exact	ours	matched	0.089
ASGPO	ours	matched	**0.127**

Table 3: 4B variants on OpenThoughts-73. κ labels context subgroups. Action phase denotes action type. Normalized exact matches canonicalized states. Random same-size shuffles members at fixed group sizes.

Role	Configuration	Factor	Mean
Ours	$\alpha = (\eta, \rho)$ + progress return	final anchor	**0.314**
Abl.	refined ASGPO with κ	context refinement	0.257
Abl.	no action phase	action phase	0.273
Abl.	no progress ρ	progress stage	0.219
Abl.	no feedback η	feedback mode	0.245
Abl.	raw-state + progress return	state abstraction	0.229
Abl.	progress return, no ASGPO	step grouping	0.188
Abl.	ASGPO, sparse return only	progress return	0.135
Ctrl.	normalized exact + sparse return	normalization alone	0.207
Ctrl.	random same-size + progress return	group size alone	0.163‡

‡Mean over the three completed step-50–70 evaluations. No step-80 result was available.

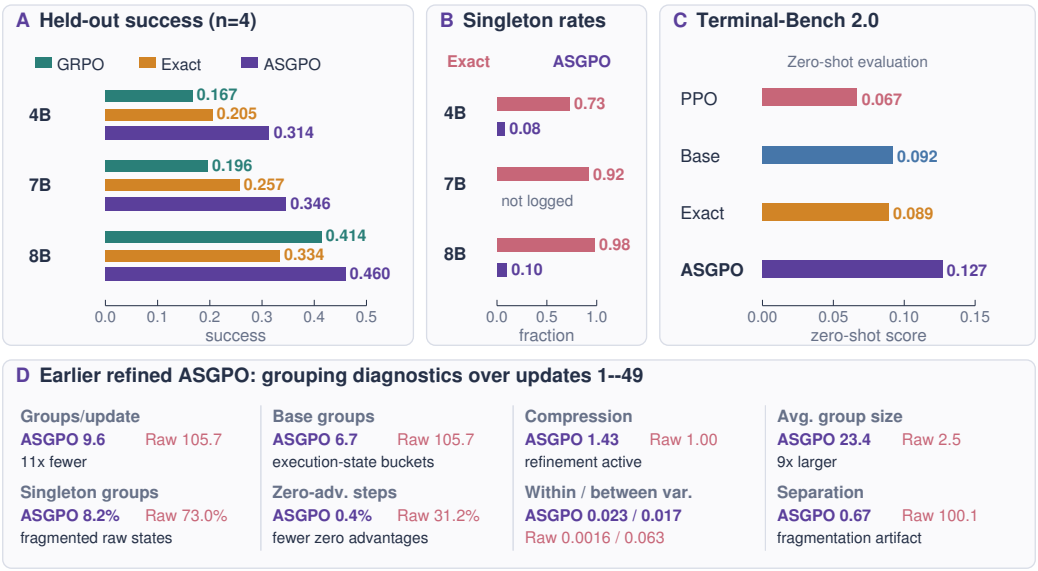

Figure 3: Results from Tables 1–4. Panels A–C show held-out success, singleton rates, and Terminal-Bench 2.0 transfer. Panel D reports diagnostics for the earlier refined ASGPO run. PPO is an externally reported result.

Table 4: Grouping diagnostics for earlier refined ASGPO and raw-state matching (Raw), updates 1–49. Base groups precede refinement. Compression is final/base group count. Within variance averages non-singleton return variances. Between variance measures group-mean dispersion. Separation is between/within. Metrics are averaged over updates.

Metric	Refined ASGPO	Raw	Interpretation
Groups/update	9.6	105.7	11× fewer
Base groups	6.7	105.7	execution-state buckets
Compression	1.43	1.00	refinement active
Avg. group size	23.4	2.5	9× larger
Singleton groups	8.2%	73.0%	fragmented raw states
Zero-adv. steps	0.4%	31.2%	fewer zero advantages
Within var.	0.023	0.0016	small raw-state groups
Between var.	0.017	0.063	–
Separation	0.67	100.1	fragmentation artifact

6.3 Main Findings

Controlled comparisons. ASGPO outperforms matched Exact at all three scales (Table 1, Figure 3A). Gains are $+0.110$ at 4B, $+0.089$ at 7B, and $+0.126$ at 8B. The 4B gain is significant under paired task bootstrap (95% CI $[0.048, 0.176]$). The 8B singleton rate falls from 0.98 to 0.10 (Figure 3B), showing reduced fragmentation.

Component evidence. The ablations support both abstract grouping and progress-aware returns (Table 3, Figure 4). Removing grouping yields 0.188. Using sparse returns yields 0.135. These ablations remove abstract peer selection or progress-based return distinctions. Removing the progress bucket yields 0.219, supporting stage-aware comparisons. In contrast, removing refinement raises the mean from 0.257 to 0.314, supporting the simpler final key.

The two coordinates control different comparisons. Removing η pools different feedback conditions at the same interaction stage. Removing ρ pools shallow and deep prefixes with the same feedback. Sparse verifier returns instead omit progress-based outcome differences. These ablations distinguish the role of comparison peers from that of the return used to rank them.

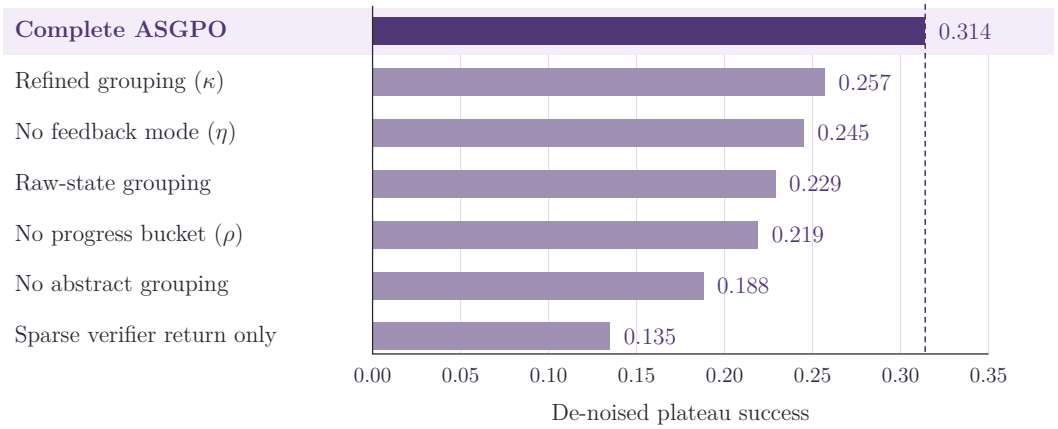

Figure 4: Selected component ablations on OpenThoughts-73. Bars show de-noised plateau mean success on a common axis. The dashed line marks the complete ASGPO result.

How grouping changes the baseline. Table 4 examines the earlier refined variant over 49 updates. Relative to raw-state matching, its mean group count falls from 105.7 to 9.6, and mean group size rises from 2.5 to 23.4. The zero-advantage step fraction falls from 31.2% to 0.4%. This step-level measure complements the singleton-group rate. A group with several steps can still produce zero advantages if all returns coincide. Singleton and zero-advantage rates therefore describe group occupancy and the availability of relative credit, respectively.

Interpreting return separation. Raw-state matching has a larger between/within variance ratio (100.1 versus 0.67). Its small non-singleton groups have little within-group return variation, inflating this ratio despite many zero advantages. A high separation ratio alone does not establish a useful baseline. Group size and zero-advantage rates are needed to interpret this diagnostic. This pattern is consistent with the trade-off in Theorem 1. Retaining more state detail can reduce within-group variation while leaving too few peers for comparison. The refinement ablation tests this balance: additional context distinctions do not improve the observed mean success.

Fixed anchor vocabulary. For final ASGPO, six feedback modes and four progress stages yield at most 24 labels per comparison pool. Transcript diversity cannot expand this label set.

Diagnostic controls. Normalized-exact with sparse returns (0.207) and partial-window random same-size grouping (0.163) yield lower means than ASGPO. Random same-size grouping preserves group sizes while shuffling membership. Its lower observed mean is consistent with peer selection mattering beyond occupancy, although the control covers only three completed evaluations. Normalized exact also uses sparse returns, so its score does not isolate the effect of state normalization. These controls and paper-based BiPACE (0.139) differ from the matched setup. The notes to Tables 1 and 3 give the settings. Appendix A provides further details.

Zero-shot transfer. ASGPO also improves zero-shot transfer on Terminal-Bench 2.0 (Table 2). It raises the locally evaluated Nemotron base from 0.092 to 0.127 and exceeds the matched exact-state estimator (0.089). Comparing with the released SFT base shows the overall change after RL fine-tuning under the same local evaluator. Comparing with matched Exact instead holds initialization, returns, and optimization fixed, making it a more direct test of the comparison-state choice. The published PPO result (0.067) (Gandhi et al., 2026) provides external context under a different setup.

7 Conclusion and Limitations

We introduced ASGPO for critic-free step-level credit assignment in CLI agents. It grouped transcripts by recent feedback and interaction depth, providing peers when identical states rarely recurred. Bounded progress-aware returns distinguished outcomes within these groups. Experiments showed improved held-out success across three model scales. The abstraction could still merge states with different continuation values. In future work, we will study learned refinements that separate such states while preserving group occupancy.

AI USE STATEMENT

Generative AI tools assisted with conceptual and theoretical presentation. They supported experiment design and debugging, analysis-code editing, and interpretation of logged results. They also assisted with scientific-figure revision and English-language editing. Formatting assistance covered LaTeX and the bibliography. Reported benchmark scores come from the benchmark verifiers. The authors take responsibility for the complete content of this work.

REPRODUCIBILITY STATEMENT

Section 6 specifies the controlled evaluation protocol. Appendix A records splits, checkpoints, and refinement constants. Appendix B gives implementation-level pseudocode. Appendix C provides complete proofs. Section F reports the logged grouping diagnostics.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, 2024.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive LLM agents. *arXiv preprint arXiv:2502.01600*, 2025.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yangyi Fang, Jiaye Lin, Xiaoliang Fu, Cong Qin, Chang Liu, and Haolin Shi. Proximity-based multi-turn optimization: Practical credit assignment for LLM agent training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 285–307, 2026.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, volume 38, pp. 46375–46408, 2025.

Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pp. 162–169, 2004.

Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling RL environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026.

Shuo He, Lang Feng, Qi Wei, Xin Cheng, Lei Feng, and Bo An. Hierarchy-of-groups policy optimization for long-horizon agentic tasks. In *International Conference on Learning Representations*, 2026.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free! In *Deep Reinforcement Learning Meets Structured Prediction, International Conference on Learning Representations Workshop*, 2019.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Zhihang Lin, Mingbao Lin, Yuan Xie, and Rongrong Ji. CPPO: Accelerating the training of group relative policy optimization-based reasoning models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 61043–61068, 2025.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. In *Conference on Language Modeling*, 2025.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling LLM terminal capabilities. *arXiv preprint arXiv:2602.21193*, 2026.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent Q: Advanced reasoning and learning for autonomous AI agents. *arXiv preprint arXiv:2408.07199*, 2024.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. In *Advances in Neural Information Processing Systems*, volume 38, pp. 105523–105553, 2025.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Hanyang Wang, Weijieying Ren, Yuxiang Zhang, Ding Cao, Zhizhao Zeng, Ke Zeng, and Tianxiang Zhao. BiPACE: Bisimulation-guided policy optimization with action counterfactual estimation for LLM agents. *arXiv preprint arXiv:2606.25556*, 2026a.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. OTC: Optimal tool calls via reinforcement learning. *arXiv preprint arXiv:2504.14870v1*, 2025a.

Tao Wang, Suhang Zheng, and Xiaoxiao Xu. RTMC: Step-level credit assignment via rollout trees. *arXiv preprint arXiv:2604.11037*, 2026b.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM reasoning via reinforcement learning on open software evolution. In *Advances in Neural Information Processing Systems*, volume 38, pp. 78500–78525, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094, 2024.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 23826–23854, 2023.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113222–113244, 2025.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62178–62209, 2024.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A Implementation Details

We use fixed task partitions throughout. The 4B/7B OpenThoughts-format experiments use 655 training tasks and 73 held-out tasks. The 8B experiments start from the released Nemotron-Terminal-8B SFT checkpoint and use a SETA dataset with 156 training tasks and a stratified 70-task held-out split. Within each scale, the primary ASGPO and Exact rows share the SFT initialization and task

split. They use the same rollout count and maximum environment steps. Decoding settings and optimizer schedules are matched. The checkpoint window and evaluator are also shared. Historical component variants and partial diagnostics retain their stated run settings.

For these held-out splits, validation uses verifier pass rate with four rollouts per task, and plateau scores average the pre-specified checkpoint window rather than selecting the highest-scoring checkpoint. For a validation set $\mathcal{T}$, the reported $n{=}4$ score is

$$\frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}}\frac{1}{4}\sum_{k=1}^{4}\mathbf{1}\{\text{verifier passes on rollout } k \text{ of } \tau\},$$

not pass@4. The evaluator reads only the benchmark verifier outcome, not the progress features or grouping metadata used during training. The 4B component sweep uses checkpoints 50, 60, 70 and 80 on the 73-task OpenThoughts held-out split. Primary pairwise claims use the same complete checkpoint set for both methods. Partial controls use only completed evaluations and are marked in the main table.

The final simplified ASGPO row uses the state-only key $\alpha(s) = (\eta(\tilde{s}), \rho(s))$. It excludes the sampled action and verifier label. It does not use conditional clustering. The original refined ASGPO row uses the optional refinement in Eq. (13). Its constants are $m = 4$, $\lambda = 0.5$, and $\gamma = 0.33$. The cluster cap is $P = 256$ and the hash dimension is $d = 256$. Clustering uses eight assignment-and-update iterations with centroid normalization.

Action phase. Action phase denotes a rule-based category of the sampled tool response, not the stage of the interaction. The classifier checks for submission, thinking, editor operations, and shell commands in that order. Shell commands are classified as testing, file inspection, installation, or other commands. An empty response has its own category; unmatched responses use an assistant category. When enabled, this category is included in the feedback–progress key before optional context refinement. When `include_action_phase=False`, it is replaced by a constant and does not distinguish groups. In contrast, $\rho(s)$ measures interaction depth from the pre-action transcript. The final ASGPO key excludes action phase.

Two early controls test normalization and group size as explanations for the gain. The normalized-exact control canonicalizes transcripts before exact matching. It uses sparse returns with within-group standardization. The random same-size control preserves ASGPO's empirical group-size distribution. It permutes group membership independently of state semantics. Their plateau means are 0.207 and 0.163, respectively. The random same-size mean uses the three completed step-50–70 evaluations. Its step-80 evaluation did not finish. We treat this partial-window number as a diagnostic control and exclude it from paired significance claims.

The BiPACE row uses our implementation of the published method description, not the official code. Its thinking-enabled training checkpoints were evaluated under the common no-thinking $n{=}4$ protocol. Evaluation covers steps 10, 30 and 40, plus steps 50 and 60. The five-point mean is 0.139. Its training mode and checkpoint window differ from the primary matched comparisons. We report this row as a reference comparison.

The refined component row uses the following additional operation. For a base anchor b with bucket $\mathcal{B}_b = \{i : (\eta(\tilde{s}_i), \rho(s_i)) = b\}$, choose

$$k_b = \mathrm{clip}(\lfloor \lambda |\mathcal{B}_b|^{\gamma} \rceil, 1, \min(P, |\mathcal{B}_b|)) . \tag{12}$$

If $|\mathcal{B}_b| < m$, then $\kappa_b \equiv 0$ and the base anchor is kept intact. For larger buckets, κ_b returns the k-means cluster id of a deterministic context embedding $\psi(\tilde{s}_i)$ inside bucket b. The refined anchor used only for that ablation is

$$\alpha_{\mathrm{ref}}(s_i) = (\eta(\tilde{s}_i), \rho(s_i), \kappa_b(\psi(\tilde{s}_i))) . \tag{13}$$

The component sweep selected the simpler state-only anchor. The main method excludes the refinement term.

Context embedding and clustering. The optional embedding ψ uses only the normalized pre-action context; the sampled response is excluded. Tokens are extracted using the regular expression

```
<[A-Z]+>|[a-zA-Z_][a-zA-Z0-9_./-]{1,}|[0-9]+
```

Each token occurrence and adjacent token pair contributes one signed count to a 256-dimensional vector. Pairs are joined with an underscore before hashing. For each feature, the implementation computes an eight-byte BLAKE2b digest and interprets it as an unsigned little-endian integer. The hash modulo 256 selects the coordinate; bit 8 selects the sign ($+1$ when set and -1 otherwise). Counts that share a coordinate are added. The vector is divided by the larger of its L_2 norm and 10^{-6}, leaving an empty feature vector at zero. This construction requires no learned encoder.

Clustering starts with the first vector in each bucket. Each additional centroid is the vector with the largest squared distance to its nearest selected centroid. Each iteration assigns vectors to the nearest centroid in squared Euclidean distance and replaces each nonempty cluster's centroid by its mean. Centroids are then L_2-normalized with the same 10^{-6} safeguard; empty clusters retain their previous centroid. Seeding ties use the first candidate in bucket order; assignment ties use the first centroid. The implementation performs eight iterations. Buckets with one requested cluster remain intact; when $k_b = |\mathcal{B}_b|$, each sample receives its own label.

To link these configurations to their records, we retain the original run identifiers in Appendix E. The result tables exclude single-rollout validation and known evaluation-cache contamination. The BiPACE and random same-size rows remain as marked diagnostics. Their individual checkpoints completed de-noised n=4 evaluation. Their checkpoint-window coverage is incomplete or differs from the primary comparison.

The Terminal-Bench 2.0 rows are handled separately from the controlled tables. They measure zero-shot transfer. TB2 tasks are not used for RL training or reward tuning. They are also excluded from component and checkpoint selection. Local TB2 rows use the same harness and 79 scored tasks from the 89-task suite, while published external numbers are labeled as reported rather than treated as controlled baselines. The matched held-out splits support the statistical comparisons, and TB2 measures out-of-domain transfer from the public terminal SFT base.

B Full ASGPO Algorithm

This section specifies the complete ASGPO update. The state map $\alpha(s) = (\eta(\tilde{s}), \rho(s))$ selects peers for the group-relative advantage in Eq. (1). Deterministic rules extract comparison keys from pre-action transcripts and progress features from benchmark-visible tool outputs. The steps below define these rules and their use in the policy update.

Inputs and outputs. One policy update receives a batch of rollouts

$$\mathcal{B} = \{\tau^m\}_{m=1}^{M}, \qquad \tau^m = (s_1^m, a_1^m, o_1^m, \ldots, s_{T_m}^m, a_{T_m}^m, o_{T_m}^m),$$

where s_t^m is the pre-action prompt/transcript prefix. The action a_t^m is sampled by the old policy. The terminal observation o_t^m follows execution of that action. Let $\mathcal{I} = \{(m,t) : 1 \le m \le M,\ 1 \le t \le T_m\}$ index all sampled steps. The environment verifier produces an episode outcome $R_{\text{env}}^m \in \{0,1\}$ or an equivalent benchmark score. ASGPO outputs step-level advantages $\hat{A}_t^m$ used in the clipped policy update. It logs group identifiers and group sizes. It also records singleton flags and progress features. These diagnostics are not used to compute evaluation scores.

Semantics of a multi-instance abstract anchor. For an abstract state $z \in \mathcal{Z}$, define its concrete support in the current update as

$$\mathcal{C}_z(\mathcal{B}) = \{s_t^m : (m,t) \in \mathcal{I},\ \alpha(s_t^m) = z\}. \tag{14}$$

The support $\mathcal{C}_z$ contains the distinct observed concrete states with label z. The corresponding group contains all step samples carrying this label, including repeated occurrences of the same state. Group size therefore counts step samples rather than $|\mathcal{C}_z|$ distinct states. An abstract state is "anchored by multiple concrete states" when those states have equal outputs under α. The shared label represents the class. Each group is formed from observed steps in the current update. The key is a class label rather than a medoid or prototype transcript. Group construction uses neither retrieved demonstrations nor separately generated anchors. This distinction matters in a terminal environment. A prototype retains its particular path or hash. An equivalence class can be invariant to both.

Design criteria for the abstraction. We chose the coordinates of α to meet four requirements. *Pre-action measurability* requires the key to be determined before the action whose advantage is

estimated. *Continuation relevance* requires members to admit similar next actions and have comparable remaining horizons. *Nuisance invariance* prevents temporary identifiers and absolute paths from splitting a group. Line numbers, timestamps, and verbose trace text should not split it either. *Batch occupancy* requires a representation coarse enough that the same abstract state occurs multiple times in a practical rollout batch. Feedback mode η mainly addresses continuation relevance and nuisance invariance. Progress stage ρ separates coarse interaction depths that often have different remaining horizons. Their Cartesian product retains both execution variables with a small key space. The component sweep evaluates the effect of removing either coordinate.

Additional transcript fields introduce a trade-off. A new coordinate can reduce within-group value heterogeneity, but it also partitions each support set $\mathcal{C}_z$ into smaller sets and their step samples into smaller groups. This reduces the average leave-in retention factor in Eq. (9). A group with only one step sample has exactly zero centered numerator. This trade-off motivates testing the context coordinate κ. The refined key retains more detail but has a lower observed held-out mean, so the final method excludes it. Removing ρ merges states at different interaction depths whenever they share the same feedback mode.

Concrete grouping examples. Consider two pre-action prefixes from different rollouts. The first contains a Python traceback under `/tmp/run_42` after initial inspection. The second contains an assertion traceback under `/workspace/job_77` after similarly early error localization. Their paths, line numbers, and exception messages differ. Exact-state grouping therefore separates them. ASGPO extracts $(\texttt{traceback}, \texttt{early})$ from both and places them in the same support set. Their subsequent returns provide a within-group comparison of debugging actions.

Consider a third prefix whose recent window contains a passing check and no higher-priority failure after at least seven completed interactions. It maps to $(\texttt{passing}, \texttt{late})$ and is not merged with the two early failures. A fourth prefix with a traceback after four completed interactions maps to $(\texttt{traceback}, \texttt{middle})$. This illustrates why feedback alone is insufficient to distinguish interaction stages. The abstraction therefore pools states with different raw text when their feedback labels and interaction-depth buckets agree.

Step 1: transcript normalization. Raw terminal prefixes vary with temporary directories and run identifiers. Object hashes and random ports introduce further variation. Logs also differ in line numbers, timestamps, and stack traces. To reduce these incidental differences, ASGPO first applies a deterministic normalizer

$$\tilde{s}_t = N(s_t),$$

which uses only visible transcript content to reduce run-specific variation before feedback extraction. The feedback extractor then checks the recent prefix for tracebacks and permission failures. It also checks for missing resources, generic errors, and passing checks.

Step 2: recent feedback mode η. The feedback mode summarizes the most salient recent terminal outcome before the next action is chosen. It is computed from the last 2,048 characters of the normalized prefix by the following ordered rule:

$$\eta(\tilde{s}) = \begin{cases} \texttt{traceback}, & \text{recent stack trace, exception, or assertion crash,} \\ \texttt{permission}, & \text{the prefix contains “permission denied”,} \\ \texttt{missing}, & \text{a file, command, or resource is not found,} \\ \texttt{error}, & \text{failure, error, exit-code, or return-code evidence,} \\ \texttt{passing}, & \text{passed, success, or all-tests evidence,} \\ \texttt{normal}, & \text{no salient failure or success signal.} \end{cases}$$

The priority ordering matters. For example, a failed test command that contains a Python traceback should be grouped as `traceback`. The next useful action is usually debugging the exception rather than simply rerunning tests. A missing resource should not be merged with an arbitrary nonzero exit. Their continuation values and useful next actions differ.

Step 3: progress bucket ρ. The progress bucket is a coarse interaction-depth summary. Let $d(s)$ be the number of completed agent–environment interactions represented in prefix s. The deployed

deterministic rule is

$$\rho(s) = \begin{cases} \texttt{start}, & d(s) = 0, \\ \texttt{early}, & 1 \le d(s) \le 2, \\ \texttt{middle}, & 3 \le d(s) \le 6, \\ \texttt{late}, & d(s) \ge 7. \end{cases}$$

The implementation reads the stored zero-based interaction index, which is equivalent to $d(s)$ for the accumulated transcript prefix. The coordinate ρ is therefore determined by the pre-action prefix. The current action, reward, and verifier outcome are excluded from its inputs. Its coarse thresholds prevent an initial exploration state from being compared directly with a much deeper interaction state while preserving useful group occupancy.

Step 4: abstract anchor construction. The deployed anchor is

$$g_t^m = \alpha(s_t^m) = (\eta(N(s_t^m)), \rho(s_t^m)) .$$

This pair defines the abstract state used to select comparison peers in the main algorithm. Different concrete transcripts share an anchor whenever both coordinates agree. Neither the current sampled action nor its subsequent reward, verifier result, or observation is part of g_t^m. Earlier execution feedback is already present in the prefix. The group mean serves as a baseline for the current action. Selecting its peers using that action's reward could remove or distort the signal being estimated.

The construction induces the relation

$$s_i \sim_{\mathrm{ASGPO}} s_j \quad \Longleftrightarrow \quad \eta(N(s_i)) = \eta(N(s_j)) \ \wedge \ \rho(s_i) = \rho(s_j). \tag{15}$$

This relation is equality of two deterministic categorical coordinates. It is reflexive, symmetric, and transitive. The relation therefore partitions observed pre-action states. Group membership follows coordinate equality and is independent of processing order. The same transcript retains its label as policy parameters change. The samples carrying that label still depend on the current batch.

The optional refinement ablation appends a context cluster id, $g = (\eta, \rho, \kappa)$, where κ is computed from a deterministic embedding inside each base bucket. The ablation in Table 3 shows that this extra subdivision has a lower observed mean on the 4B split. The final method therefore excludes κ. We keep the refinement derivation in Appendix C. It explains why excessive precision can increase the number of singleton groups.

Step 5: progress return assembly. The following definitions expand the execution score used in Section 4.2. Before episode-level rescaling, the score is

$$\begin{aligned} r_t^{\mathrm{prog}} = c_0 &+ c_{\mathrm{cmd}} n_t^{\mathrm{ok}} + c_{\mathrm{test}} n_t^{\mathrm{test}} \\ &- p_{\mathrm{err}} e_t - p_{\mathrm{rep}} q_t - p_{\mathrm{to}} h_t \\ &- p_{\mathrm{net}} w_t - p_{\mathrm{tf}} f_t - p_{\mathrm{cap}} c_t. \end{aligned} \tag{16}$$

All event quantities aggregate the tool results produced at step t. The two positive counts are n_t^{ok} for successful command exits and n_t^{test} for successful exits of recognized test runners. A passing local test need not imply success under the final task verifier. The coefficients c_{cmd} and c_{test} weight these events. The constant valid-tool bonus c_0 is disabled.

The penalty variables have the following meanings:

- e_t counts tool errors or nonzero exits, excluding internal `think` actions. Its coefficient is p_{err}.
- q_t measures repeated normalized shell commands within the episode. Each repeat contribution is capped before aggregation and weighted by p_{rep}.
- h_t counts timeouts detected from exit codes or termination messages. Its coefficient is p_{to}.
- w_t counts offline-network probes identified from command patterns. Its coefficient is p_{net}.
- f_t counts failed test-runner commands, weighted by p_{tf}.
- c_t counts known capture-artifact shortcuts, weighted by p_{cap}. This benchmark-specific detector targets commands that fabricate the expected artifact rather than perform the requested work.

These detectors read command strings, exit codes, and visible tool output. More than one event may apply to a command. For example, a passing test can contribute both the successful-command and test-success terms. A failed test can contribute both the error and test-failure penalties.

The fixed reward weights are $c_{\mathrm{cmd}} = 0.001$ and $c_{\mathrm{test}} = 0.060$. The penalties are $p_{\mathrm{err}} = 0.006$, $p_{\mathrm{rep}} = 0.015$ and $p_{\mathrm{to}} = 0.040$. We also use $p_{\mathrm{net}} = 0.030$, $p_{\mathrm{tf}} = 0.020$ and $p_{\mathrm{cap}} = 0.030$. Both episode caps are $C^{+} = C^{-} = 0.060$.

For every post-action observation o_t^m, ASGPO computes the execution score r_t^{prog} from Eq. (16). Positive and negative progress masses are capped by rescaling as in Eq. (5). The rescaled deltas are then converted into suffix returns:

$$D_t^m = \sum_{u=t}^{T_m} r_u^{\mathrm{prog}}, \qquad R_t^m = R_{\mathrm{env}}^m + D_t^m + \Gamma_t^m,$$

where Γ_t^m contains the short-failed-episode and context-truncation guards described in Section 4.2. The suffix form means that a useful late signal, such as a passing test, gives credit to earlier actions in that same trajectory. The caps keep this process signal small relative to a full task-completion reward.

Step 6: group-normalized advantages. All step samples in the update batch are collected into groups

$$\mathcal{I}_g = \{(m,t) : g_t^m = g\}.$$

For each nonempty group, compute

$$\mu_g = \frac{1}{|\mathcal{I}_g|} \sum_{(m,t)\in\mathcal{I}_g} R_t^m, \qquad \sigma_g = \mathrm{std}_{(m,t)\in\mathcal{I}_g} R_t^m.$$

The advantage for step (m,t) is

$$\hat{A}_t^m = \frac{R_t^m - \mu_{g_t^m}}{\sigma_{g_t^m} + \epsilon}.$$

The baseline uses only sampled returns from the same group in the current update. If a group has a single element, then $\mu_g = R_t^m$ and the centered numerator is zero. Such a step contributes no centered policy-gradient signal, as analyzed in Section 5. It can still contribute to auxiliary terms such as KL regularization.

Step 7: policy update. Let

$$r_t^m(\theta) = \frac{\pi_\theta(a_t^m \mid s_t^m)}{\pi_{\theta_{\mathrm{old}}}(a_t^m \mid s_t^m)}.$$

The policy maximizes the clipped GRPO-style surrogate

$$\mathcal{L}_{\mathrm{clip}}(\theta) = \frac{1}{|\mathcal{I}|} \sum_{(m,t)\in\mathcal{I}} \min\Big(r_t^m(\theta)\hat{A}_t^m,\ \mathrm{clip}(r_t^m(\theta), 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}})\hat{A}_t^m \Big),$$

with a KL penalty to the SFT reference policy. The update remains critic-free and retains the clipped policy-ratio objective. ASGPO changes the comparison set used to standardize returns.

Pseudocode. The complete update can be summarized as

1. Collect rollouts with the old policy and store every pre-action prefix s_t^m, action a_t^m, and post-action observation o_t^m.
2. For each step, compute $\tilde{s}_t^m = N(s_t^m)$, then $\eta_t^m = \eta(\tilde{s}_t^m)$ and $\rho_t^m = \rho(s_t^m)$.
3. Set the abstract anchor $g_t^m = (\eta_t^m, \rho_t^m)$ before using any reward or future observation from that step.
4. Compute bounded progress deltas from observations, apply episode-level positive/negative caps, and form suffix returns D_t^m.
5. Form the training return $R_t^m = R_{\mathrm{env}}^m + D_t^m + \Gamma_t^m$.
6. Group all step samples by g_t^m and compute (μ_g, σ_g) for each nonempty group.

7. Standardize each step return into $\hat{A}_t^m$ and update the policy with the clipped GRPO surrogate plus KL regularization.
8. Log group counts and average group size. Record singleton and zero-advantage rates. Record within-group and between-group return variance.

Why each component is needed. The algorithm has three coupled design choices. The feedback mode η separates observed failure and success categories, so a missing-resource failure is not compared with a traceback or a passing check. The progress bucket ρ separates shallow and deep prefixes even when they share the same feedback label. The bounded progress return makes differences in execution outcomes visible even in failed rollouts. Each ablation tests a corresponding role: removing η merges feedback categories, removing ρ merges interaction stages, and using sparse returns alone leaves failed rollouts with nearly identical returns. Replacing the abstract key with the raw state tests whether exact matching fragments these comparison groups into singletons.

B.1 Relation To The GiGPO Estimator

GiGPO's step component compares actions from common observed states using $g^{\text{exact}}(i) = s_i$ in Eq. (1). Its full estimator adds an episode-level advantage (Feng et al., 2025):

$$A_m^E = \frac{R_m - \text{mean}_{j\in[M]} R_j}{F_{\text{norm}}(\{R_j\}_{j\in[M]})}, \tag{17}$$

$$A_i^{\text{GiGPO}} = A_{\tau(i)}^E + \omega A_i^S(g^{\text{exact}}), \tag{18}$$

where $\tau(i)$ indexes the trajectory containing step i and ω weights the step term. For singletons, the step term vanishes but the episode term remains. Our matched EXACT baseline uses concrete-state identity with ASGPO's returns and optimizer. Both include the episode verifier in R_i before group normalization and do not add the separate A^E term.

Concrete-state identity and ASGPO's feedback–progress map choose different comparison resolutions. Identity provides precise peers when observed states recur. The abstract map pools distinct transcripts with shared feedback and stage. Appendix C formalizes the resulting trade-off between group occupancy and continuation-value heterogeneity. It identifies conditions under which coarser grouping improves baseline estimation.

Relation to contemporaneous abstraction choices. ProxMO, BiPACE, and RTMC also weaken raw observation identity (Fang et al., 2026; Wang et al., 2026a;b). Their concrete supports are defined differently. ProxMO uses soft TF–IDF-weighted neighborhoods rather than discrete equivalence classes. BiPACE forms clusters in the current actor's hidden representation and then conditions peers on action identity. RTMC maps histories to cumulative state–action signatures and treats shared rollout-tree nodes as states. ASGPO's support sets are instead determined solely by two human-readable pre-action process variables. This provides a deterministic, computationally inexpensive comparison rule that maps variable terminal transcripts to a small set of execution categories.

Failure cases and safeguards. The two-coordinate key can merge states with different continuation values. For example, if two tasks produce superficially similar tracebacks but require unrelated fixes, the abstraction can pool incomparable steps. Priority-ordered feedback modes and coarse progress stages separate different observable conditions, but they do not resolve every distinction within a feedback–progress group. We therefore examine both occupancy and return variation in the grouping study: singleton and zero-advantage rates diagnose missing comparisons, while within-group variance helps assess heterogeneity among the available peers.

C DERIVATIONS FOR ASGPO

Abstraction family and nested estimators. Let S denote the complete pre-action information state available to the policy. In our CLI setting this is the terminal transcript and associated file context, not necessarily a minimal Markov state. Consider a family Φ of deterministic maps $\phi : \mathcal{S} \to \mathcal{Z}_\phi$. Each map induces the equivalence relation

$$s \sim_\phi s' \quad \Longleftrightarrow \quad \phi(s) = \phi(s'), \tag{19}$$

Table 5: How representative methods decide whether the two concrete early traceback states in Eq. (4) should share credit. This comparison concerns the state relation. The methods also differ in their return and advantage estimators.

Method	Evidence defining peers	Treatment of $s^{(1)}, s^{(2)}$ and implication
GiGPO/ HGPO	Exact observation identity. HGPO adds historical consistency.	Separate under byte-wise inequality $s^{(1)} \neq s^{(2)}$. Precise only when raw observations recur.
ProxMO	TF–IDF cosine similarity of full state text	Assign a soft weight based on lexical overlap. Paths and logs can alter the weight despite the shared execution phase.
BiPACE	Cosine clusters of normalized actor hidden states. Action-conditioned peers follow.	Merge only within the chosen representation radius. This requires an extra representation pass and layer/radius choices.
RTMC	Cumulative state–action/effect signatures and rollout-tree nodes	Merge only under the same parsed signature. This depends on the domain signature and tree construction.
ASGPO (ours)	Recent feedback mode η and prefix-depth stage ρ	Merge under the common label $(\texttt{traceback}, \texttt{early})$. Paths and exception text are nuisance variables by construction.

and therefore a partition of sampled steps. For step i, write $g_i^\phi = \phi(s_i)$ and $\mathcal{I}_z^\phi = \{i : g_i^\phi = z\}$. The common estimator represented by this family is

$$\widehat{A}_i(\phi) = \frac{R_i - \bar{R}_{g_i^\phi}}{\widehat{\sigma}_{g_i^\phi} + \epsilon}, \tag{20}$$

where $\widehat{\sigma}_z$ is the empirical return standard deviation in cell z. Setting $\phi = \phi_{\mathrm{id}}$, with $\phi_{\mathrm{id}}(s) = s$, gives the matched exact-state estimator corresponding to GiGPO's step-level component. The deployed ASGPO estimator uses $\phi = \alpha = (\eta, \rho)$. The refinement ablation uses $\phi = (\alpha, \kappa)$. GiGPO's original estimator separately adds a trajectory-normalized term as shown in Eq. (18). In our controlled comparison, both ASGPO and Exact include the episode verifier in R_i before group normalization. Adding a separate episode advantage would change this estimator, so it is excluded from both methods.

This construction includes the concrete state as an *endpoint of the map family*. It does not append it to every abstract key. The equality $(s, \alpha(s)) = (s', \alpha(s'))$ holds if and only if $s = s'$. The concatenated map $s \mapsto (s, \alpha(s))$ induces exactly the identity partition. It cannot pool steps from distinct concrete states. The map ϕ_2 refines ϕ_1 when there is a deterministic h such that $\phi_1 = h \circ \phi_2$. Every state map is coarser than the identity, as $\phi = \phi \circ \phi_{\mathrm{id}}$.

The abstraction is estimator-level: $\phi(S)$ chooses which sampled returns are compared when constructing a baseline. The policy still conditions on the full S. Environment transitions are unchanged. The theorem does not require ϕ to be an MDP homomorphism or bisimulation. The relevant error is instead the continuation-value heterogeneity within a comparison cell.

Setup and conditioning. The analysis in Section 5 conditions on a fixed peer-selection design. Consider one update batch containing N step-level samples indexed by $\mathcal{I} = \{1, \ldots, N\}$. Step i has pre-action terminal state s_i and sampled action a_i. Its return is R_i and its score is

$$u_i = \nabla_\theta \log \pi_\theta(a_i \mid s_i). \tag{21}$$

An abstraction map constructs a group id g_i from pre-action information in the batch. For deployed ASGPO, these inputs are the transcript-derived feedback bucket and the prefix-depth bucket. Only the optional refinement ablation adds a deterministic context embedding. In the independent-branch model, let $\mathcal{F}$ fix the pre-action states and peer-selection design, including the partition. Assume $\Pr(a_i = a \mid \mathcal{F}) = \pi_\theta(a \mid s_i)$. For every peer $j \neq i$ in the target group, assume that R_j is generated on a separate branch and is conditionally independent of a_i given $\mathcal{F}$. Return independence is required for every other peer in the group, not merely for cross-trajectory pairs. These are sampling-design conditions, not consequences of an action-independent key alone.

A retrospective partition of full trajectories can depend on a_i through later states in its branch. Conditioning on such a partition need not preserve the conditional action law above. The algebraic expansion below remains valid for a fixed realized partition without assuming a zero conditional score mean. It therefore retains cross terms for general trajectory batches. The zero-cross-term reduction

uses the independent-branch conditions explicitly. Deployed ASGPO excludes the current action and future outcomes from each step's key, but may pool dependent steps from the same trajectory.

For a group g, define

$$\mathcal{I}_g = \{j \in \mathcal{I} : g_j = g\}, \qquad n_g = |\mathcal{I}_g|,$$
$$\bar{R}_g = \frac{1}{n_g} \sum_{j \in \mathcal{I}_g} R_j. \tag{22}$$

All expectations below are on-policy expectations. Under the conditional action law specified above, the score has zero conditional mean:

$$\begin{aligned}\mathbb{E}[u_i \mid \mathcal{F}] &= \sum_a \pi_\theta(a \mid s_i) \nabla_\theta \log \pi_\theta(a \mid s_i) \\ &= \sum_a \nabla_\theta \pi_\theta(a \mid s_i) = \nabla_\theta \sum_a \pi_\theta(a \mid s_i) = 0,\end{aligned} \tag{23}$$

with the analogous integral form for continuous actions. The same-step return R_i can depend on the sampled action. Its correlation with u_i supplies the policy-gradient signal. Under the independent-branch assumptions, every other peer's return R_j is independent of u_i conditional on $\mathcal{F}$. Different steps from the same sampled trajectory can remain dependent. The general derivation retains this dependence in a cross-step residual. The residual vanishes when the stated conditions cover every peer in the group.

Leave-in group centering. We derive Eq. (8) by separating the target return from its peers in the group mean. Fix a step i and let $g_i = g$. Conditional on $\mathcal{F}$, the leave-in group mean is a sum over the fixed index set $\mathcal{I}_g$, but its values are still random through the sampled actions and environment outcomes. Expanding the product gives

$$\begin{aligned}\mathbb{E}[u_i \bar{R}_g \mid \mathcal{F}] &= \frac{1}{n_g} \sum_{j \in \mathcal{I}_g} \mathbb{E}[u_i R_j \mid \mathcal{F}] \\ &= \frac{1}{n_g} \mathbb{E}[u_i R_i \mid \mathcal{F}] + \frac{1}{n_g} \sum_{\substack{j \in \mathcal{I}_g \\ j \neq i}} \mathbb{E}[u_i R_j \mid \mathcal{F}].\end{aligned} \tag{24}$$

Define the cross-step residual

$$\Delta_i(\mathcal{F}) = \frac{1}{n_g} \sum_{\substack{j \in \mathcal{I}_g \\ j \neq i}} \mathbb{E}[u_i R_j \mid \mathcal{F}]. \tag{25}$$

Subtracting Eq. (24) from the uncentered policy-gradient term then yields the exact conditional identity

$$\begin{aligned}&\mathbb{E}[u_i (R_i - \bar{R}_g) \mid \mathcal{F}] \\ &\quad = \mathbb{E}[u_i R_i \mid \mathcal{F}] - \frac{1}{n_g} \mathbb{E}[u_i R_i \mid \mathcal{F}] - \Delta_i(\mathcal{F}) \\ &\quad = \left(1 - \frac{1}{n_g}\right) \mathbb{E}[u_i R_i \mid \mathcal{F}] - \Delta_i(\mathcal{F}).\end{aligned} \tag{26}$$

For a peer $j \neq i$ that satisfies the independent-branch conditions, conditional independence and Eq. (23) give

$$\mathbb{E}[u_i R_j \mid \mathcal{F}] = \mathbb{E}[u_i \mid \mathcal{F}] \mathbb{E}[R_j \mid \mathcal{F}] = 0, \tag{27}$$

When every peer $j \neq i$ in $\mathcal{I}_g$ satisfies these conditions, all cross terms vanish, so $\Delta_i(\mathcal{F}) = 0$. Eq. (26) then reduces to the shrinkage expression in the main text. For a group containing dependent steps from the target trajectory, the corresponding terms remain in Δ_i. The singleton conclusion is exact in both cases: $n_g = 1$ implies $R_i - \bar{R}_g = 0$ deterministically. Marginalizing over the random partition gives the unconditional identity

$$\mathbb{E}[u_i (R_i - \bar{R}_g)] = \mathbb{E}\left[\left(1 - \frac{1}{n_g}\right) \mathbb{E}[u_i R_i \mid \mathcal{F}] - \Delta_i(\mathcal{F})\right]. \tag{28}$$

The main text uses the conditional form for a realized partition. The identity separates finite-sample shrinkage from the cross-step residual. The grouping rule affects both the group sizes and which peer returns enter that residual.

For comparison, define the leave-one-out group baseline

$$\bar{R}_{g,-i} = \frac{1}{n_g - 1} \sum_{\substack{j \in \mathcal{I}_g \\ j \neq i}} R_j, \qquad n_g > 1. \tag{29}$$

For independent rollout branches, the same cross-term argument gives

$$\mathbb{E}[u_i(R_i - \bar{R}_{g,-i}) \mid \mathcal{F}] = \mathbb{E}[u_i R_i \mid \mathcal{F}]. \tag{30}$$

Leave-one-out centering is unbiased at the group level in the independent-branch model. The leave-in GRPO mean instead shrinks the direct same-step signal by the factor $1 - 1/n_g$. With the leave-in mean, larger groups reduce this shrinkage, and the retention factor approaches one as group size increases. If $n_g = 1$, then $R_i - \bar{R}_g = 0$ exactly, regardless of the action's quality.

Role of standard-deviation normalization. Eq. (1) divides the centered return by the within-group empirical standard deviation $\widehat{\sigma}_g$ plus ϵ. This positive denominator depends on the observed rewards. An exact expectation for $u_i(R_i - \bar{R}_g)/(\widehat{\sigma}_g + \epsilon)$ therefore requires the joint reward distribution. Two conclusions hold without that model. If $n_g = 1$, the numerator is exactly zero. Standardization cannot recover a learning signal in this case. The positive denominator preserves the sign of a nonzero centered return. To analyze larger groups further, assume independent sampling and bounded rewards, and suppose the empirical scale concentrates at rate $n_g^{-1/2}$ around a positive population scale $\sigma_g^\star$. The reciprocal scale then admits the expansion

$$\frac{1}{\widehat{\sigma}_g + \epsilon} = \frac{1}{\sigma_g^\star + \epsilon} + O_p(n_g^{-1/2}). \tag{31}$$

The singleton result holds exactly with standardization. The expansion describes scale concentration. Transferring it to an expected gradient also requires moment control on the score–return product.

Signal retention. The leave-in derivation gives a per-step shrinkage factor $1 - 1/n_{g_i}$. A scalar occupancy summary is the average leave-in retention factor in a realized batch under map ϕ:

$$\begin{aligned} \mathcal{S}_N(\phi) = \frac{1}{N} \sum_{i=1}^{N} \left(1 - \frac{1}{n_{g_i}}\right) &= 1 - \frac{1}{N} \sum_{g \in \mathcal{G}} \sum_{i \in \mathcal{I}_g} \frac{1}{n_g} \\ &= 1 - \frac{|\mathcal{G}|}{N}. \end{aligned} \tag{32}$$

Each nonempty group contributes exactly one unit to $\sum_{i \in \mathcal{I}_g} 1/n_g$, regardless of its size. At a fixed batch size, the number of nonempty groups determines the average shrinkage coefficient in the independent-branch analysis. Singleton groups are the most extreme case: they contribute one group and one step, and their individual retention is zero. If all groups had size m, then $\mathcal{S}_N(\phi) = 1 - 1/m$. If every step were its own raw state, then $|\mathcal{G}| = N$ and $\mathcal{S}_N(\phi) = 0$.

The mechanism metrics in Section F therefore quantify the occupancy component of leave-in centering. The factor averages coefficients rather than gradient magnitudes. Return heterogeneity, cross-step dependence, and empirical standardization also affect the realized update.

Proof of retention monotonicity. Let ϕ_2 refine ϕ_1, so $\phi_1 = h \circ \phi_2$. Every nonempty ϕ_2 cell is contained in exactly one ϕ_1 cell. Every nonempty ϕ_1 cell in a realized batch contains at least one nonempty ϕ_2 cell. The induced map from fine cells to coarse cells is therefore surjective, which implies

$$|\mathcal{G}_{\phi_2}| \geq |\mathcal{G}_{\phi_1}|. \tag{33}$$

Substitution into the finite-batch identity above gives

$$\mathcal{S}_N(\phi_2) = 1 - \frac{|\mathcal{G}_{\phi_2}|}{N} \leq 1 - \frac{|\mathcal{G}_{\phi_1}|}{N} = \mathcal{S}_N(\phi_1), \tag{34}$$

which proves the first monotonicity in Theorem 1. The identity map refines every deterministic state map. Exact-state grouping therefore has no greater retention than any of its coarsenings on the same batch.

Proof of aliasing monotonicity and baseline MSE. Merging every step into one group maximizes the leave-in retention factor. However, its mean can mix states with different continuation values. We therefore analyze the error of the group mean as an estimate of a state-dependent continuation value.

Let S be a random pre-action information state and $V(S) = \mathbb{E}[R \mid S]$. Write

$$R = V(S) + \xi, \qquad \mathbb{E}[\xi \mid S] = 0,$$
$$\mathrm{Var}(\xi \mid S) = \sigma^2(S). \tag{35}$$

For $z = \phi(S)$, let

$$\bar{V}_z = \mathbb{E}[V(S) \mid z], \qquad H_z = \mathrm{Var}(V(S) \mid z),$$
$$\bar{\sigma}_z^2 = \mathbb{E}[\sigma^2(S) \mid z]. \tag{36}$$

Here H_z measures the variation in continuation value within a comparison cell, referred to as value aliasing or heterogeneity. Its population average is

$$\mathcal{H}(\phi) = \mathbb{E}[H_{\phi(S)}] = \mathbb{E}[\mathrm{Var}(V(S) \mid \phi(S))]. \tag{37}$$

To prove the second monotonicity, set $Z_1 = \phi_1(S)$ and $Z_2 = \phi_2(S)$. Refinement means $Z_1 = h(Z_2)$, so conditioning on Z_2 reveals at least as much information as conditioning on Z_1. Applying the conditional law of total variance inside each Z_1 cell gives

$$\mathrm{Var}(V(S) \mid Z_1) = \mathbb{E}[\mathrm{Var}(V(S) \mid Z_2) \mid Z_1] + \mathrm{Var}(\mathbb{E}[V(S) \mid Z_2] \mid Z_1). \tag{38}$$

Taking expectations and using the nonnegativity of the second term yields

$$\mathcal{H}(\phi_1) - \mathcal{H}(\phi_2) = \mathbb{E}[\mathrm{Var}(\mathbb{E}[V(S) \mid Z_2] \mid Z_1)] \geq 0. \tag{39}$$

This proves the aliasing monotonicity in Theorem 1. In particular, $\mathcal{H}(\phi_{\mathrm{id}}) = 0$, although identity groups can have poor finite-batch occupancy.

To prove the MSE identity, draw an independent target $S_0 \sim p(S \mid z)$. Draw m conditionally independent peer returns $R_1, \ldots, R_m \sim p(R \mid z)$ from the same cell. Define their mean as $\widehat{V}_z^{(m)} = m^{-1} \sum_{\ell=1}^{m} R_\ell$. The estimation error decomposes as

$$\widehat{V}_z^{(m)} - V(S_0) = (\widehat{V}_z^{(m)} - \bar{V}_z) + (\bar{V}_z - V(S_0)). \tag{40}$$

Both error terms have zero conditional mean given z. Their conditional covariance is zero because the peer samples are independent of the target state. Therefore,

$$\begin{aligned}
&\mathbb{E}[(\widehat{V}_z^{(m)} - V(S_0))^2 \mid z] \\
&= \mathbb{E}[(\bar{V}_z - V(S_0))^2 \mid z] + \mathbb{E}[(\widehat{V}_z^{(m)} - \bar{V}_z)^2 \mid z] \\
&= H_z + \mathrm{Var}(\widehat{V}_z^{(m)} \mid z).
\end{aligned} \tag{41}$$

The estimator $\widehat{V}_z^{(m)}$ is a mean of conditionally independent peers. Its variance is

$$\begin{aligned}
\mathrm{Var}(\widehat{V}_z^{(m)} \mid z) &= \frac{1}{m} \mathrm{Var}(R \mid z) \\
&= \frac{1}{m} \left[\mathrm{Var}(V(S) \mid z) + \mathbb{E}[\mathrm{Var}(R \mid S) \mid z]\right] \\
&= \frac{H_z + \bar{\sigma}_z^2}{m},
\end{aligned} \tag{42}$$

where the second line is the law of total variance. Hence,

$$\mathbb{E}[(\widehat{V}_z^{(m)} - V(S_0))^2 \mid z] = H_z + \frac{H_z + \bar{\sigma}_z^2}{m}. \tag{43}$$

This is Eq. (11) and completes the theorem. A leave-one-out baseline for a realized cell of size n_z uses $m = n_z - 1$. A fresh-target analysis of a mean formed from all n_z peers uses $m = n_z$. For fixed peer states $s_1, \ldots, s_m$ and a fixed target s_0, independent zero-mean return noise instead gives

$$\mathbb{E}[(\widehat{V}_z^{(m)} - V(s_0))^2 \mid s_0, s_1, \ldots, s_m] = \left(\frac{1}{m} \sum_{\ell=1}^{m} V(s_\ell) - V(s_0)\right)^2 + \frac{1}{m^2} \sum_{\ell=1}^{m} \sigma^2(s_\ell). \tag{44}$$

Here the first term is the fixed-design approximation error, rather than the population quantity H_z. Both designs expose an approximation–estimation trade-off:

- Raw-state grouping can reduce H_z. When $n_z \approx 1$, few peers remain for baseline estimation. Eq. (26) also predicts little or no centered policy-gradient signal in this case.
- Pooling steps by trajectory can increase n_z. It mixes different progress phases and feedback modes and can increase H_z.
- ASGPO uses feedback and progress to limit the comparisons while preserving enough peers for signal retention and baseline estimation.

Conditional refinement. The optional refinement rule in Eq. (12) trades lower value heterogeneity for fewer peers per subgroup. Fix a base bucket b with n_b samples. Conditional refinement splits it into clusters $\mathcal{C}_{b,1}, \ldots, \mathcal{C}_{b,k_b}$ with centroids $m_{b,c}$ in the hashed context space. Assume that the continuation value factors through the embedding as $V(s) = v(\psi(s))$ and that v is L-Lipschitz on the convex hull of the bucket:

$$|v(x) - v(x')| \le L\|x - x'\|_2. \tag{45}$$

For a cluster $\mathcal{C}$ with centroid $m_{\mathcal{C}}$ and mean value $\bar{V}_{\mathcal{C}}$,

$$\mathrm{Var}(V(s) \mid \mathcal{C}) = \mathbb{E}[(V(s) - \bar{V}_{\mathcal{C}})^2 \mid \mathcal{C}]. \tag{46}$$

For any constant c_0, variance is no larger than mean squared error around c_0. Choosing $c_0 = v(m_{\mathcal{C}})$ gives

$$\begin{aligned}\mathbb{E}[(V(s) - \bar{V}_{\mathcal{C}})^2 \mid \mathcal{C}] &\le \mathbb{E}[(v(\psi(s)) - v(m_{\mathcal{C}}))^2 \mid \mathcal{C}] \\ &\le L^2 \mathbb{E}[\|\psi(s) - m_{\mathcal{C}}\|_2^2 \mid \mathcal{C}].\end{aligned} \tag{47}$$

Weighting Eq. (47) by cluster sizes gives the average abstraction error after refinement:

$$\begin{aligned}B_b^2(k_b) &:= \frac{1}{n_b} \sum_{c, i \in \mathcal{C}_{b,c}} (V(s_i) - \bar{V}_{b,c})^2 \\ &\le \frac{L^2}{n_b} \sum_{c, i \in \mathcal{C}_{b,c}} \|\psi(s_i) - m_{b,c}\|_2^2.\end{aligned} \tag{48}$$

Here $\sum_{c, i \in \mathcal{C}_{b,c}}$ abbreviates $\sum_{c=1}^{k_b} \sum_{i \in \mathcal{C}_{b,c}}$. This is the distortion bound for the optional refinement: within-cluster value heterogeneity is bounded by the representation's k-means distortion.

The estimation-variance term has the opposite dependence on k_b. If cluster c has size $n_{b,c}$ and return-noise variance at most σ_b^2, then the average sample-mean variance over a uniformly chosen step in bucket b is bounded above by

$$\frac{1}{n_b} \sum_{c=1}^{k_b} \sum_{i \in \mathcal{C}_{b,c}} \frac{\sigma_b^2}{n_{b,c}} = \frac{1}{n_b} \sum_{c=1}^{k_b} n_{b,c} \frac{\sigma_b^2}{n_{b,c}} = \sigma_b^2 \frac{k_b}{n_b}. \tag{49}$$

Substituting the optional refinement rule $k_b \approx \lambda n_b^{\gamma}$ gives

$$\sigma_b^2 \frac{k_b}{n_b} \approx \sigma_b^2 \lambda n_b^{\gamma - 1}. \tag{50}$$

For $\gamma < 1$, this term decreases as the base bucket accumulates more steps. In contrast, a raw-state partition behaves like $k_b \approx n_b$. It keeps $\sigma_b^2 k_b / n_b = O(\sigma_b^2)$ and drives the leave-in retention factor toward zero. A single coarse trajectory-style bucket has small k_b but can have large distortion and large H_z. The optional refinement rule lies between these extremes. Its cluster count grows sublinearly, allowing group sizes to increase with data. Deployed ASGPO instead uses the unrefined feedback–progress key.

Comparison with GiGPO-style repeated-state grouping. We compare concrete-state identity with the deployed abstract map. Let $\phi_{\mathrm{G}} = \phi_{\mathrm{id}}$ be the repeated-state map used by GiGPO: two steps share a group only when their concrete states are identical. Let $\phi_{\mathrm{A}} = \alpha$ be deployed ASGPO. The identity $\alpha = \alpha \circ \phi_{\mathrm{id}}$ shows that the exact-state map refines ASGPO. Every GiGPO cell is contained in one ASGPO cell. One ASGPO cell may contain several exact states with similar progress and feedback context.

The signal-retention comparison is immediate from Eq. (9). For any realized batch partition,

$$\mathcal{S}_N(\phi_{\mathrm{A}}) - \mathcal{S}_N(\phi_{\mathrm{G}}) = \left(1 - \frac{|\mathcal{G}_{\mathrm{A}}|}{N}\right) - \left(1 - \frac{|\mathcal{G}_{\mathrm{G}}|}{N}\right) = \frac{|\mathcal{G}_{\mathrm{G}}| - |\mathcal{G}_{\mathrm{A}}|}{N}. \tag{51}$$

ASGPO therefore has a strictly larger finite-batch retention factor whenever it uses fewer nonempty groups. This ordering is exact for the realized batch. If repeated-state grouping produces only singletons, then $|\mathcal{G}_{\mathrm{G}}| = N$ and $\mathcal{S}_N(\phi_{\mathrm{G}}) = 0$. For mean group size $m = N/|\mathcal{G}_{\mathrm{A}}| > 1$, ASGPO's retention factor is exactly $1 - 1/m$.

The MSE comparison also accounts for value heterogeneity introduced by merging. Let $G \subseteq A$ be representative exact and abstract cells, with $n_A > n_G$ independent peers. Each MSE below averages over a fresh target drawn from its cell, as in Eq. (43). This is not a pointwise risk comparison for every fixed state in G. Suppose exact repeated-state grouping has negligible value heterogeneity inside G:

$$H_G = \mathrm{Var}(V(S) \mid G) = 0, \tag{52}$$

which is the most favorable case for repeated-state baseline estimation. Assume the same average return-noise variance in both groups, $\bar{\sigma}_A^2 = \bar{\sigma}_G^2 = \bar{\sigma}^2$. Let ASGPO's extra abstraction error be

$$\delta_A^2 = H_A = \mathrm{Var}(V(S) \mid A). \tag{53}$$

Using the random-design MSE formula from Eq. (43), the GiGPO baseline error is

$$\mathrm{MSE}_{\mathrm{G}} = H_G + \frac{H_G + \bar{\sigma}^2}{n_G} = \frac{\bar{\sigma}^2}{n_G}, \tag{54}$$

while the ASGPO baseline error is

$$\mathrm{MSE}_{\mathrm{A}} = \delta_A^2 + \frac{\delta_A^2 + \bar{\sigma}^2}{n_A}. \tag{55}$$

The abstract cell has no larger cell-averaged baseline MSE when

$$\begin{aligned} \delta_A^2 + \frac{\delta_A^2 + \bar{\sigma}^2}{n_A} &\le \frac{\bar{\sigma}^2}{n_G} \\ \delta_A^2 \left(1 + \frac{1}{n_A}\right) &\le \bar{\sigma}^2 \left(\frac{1}{n_G} - \frac{1}{n_A}\right) \\ \delta_A^2 &\le \bar{\sigma}^2 \, \frac{n_A - n_G}{n_G(n_A + 1)}. \end{aligned} \tag{56}$$

This condition clarifies when abstraction is beneficial. If GiGPO already has many exact repeats (n_G large and close to n_A), the allowable abstraction error is small, so exact repeated-state grouping can be competitive. If GiGPO mostly forms singletons ($n_G = 1$), the condition becomes

$$\delta_A^2 \le \bar{\sigma}^2 \frac{n_A - 1}{n_A + 1}, \tag{57}$$

which quantifies the heterogeneity tolerated when replacing a one-sample baseline with an n_A-sample baseline. The associated non-singleton groups also permit nonzero centered advantages when their returns differ.

These results give two complementary comparisons. Coarsening to fewer nonempty groups strictly increases the finite-batch retention factor. The cell-averaged baseline MSE is no larger under Eq. (56) and is strictly lower when the inequality is strict. For steps from the same trajectory, $\Delta_i(\mathcal{F})$ accounts for cross-step score–return correlations. The empirical diagnostics support the occupancy part of this comparison. Raw-state grouping forms more groups and a higher fraction of singleton groups than the earlier refined ASGPO variant on the logged terminal-agent batches.

Scope of the theoretical comparison. The analysis separates the occupancy benefit from the baseline-estimation trade-off. The expected centered-signal comparison requires the stated conditional action law and return independence for every other peer in the group. For general trajectory batches, the comparison retains the cross-step residual. The baseline comparison depends on continuation-value heterogeneity within the abstraction. Fewer comparison cells increase the exact occupancy factor. Eq. (56) specifies the heterogeneity allowed for a lower cell-averaged MSE under the stated sampling and noise assumptions. Section F evaluates the fragmentation mechanism through within-group and between-group variance, singleton rates, and zero-advantage diagnostics.

D Additional Ablations

Table 3 includes the component rows that have completed the de-noised protocol. The reported sweep covers anchor coordinates and refinement. It also tests grouping and progress returns. Exploratory single-rollout runs of other settings are excluded from the four-rollout plateau tables.

E Run Log Mapping

The 4B run identifiers are listed below. Here `LOO` denotes the leave-one-component-out sweep, not a leave-one-out baseline. The historical "error bucket" name refers to the feedback coordinate η.

- Simplified ASGPO, no conditional refinement: ablation-LOO-cond-0629-010639.
- No progress bucket: ablation-LOO-progress-0628-115338.
- No error bucket: ablation-LOO-error-0627-222412.
- No action phase: ablation-LOO-phase-0627-082337.
- Original refined ASGPO: `full_ASG` reference row in the LOO denoise log.

F Mechanism: How Abstract Grouping Reshapes the Baseline

F.1 What A Good Grouping Must Do

GRPO estimates advantages using group-return statistics instead of a learned critic (Eq. (1)). A useful grouping must satisfy two competing requirements. Groups must be large enough to estimate a stable baseline. In the singleton limit, $\mu_g = R_i$ and the standardized advantage is exactly zero, so the centered step-level contribution vanishes. Groups must also be homogeneous enough that the group mean is a fair reference for each member.

We diagnose grouping quality using training-batch statistics. Group counts, average size, and singleton rates measure occupancy. Base-bucket counts and compression ratios track refinement. We also record within-group return variance, between-group return variance, and their ratio. These metrics are computed on the same step returns that GRPO standardizes after the ASGPO progress-aware return construction.

F.2 ASGPO Reduces State Fragmentation

The earlier refined ASGPO variant forms larger comparison groups than literal raw-state matching (Table 4). Both runs share the SFT base, progress-aware returns, and optimizer. Their training setup is also matched. Over the 49 logged updates, raw-state grouping forms 105.7 ± 62.4 groups per update, with 73.0% singleton groups and mean group size 2.5. Refined ASGPO forms 9.6 ± 3.4 groups, with mean group size 23.4 and only 8.2% singleton groups.

The logged zero-advantage step fraction falls from 31.2% for raw-state grouping to 0.4% for refined ASGPO. This step-level metric differs from the group-level singleton rate: non-singleton groups can also have equal returns and zero advantages. These metrics show that abstraction reduces fragmentation and recovers usable step-level comparisons in this run.

F.3 Interpreting Variance Under Fragmentation

Small fragmented groups can produce a high separation ratio. For raw-state grouping, this ratio is approximately 100, compared with 0.67 for refined ASGPO. However, this ratio must be read alongside the group sizes. Its within-group term excludes singletons, and the remaining small raw-state groups have little return variation. The means of its many groups span a wider return range, producing a large between/within ratio. Many of these steps still have zero centered advantage. Refined ASGPO forms larger groups and has a lower logged zero-advantage fraction. The separation ratio therefore needs to be interpreted alongside occupancy and zero-advantage rates.

F.4 Refinement Ablation

The earlier variant subdivides abstract groups using Eq. (13). Its original 4B run has an average compression ratio of 1.43: roughly six to seven base buckets become about ten final groups. Refinement was therefore active in this run. The component sweep tests whether this extra subdivision improves success. Removing refinement raises the observed de-noised plateau mean from 0.257 to 0.314 on the same held-out split. This result supports the simpler default anchor $\alpha(s) = (\eta(\tilde{s}), \rho(s))$, which eliminates clustering.

F.5 Component Sweep

Table 3 summarizes the 4B component sweep. Each row is trained with the same SFT base and schedule, then re-evaluated with four validation rollouts per task. The primary rows use checkpoints 50–80. Partial diagnostic controls are marked in Table 3. Removing either anchor coordinate lowers mean success. The progress-bucket ablation gives 0.219 without ρ, compared with 0.245 without feedback η. The larger drop without progress supports separating interaction stages as well as feedback categories. Removing conditional refinement gives the highest observed mean, 0.314, motivating the simpler anchor in Eq. (2).

Normalized exact-state grouping and random same-size grouping also yield lower means. These controls test normalization and group size as alternative explanations. Their settings differ from the primary comparisons. The normalized-exact control uses sparse returns. The random control covers checkpoints 50–70 rather than 50–80.

Task-level paired bootstrap supports the gains over matched Exact and the no-grouping ablation. The simplified ASGPO variant significantly outperforms matched Exact ($+0.110$, 95% CI $[0.048, 0.176]$) and the no-grouping ablation ($+0.126$, 95% CI $[0.057, 0.195]$). Removing refinement raises the observed mean by 0.057 and eliminates clustering, although the paired interval on this 73-task split includes zero (95% CI $[-0.005, 0.121]$).