A TWO-PLAYER AGENTIC GAME FOR VERIFIABLE ASSURANCE REASONING AND INTERACTIVE TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

In structured decision-making workflows such as form filling, compliance checking, and maintenance reporting, LLM outputs must be locally correct, globally consistent, and auditable against task-specific rules. Existing refinement methods often rely on heuristic debate, self-play, or LLM-generated supervision, creating a second-order assurance problem. We propose **DPA-GRPO** (Dual Paired-Action Group-Relative Policy Optimization), a paired-action training method for a two-player generator–verifier game with structured verifier interventions. The generator proposes outputs and may revise them when challenged; the verifier either remains silent or raises a safety assurance case (SAC) containing a claim, argument, and evidence. These SAC/no-SAC and KEEP/REVISE decisions induce paired counterfactual action groups, which DPA-GRPO uses for role-specific KL-regularized GRPO updates. We analyze the unregularized game and show that positive probability on strictly lower-reward intervention or revision actions creates a profitable unilateral deviation. Under standard stochastic-approximation assumptions, DPA-GRPO tracks the corresponding game ODE, whose isolated asymptotically stable limit points are stationary and candidate local equilibria under role-wise local optimality. Experiments on TaxCalcBench TY24 and FinQA show that DPA-GRPO improves structured decision accuracy over zero-shot generation and generator-only RL, with additional TaxCalcBench controls against static paired-training baselines. Training increases correct silent acceptance, reduces missed errors, and improves revision behavior.

1 INTRODUCTION

Large language models (LLMs) have achieved strong performance on open-ended generation, reasoning, and tool-use tasks (Brown et al., 2020; Achiam et al., 2023; Wei et al., 2022; Yao et al., 2022; Schick et al., 2023), yet their use in complex, structured decision-making workflows remains brittle. In domains such as form filling, compliance checking, maintenance reporting, and program-like reasoning, LLM outputs must be plausible, locally correct, globally consistent, and auditable against required rules and standards (Ji et al., 2023; Huang et al., 2025; Cobbe et al., 2021; Lightman et al., 2023).

Recent progress highlights useful ingredients for this problem and leaves open how to train verifier intervention and generator revision policies around structured assurance evidence. First, using LLMs to directly verify or supervise other LLMs creates a second-order verification problem in which the verifier itself requires assessment. A verifier rewarded for finding errors may produce persuasive objections with weak grounding (Ji et al., 2023; Huang et al., 2025; Bai et al., 2022; Madaan et al., 2023; Shinn et al., 2023). A generator that follows verifier feedback uncritically may over-correct accurate outputs. A generator that discounts useful feedback may preserve unsafe errors. Second, multi-agent debate improves reasoning by allowing model instances to challenge one another. Existing implementations often use debate as a prompting or test-time scaling mechanism rather than as a role-specific training paradigm (Du et al., 2024; Khan et al., 2024), without a systematic framework to co-train or co-adapt all agents in concert (Cobbe et al., 2021; Lightman et al., 2023; Lifshitz et al., 2025; Jung et al., 2026; Yu et al., 2025a). Finally, recent work on self-play and competitive evaluation methods shows promise for learning from game outcomes and inter-model competition, and

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.

leave open safety-case-style assurance with respect to specific requirements and auditability (Kuba et al., 2025; Alyahya et al., 2025; Liu et al., 2025; Liao et al., 2024). The challenges to get AI agents to agree are highlighted in (Berdoz et al., 2026).

We propose **DPA-GRPO** (Dual Paired-Action Group-Relative Policy Optimization), a game-theoretic solution for safety-case-based assurance of structured LLM outputs. We train a two-player Markov game (Shapley, 1953) between a *generator* and a *verifier*. The generator proposes solutions (e.g., form filling or multi-step planning) and may revise them when challenged. The verifier observes the proposed solutions and chooses whether to remain silent or raise a safety assurance case (SAC) consisting of a claim, argument, and evidence. Safety-critical engineering has long used *safety cases*: structured claim-argument-evidence records that justify why a system satisfies required properties and is sufficiently safe for deployment (Kelly & Weaver, 2004; Denney et al., 2015; Dezfuli et al., 2015; Lincoln, 2012). Such records are attractive for LLM validation because they make objections interpretable: the verifier states a claim, supports it with an argument, and grounds the argument in evidence. A more detailed discussion of related work is provided in Appendix A.

The generator-verifier game turns structured assurance feedback into paired intervention and revision decisions, producing paired counterfactual training signals at each evaluated output. In particular, SAC/no-SAC decisions and keep/revise responses arising from the game produce novel positive/negative samples that do not exist in standard training regimes, such as SFT, DPO, or GRPO (Ouyang et al., 2022; Rafailov et al., 2023; Shao et al., 2024). We analyze this game and categorize the counterfactual training signals into eight informative cases. The resulting paired-action updates are related to recent interpretations of GRPO as a contrastive or preference-like objective (Shao et al., 2024; Wu et al., 2025), but differ in that the preference pairs are induced by structured generator–verifier interactions rather than external preference labels. Both agents are trained with role-conditional policies and KL-regularized objectives, allowing the system to improve through interaction while limiting unstable policy drift.

Our theoretical analysis studies necessary conditions for local Nash equilibria in the two-player Markov game (Nash Jr, 1950; Shapley, 1953). We show that under standard non-degeneracy assumptions, positive probability mass on lower-reward cases, such as missed violations or harmful unnecessary revisions, creates a profitable unilateral deviation in the unregularized game. Under the stochastic-approximation assumptions, the interpolated DPA-GRPO iterates form an asymptotic pseudo-trajectory of the game ODE. Consequently, every isolated asymptotically stable limit point is stationary for the game vector field. A local Nash interpretation additionally requires role-wise local optimality conditions.

Our main contributions are as follows: First, we formulate structured LLM validation as a two-player Markov game between a generator and a verifier, where verifier interventions are expressed as structured SACs containing claims, arguments, and evidence reasoning. Second, we propose DPA-GRPO, a dual paired-action training method that constructs counterfactual SAC/no-SAC and keep/revise comparisons to train both agents with role-conditional policies and KL-regularized updates. Third, we develop an interpretable case taxonomy and connect it to local Nash conditions and stochastic-approximation dynamics. Fourth, we evaluate DPA-GRPO on TaxCalcBench TY24 and FinQA against zero-shot generation and generator-only reinforcement learning, with additional paired supervised and preference-learning controls on TaxCalcBench.

2 A Generator–Verifier Markov Game

We model structured output generation as a finite-horizon two-player Markov game between a generator G and a verifier V. A task instance $\tau \sim \mathcal{D}$ contains the structured input, task rules, output schema, and any dependencies required to complete the output. For each task τ, the game unfolds over a finite horizon $t = 1, \ldots, T_\tau$. At each time step t, the players complete one structured decision unit u_t. The state m_t contains the task instance, the current decision unit, and the partial output generated so far: $m_t = (\tau, u_t, o_{<t})$.

The two players have role-specific policies and action spaces. The generator first produces an initial proposal for the current decision unit, $x_t \sim \pi_G^x(\cdot \mid m_t)$. The verifier observes the current state and proposal, (m_t, x_t), and chooses an intervention action $y_t \sim \pi_V(\cdot \mid m_t, x_t)$, where $y_t \in \{\mathrm{SAC}, \mathrm{NS}\}$.

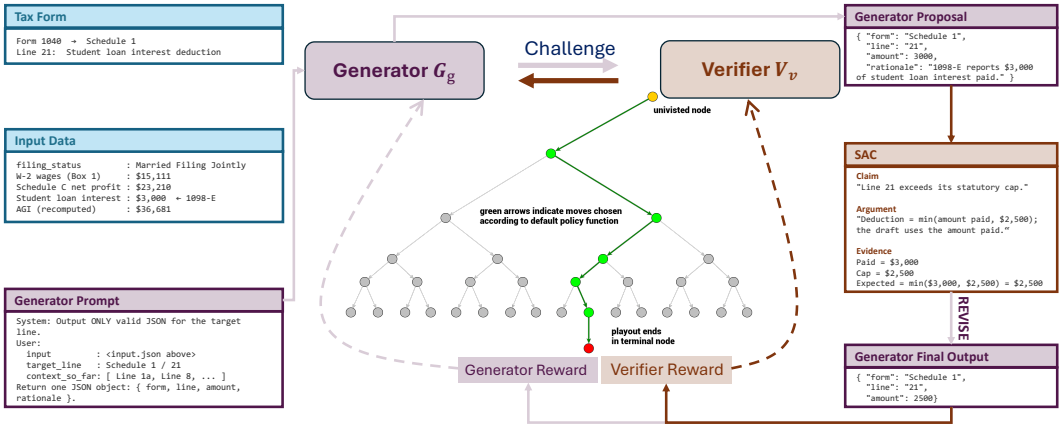


Figure 1: **Illustration of the generator–verifier game in a tax-form completion task.** The left panel shows the application context for one evaluated decision unit, including the relevant form text, structured input data, and generator prompt. The generator first proposes an output value, after which the verifier may challenge the proposal by producing a structured safety assurance case (SAC) with a claim, argument, and evidence. The generator then decides whether to keep the original proposal or revise it. This interaction produces role-specific rewards for both agents, which DPA-GRPO uses to construct paired counterfactual training signals.

Here, SAC denotes a structured safety assurance case and NS denotes no-SAC. An SAC may include a claim, supporting argument, evidence, and a suggested correction.

If the verifier chooses NS, the proposal is submitted directly, so $o_t = x_t$. If the verifier chooses SAC, the generator observes the verifier intervention and chooses how to respond. It may generate a candidate revision $z_t \sim \pi_G^z(\cdot \mid m_t, x_t, y_t)$ and then select a revision action $a_t \sim \pi_G^a(\cdot \mid m_t, x_t, y_t, z_t)$, where $a_t \in \{\text{KEEP}, \text{REVISE}\}$. If $a_t = \text{KEEP}$, the original proposal is submitted ($o_t = x_t$); if $a_t = \text{REVISE}$, the revised proposal is submitted ($o_t = z_t$). The transition appends the submitted output component to the partial output and advances to the next decision unit: $m_{t+1} = T(m_t, x_t, y_t, a_t, o_t)$. The episode terminates after all T_τ decision units have been completed.

The game is general-sum: both players are evaluated against output correctness but receive role-specific rewards. DPA-GRPO exploits the binary verifier decision $\{\text{SAC}, \text{NS}\}$ and, on SAC branches, the generator decision $\{\text{KEEP}, \text{REVISE}\}$ to construct paired counterfactual training signals.

2.1 Game and SAC Taxonomy

We now define what is contained in a SAC and how SAC outcomes are used to diagnose training behavior.

Safety Assurance Case (SAC). When $y_t = \text{SAC}$, the verifier emits $\text{SAC}_t = (c_t, g_t, e_t)$, where c_t is the claim that the initial proposal is incorrect or requires review, g_t is the argument explaining why the claim follows from the task context and applicable rule, and e_t is supporting evidence such as relevant input fields, intermediate computations, or dependencies. When $y_t = \text{NS}$, the SAC field is empty. Figure 1 illustrates this structure in a complete generator–verifier interaction. To examine whether this structured representation behaves differently from free-form verifier feedback, we report a matched comparison with unstructured critique in Appendix E.6.

Case taxonomy. We use case taxonomy to diagnose behavior and connect training dynamics to the equilibrium conditions in Section 3. Let $S_x = \mathbf{1}\{x_t \text{ is strictly correct}\}$ and $S_z = \mathbf{1}\{z_t \text{ is strictly correct}\}$. Each transition is classified by initial proposal correctness S_x, verifier de-

cision y_t, generator action a_t, and revision correctness S_z:

Case	S_x	y_t	a_t	S_z	Interpretation
1	1	NS	–	–	correct initial proposal accepted
2	1	SAC	REVISE	–	false-positive SAC, generator revises
3	1	SAC	KEEP	–	false-positive SAC, generator keeps
4	0	NS	–	–	missed initial proposal error
5a	0	SAC	REVISE	1	successful repair
5b	0	SAC	REVISE	0	failed repair accepted
6a	0	SAC	KEEP	0	bad revision rejected
6b	0	SAC	KEEP	1	good revision rejected

Cases 1 and 5a represent desired outcomes, while Case 4 captures under-intervention. Cases 2–3 diagnose over-intervention, and Cases 5b–6b characterize generator revision-selection behavior. Detailed generator–verifier interactions are provided in Appendix G.

2.2 Paired Rewards and Regularized Objectives

During training, a reward oracle provides correctness labels. Let $S_x = \mathbf{1}\{x_t \text{ is strictly correct}\}$ and $S_z = \mathbf{1}\{z_t \text{ is strictly correct}\}$. We define the SAC-correctness label as $c_{\text{sac}} = 1 - S_x$, so SAC is correct if and only if the initial proposal is incorrect. The oracle is used only for training-time reward computation and diagnostics; its labels are not provided as inputs to either policy and are not required at inference time. In the experiments, the oracle is programmatic and deterministic up to benchmark-specific numerical tolerances. We discuss the effect of noisy and partial supervision, including a simple pairwise preference noise analysis, in Appendix F.

The verifier receives paired intervention rewards

$$R_V(\text{SAC} \mid m_t, x_t) = c_{\text{sac}}, \qquad R_V(\text{NS} \mid m_t, x_t) = 1 - c_{\text{sac}}.$$

On the SAC branch, the generator compares KEEP and REVISE using

$$R_G(\text{KEEP} \mid m_t, x_t, \text{SAC}) = S_x, \qquad R_G(\text{REVISE} \mid m_t, x_t, \text{SAC}) = S_z.$$

Let $\widetilde{J}_f(\phi;\theta) = \mathbb{E}_{\tau\sim d^{\phi,\theta}}[R_f(\tau)]$ and $\widetilde{J}_v(\theta;\phi) = \mathbb{E}_{\tau\sim d^{\phi,\theta}}[R_v(\tau)]$ denote the unregularized expected role rewards. The KL-regularized objectives are

$$J_v(\theta;\phi) = \widetilde{J}_v(\theta;\phi) - \beta_v\, \mathbb{E}_{I_v}\left[\mathrm{KL}(\pi_{v,\theta}(\cdot \mid I_v) \,\|\, \pi_{v,\text{ref}}(\cdot \mid I_v))\right],$$

and

$$\begin{aligned} J_f(\phi;\theta) = \widetilde{J}_f(\phi;\theta) &- \beta_{fx}\, \mathbb{E}_{I_f^x}\left[\mathrm{KL}\big(\pi^x_{f,\phi}(\cdot \mid I_f^x) \,\|\, \pi^x_{f,\text{ref}}(\cdot \mid I_f^x)\big)\right] \\ &- \beta_{fz}\, \mathbb{E}_{I_f^z}\left[\mathrm{KL}\big(\pi^z_{f,\phi}(\cdot \mid I_f^z) \,\|\, \pi^z_{f,\text{ref}}(\cdot \mid I_f^z)\big)\right] \\ &- \beta_{fa}\, \mathbb{E}_{I_f^a}\left[\mathrm{KL}\big(\pi^a_{f,\phi}(\cdot \mid I_f^a) \,\|\, \pi^a_{f,\text{ref}}(\cdot \mid I_f^a)\big)\right]. \end{aligned}$$

The KL terms anchor both role policies to their respective reference policies, limiting policy drift during joint optimization; their empirical contribution is examined in Section 4.3. They are also incorporated into the equilibrium analysis in Section 3.

2.3 Dual Paired-Action GRPO

The central implementation choice is that the verifier decision and generator-on-SAC decision are binary and counterfactually scorable. For the verifier, the action group is $G_V(m_t, x_t) = \{\text{SAC}, \text{NS}\}$. For the generator on the SAC branch, the action group is $G_G(m_t, x_t, y_t, z_t) = \{\text{KEEP}, \text{REVISE}\}$.

For a context I and binary group $G(I) = \{a_1, a_2\}$, we compute rewards $R(a_1)$ and $R(a_2)$ and form the group-normalized advantage

$$A(a_i) = \frac{R(a_i) - \bar{R}}{\sqrt{\frac{1}{2}\sum_{j=1}^{2}(R(a_j) - \bar{R})^2 + \epsilon_A}}, \qquad \bar{R} = \frac{1}{2}(R(a_1) + R(a_2)).$$

With two actions this reduces to a contrastive pressure: the higher-reward action receives positive advantage and the other receives negative advantage of equal magnitude.

Each role's update is an on-policy policy-gradient step with a KL anchor against a frozen reference policy. For role policy π_ψ with reference $\pi_{\psi,\mathrm{ref}}$, the paired-action loss is

$$\mathcal{L}_{\mathrm{pair}}(\psi) = \mathbb{E}[-A(a)\,\log \pi_\psi(a \mid I) + \beta(\log \pi_\psi(a \mid I) - \log \pi_{\psi,\mathrm{ref}}(a \mid I))]\,.$$

The verifier update uses $\psi = \theta$ and $G = G_V$. On SAC branches, the generator receives two updates. First, candidate revision texts are optimized with GRPO: the K sampled revisions form a group, with each revision receiving correctness reward S_z. Second, the generator's revision-action policy is optimized over the paired group $G_G = \{\mathrm{KEEP}, \mathrm{REVISE}\}$ using rewards S_x and S_z, respectively. Both updates use the same generator LoRA parameters and KL anchoring.

At each iteration, the current generator and verifier produce a fresh batch of decision-unit interactions, which is used for the GRPO updates described above and then discarded. The complete DPA-GRPO training procedure is given in Algorithm 1 in Appendix C.

3 Theoretical Guarantees

We next connect the implemented rewards to local equilibrium behavior and convergence.

Local Nash equilibrium. A pair (ϕ^*, θ^*) is a local Nash equilibrium if there exists a neighborhood $\mathcal{N}$ such that $J_f(\phi^*; \theta^*) \geq J_f(\phi; \theta^*) \quad \forall \phi \in \mathcal{N}, \quad J_v(\theta^*; \phi^*) \geq J_v(\theta; \phi^*) \quad \forall \theta \in \mathcal{N}$. At an interior local Nash equilibrium, the necessary first-order conditions are $\nabla_\phi J_f(\phi^*; \theta^*) = 0, \quad \nabla_\theta J_v(\theta^*; \phi^*) = 0$. These are each player's own gradients; cross-gradients describe coupling but are not Nash stationarity conditions.

Assumption 1 (Local non-degeneracy)**.** For each binary role decision, whenever a policy assigns nonzero probability to both actions on a positive-measure set of contexts, there exists a local parameter direction that transfers an arbitrarily small amount of probability mass from one action to the other on that set.

Lemma 1 (Verifier pointwise best response)**.** *Fix a context (m_t, x_t). Under the verifier rewards $R_v(\mathrm{SAC} \mid m_t, x_t) = c_{\mathrm{sac}}, \quad R_v(\mathrm{NS} \mid m_t, x_t) = 1 - c_{\mathrm{sac}}$, the unique pointwise reward-maximizing verifier action is*

$$y^*(m_t, x_t) = \begin{cases} \mathrm{SAC}, & c_{\mathrm{sac}} = 1, \\ \mathrm{NS}, & c_{\mathrm{sac}} = 0. \end{cases}$$

Consequently, if the verifier places nonzero probability on the opposite action on a positive-measure set of contexts, then there exists a local direction $\Delta\theta$ such that $D_\theta \widetilde{J}_v(\theta; \phi)[\Delta\theta] > 0$.

Lemma 2 (Generator best response on the SAC branch)**.** *On the SAC branch, define $\bar{S}_z(m_t, x_t) = \mathbb{E}[S_z \mid m_t, x_t, \mathrm{SAC}, a_t = \mathrm{REVISE}]$. KEEP has reward S_x, while REVISE has conditional expected reward $\bar{S}_z(m_t, x_t)$. Therefore, if $S_x > \bar{S}_z(m_t, x_t)$, KEEP strictly dominates REVISE; if $S_x < \bar{S}_z(m_t, x_t)$, REVISE strictly dominates KEEP. Consequently, whenever the generator places nonzero probability on the strictly dominated action on a positive-measure set of SAC contexts, there exists a local direction $\Delta\phi$ such that $D_\phi \widetilde{J}_f(\phi; \theta)[\Delta\phi] > 0$.*

Theorem 2 (KL-regularized paired-action best response)**.** *Suppose (ϕ^*, θ^*) is a local Nash equilibrium of the KL-regularized game. Under Assumption 1 and the KL-dominance condition stated in Appendix D.1, the verifier assigns zero probability to pointwise reward-suboptimal SAC/no-SAC actions on every positive-measure set of contexts, and the generator assigns zero probability to strictly suboptimal KEEP/REVISE actions on every positive-measure set of SAC contexts.*

This condition does not imply concentration on Case 1: successful SAC-and-revise behavior (Case 5a) can remain reward-consistent. A stronger Case 1-favoring result therefore requires an additional intervention or revision cost.

Convergence of alternating GRPO. Let $F(\phi, \theta) = (\nabla_\phi J_f(\phi; \theta), \nabla_\theta J_v(\theta; \phi))$ denote the game vector field. The alternating GRPO updates can be viewed as a stochastic approximation to the ODE $\dot{\phi} = \nabla_\phi J_f(\phi; \theta), \quad \dot{\theta} = \nabla_\theta J_v(\theta; \phi)$.

Theorem 3 (Stochastic-approximation tracking)**.** *Under the stochastic-approximation assumptions in Appendix D.2, the interpolated alternating-GRPO iterates form an asymptotic pseudo-trajectory of the game ODE. Consequently, every isolated asymptotically stable limit point satisfies* $\nabla_\phi J_f(\phi;\theta) = 0, \quad \nabla_\theta J_v(\theta;\phi) = 0$, *and is therefore stationary for the game vector field. A local Nash interpretation additionally requires role-wise local optimality conditions.*

4 EXPERIMENTS

Table 1: Main held-out results on TaxCalcBench TY24 and FinQA. Panel (a) reports results on the fixed-seed TaxCalcBench 80/20 split. Panel (b) reports results on a fixed 100-example subset of the official FinQA test split. The case-taxonomy columns follow Section 2.1: Case 1 + Case 3 captures correct initial outputs retained after the generator–verifier interaction, Case 4 captures missed errors, and Case 6a captures rejection of bad candidate revisions. For generator-only methods, Case 1 + Case 3 reduces to correct generator outputs, Case 4 to incorrect generator outputs, and Case 6a is not applicable.

Method	**Backbone**	**Case taxonomy**			**Test acc.↑**
		Case 1 + 3↑	**Case 4↓**	**Case 6a↑**	
(a) TaxCalcBench TY24					
Untrained					
Zero-shot	Qwen3-4B	0.41 + 0	0.59	–	0.41
Zero-shot	Qwen3-8B	0.53 + 0	0.47	–	0.53
Generator-only RL					
GRPO	Qwen3-4B	0.52 + 0	0.48	–	0.52
DAPO	Qwen3-4B	0.47 + 0	0.53	–	0.47
GSPO	Qwen3-4B	0.46 + 0	0.54	–	0.46
GRPO	Qwen3-8B	0.54 + 0	0.46	–	0.54
DAPO	Qwen3-8B	0.55 + 0	0.45	–	0.55
GSPO	Qwen3-8B	0.52 + 0	0.48	–	0.52
Ours					
DPA-GRPO	Qwen3-4B	0.52 + 0.01	**0.40**	**0.06**	**0.53**
DPA-GRPO	Qwen3-8B	0.43 + 0.13	**0.30**	**0.11**	**0.57**
(b) FinQA					
Untrained					
Zero-shot	Qwen3-4B	0.29 + 0	0.71	–	0.29
Generator-only RL					
GRPO	Qwen3-4B	0.29 + 0	0.71	–	0.29
DAPO	Qwen3-4B	0.28 + 0	0.72	–	0.28
GSPO	Qwen3-4B	0.28 + 0	0.72	–	0.28
Ours					
DPA-GRPO	Qwen3-4B	0.28 + 0	**0.51**	**0.02**	**0.37**

4.1 EXPERIMENTAL SETUP

Benchmarks and evaluation units. We evaluate DPA-GRPO on TaxCalcBench TY24 (Bock et al., 2025) and FinQA (Chen et al., 2022). TaxCalcBench provides synthetic U.S. federal tax-form instances with ground-truth Form 1040 outputs produced by a deterministic tax engine. We use a fixed-seed 80/20 train/test split and evaluate 19 selected Form 1040 lines per case. A decision unit corresponds to a structured output component whose submitted value can be checked against the ground-truth output.

FinQA is a benchmark for multi-step numerical reasoning over financial reports, including tables and surrounding text. Our converted dataset contains 6,251 training examples and 883 test examples. We evaluate all reported FinQA methods on the same fixed 100-example subset of the official test split. For both benchmarks, accuracy is the fraction of evaluated decision units whose final submitted output matches the ground-truth answer.

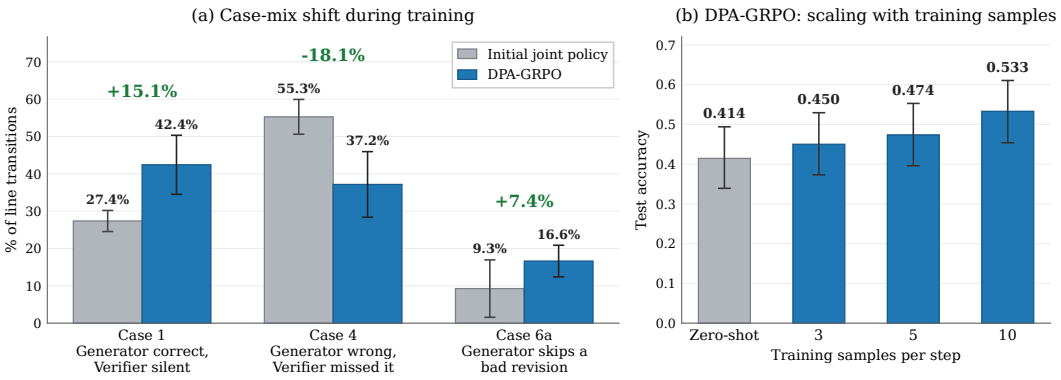


Figure 2: **DPA-GRPO improves case-taxonomy dynamics and benefits from more training samples.** (a) We compare the initial generator-verifier policy with the trained DPA-GRPO policy on three taxonomy categories. DPA-GRPO increases Case 1, corresponding to correct proposals accepted without intervention; reduces Case 4, corresponding to missed errors; and increases Case 6a, where the generator skips a bad revision. Bars report pooled case proportions over a 5-step window, and error bars show ± 1 standard deviation of per-step proportions within that window. (b) Scaling with training samples. Test accuracy improves with increasing training samples per step, and DPA-GRPO outperforms the zero-shot baseline across all training settings. Error bars show Wilson 95% confidence intervals for the binomial uncertainty of the evaluated decision-unit outcomes.

Models and methods. We evaluate Qwen3-4B and Qwen3-8B backbones (Yang et al., 2025) on TaxCalcBench and Qwen3-4B on FinQA. The zero-shot baselines use the generator directly without parameter updates and without the verifier loop. The trained single-agent baselines update only the generator using GRPO-style objectives, including GRPO (Shao et al., 2024), DAPO (Yu et al., 2025b), and GSPO (Zheng et al., 2025). These baselines produce a single proposal for each decision unit and are evaluated by proposal correctness.

DPA-GRPO trains a generator and a verifier in the two-player loop described in Section 2. For each decision unit, the generator first produces an initial proposal. The verifier either raises an SAC or remains silent. If an SAC is raised, the generator may produce a candidate revision and choose whether to keep the original proposal or revise. We report the final submitted output after this interaction. Thus, for baselines, the reported accuracy is generator-only proposal accuracy; for DPA-GRPO, it is post-interaction submitted-output accuracy.

Training and metrics. Within each benchmark, all trained methods use the same training/evaluation split and comparable LoRA fine-tuning settings. DPA-GRPO constructs paired counterfactual action groups for the verifier, $\{\text{SAC}, \text{NS}\}$, and for the generator, $\{\text{KEEP}, \text{REVISE}\}$ when an SAC is raised.

For the main evaluation, we report held-out decision-unit accuracy together with case-taxonomy diagnostics that characterize verifier intervention and generator revision behavior. Additional training-time diagnostics on three shared training instances, denoted Train-A, Train-B, and Train-C, are reported in Appendix E.3. These instances are included only as illustrative training diagnostics and are not part of the held-out evaluation.

4.2 Comparison with Baselines

Table 1 reports the main held-out results on TaxCalcBench TY24 and FinQA. On TaxCalcBench, DPA-GRPO achieves the highest observed test accuracy at both backbone sizes. With Qwen3-4B, DPA-GRPO reaches 0.53 test accuracy, compared with 0.52 for GRPO and 0.41 for zero-shot generation. With Qwen3-8B, DPA-GRPO reaches 0.57, compared with 0.55 for DAPO and 0.53 for zero-shot generation.

Because all TaxCalcBench methods are evaluated on the same eight held-out tax returns, we quantify uncertainty using a paired bootstrap over test cases, retaining all 19 evaluated decision units within

each sampled case. Relative to the strongest generator-only baseline, the observed difference is +1.3 percentage points for Qwen3-4B (95% CI $[-1.3, +4.6]$) and +2.0 percentage points for Qwen3-8B (95% CI $[-2.0, +7.9]$). Full paired-bootstrap results are reported in Appendix E.5.

On FinQA, DPA-GRPO reaches 0.37 final accuracy, compared with 0.29 for GRPO, the strongest generator-only baseline, and 0.29 for zero-shot generation. The case-taxonomy diagnostics show that DPA-GRPO reduces Case 4, corresponding to missed initial errors, from 0.71 for zero-shot generation and GRPO to 0.51, while Case 6a accounts for 0.02 of the evaluated examples. Thus, the FinQA results show both higher observed final accuracy and fewer errors that pass through without verifier intervention.

The FinQA gain is also supported by paired uncertainty analysis: relative to GRPO, the strongest generator-only baseline on this subset, DPA-GRPO improves accuracy by +8.0 percentage points, with a 95% paired-bootstrap confidence interval of $[+2.0, +14.0]$. Full paired-bootstrap FinQA comparisons are also reported in Appendix E.5.

Figure 2(b) examines sensitivity to the number of training samples per step on TaxCalcBench. Observed test accuracy rises from 0.450 with three samples per step to 0.474 with five and 0.533 with ten, compared with 0.414 for the zero-shot generator. This trend indicates that DPA-GRPO benefits from greater interaction diversity during training, although the small held-out set limits fine-grained conclusions about the shape of the scaling trend.

The case-taxonomy columns provide held-out diagnostics of verifier intervention and generator revision behavior; we analyze these dynamics further in Section 4.4.

4.3 Comparison with Supervised and Preference-Based Paired Training

To test whether DPA-GRPO can be reproduced by simpler objectives over the same generator–verifier decisions, we compare against two direct controls. **Paired-CE** trains the verifier's SAC/no-SAC and generator's KEEP/REVISE decisions using cross-entropy, while **Paired-DPO** treats the higher-reward action in each pair as preferred. Both use the same fixed set of DPA-generated transitions and oracle labels, with the same backbone, LoRA configuration, and reference-policy setup. In contrast, DPA-GRPO generates fresh interactions on-policy during training.

Table 2: Comparison with paired supervised and preference-based controls and a KL-regularization ablation on TaxCalcBench using Qwen3-4B.

Method	**Post-interaction accuracy↑**
Paired-CE	47.4%
Paired-DPO	49.3%
DPA-GRPO w/o KL	40.8%
DPA-GRPO	**53.3%**

As shown in Table 2, DPA-GRPO reaches 53.3% post-interaction accuracy, compared with 47.4% for Paired-CE and 49.3% for Paired-DPO. Because these controls use fixed interactions whereas DPA-GRPO generates fresh interactions on-policy, the comparison supports the online paired training procedure but does not isolate the optimization objective from the data-generation process.

Removing the KL anchor reduces accuracy from 53.3% to 40.8%, indicating that reference-policy anchoring is important for joint training.

We additionally test whether the gain can be explained simply by adding the generator–verifier loop at inference time. Without joint training, initial proposal accuracy is 41.4%, while post-interaction accuracy falls to 38.8%. Thus, additional verification and revision calls are not sufficient by themselves and can introduce harmful corrections when the verifier has not learned when to intervene.

4.4 Case Taxonomy Analysis

Aggregate accuracy alone does not explain how the generator-verifier interaction changes during training. Figure 2(a) compares the initial joint policy with the trained DPA-GRPO policy on three diagnostic cases from Section 2.1. Bars report pooled case proportions over a 5-step window, with error bars showing ± 1 standard deviation across steps. These are training-time interaction statistics and are distinct from the held-out case distribution in Table 1.

Case 1 corresponds to a correct generator proposal with no verifier intervention. Its frequency increases from 27.4% to 42.4%, showing increased correct silent acceptance rather than reliance solely on post-hoc repair. Case 4 captures missed errors, where an incorrect proposal passes without verifier intervention; DPA-GRPO reduces this case from 55.3% to 37.2%, indicating fewer missed errors. Case 6a captures situations where both the initial proposal and candidate revision are incorrect and the generator chooses KEEP. Its increase from 9.3% to 16.6% indicates more selective KEEP/REVISE behavior rather than revising whenever an SAC is raised.

Taken together, these shifts indicate increased correct silent acceptance, fewer missed errors, and improved revision selection. The held-out distribution should be interpreted separately. For Qwen3-8B, Case 1 decreases from 0.53 under zero-shot generation to 0.43 under DPA-GRPO, while Case 3 contributes 0.13 and Case 4 decreases from 0.47 to 0.30. Thus, the held-out accuracy gain reflects fewer missed errors together with correct outputs retained after verifier intervention, rather than an increase in silent acceptance alone.

Verifier selectivity. The learned verifier intervenes selectively rather than indiscriminately. On TaxCalcBench, it raises an SAC on 17.4% of held-out decision units, with 78.8% intervention precision and a 9.0% false-positive rate. On FinQA, the corresponding intervention rate is 21.0%, with 100.0% precision and a 0.0% false-positive rate. These results complement the case-taxonomy analysis: most raised SACs correspond to incorrect initial proposals, while unnecessary interventions on correct proposals remain relatively infrequent. Detailed definitions and intervention statistics are reported in Appendix E.2.

Structured verifier feedback. A matched FinQA feedback-format comparison yields 37.0% post-revision accuracy with structured SACs versus 35.0% with unstructured critique. The structured SAC format additionally separates the verifier's claim, supporting argument, and grounding evidence, providing an explicit interface for inspecting and tracing verifier interventions. Full results are reported in Appendix E.6.

Computational cost. On matched FinQA runs, DPA-GRPO uses $2.15\times$ the training wall-clock time and $3.11\times$ the generated tokens of generator-only GRPO; detailed training and inference costs are reported in Appendix E.4.

5 Conclusion

We proposed DPA-GRPO, a two-player generator-verifier framework for verifiable assurance reasoning in structured LLM workflows. By representing verifier feedback as safety assurance cases and converting intervention/revision decisions into paired counterfactual actions, DPA-GRPO provides training signals for both agents while preserving interpretability. Our analysis connects the induced case taxonomy to local Nash equilibrium conditions and stochastic-approximation dynamics. Across TaxCalcBench TY24 and FinQA, the results show that the learned verification–revision mechanism improves structured decision behavior across both evaluation settings. These results suggest that structured verifier interventions can serve not only as explanations, but also as actionable training signals for improving both generator and verifier behavior. We discuss limitations, including dependence on verifiable supervision, local theoretical assumptions, and computational overhead, in Appendix B.

AI USE STATEMENT

In this work, we used generative AI tools primarily for language editing, including grammar correction, improving clarity and fluency, and refining the presentation of the manuscript. We also used generative AI tools to assist with running, debugging, and checking experimental code and computational workflows, as well as to help check mathematical derivations and improve the rigor, consistency, and clarity of proof presentation. Generative AI was not used to determine the core research ideas, theoretical results, experimental design, or scientific conclusions of this work. All AI-assisted mathematical arguments, code execution, outputs, and manuscript edits were independently inspected and verified by the authors. The authors take full responsibility for the accuracy, originality, reproducibility, and final content of the manuscript.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce the proposed method and experiments throughout the paper and appendix. Section 2 specifies the DPA-GRPO generator–verifier game, reward construction, and training objectives, while Algorithm 1 in the appendix gives the complete training procedure. Section 4.1 describes the benchmarks, data splits, evaluation units, models, baselines, and reported metrics. Additional implementation details, including optimization settings, LoRA configuration, sampling procedure, and KL regularization, are provided in Appendix E.1. Statistical procedures and additional experimental analyses are reported in the corresponding appendix sections. The assumptions and complete proofs supporting the theoretical results are provided in Appendix D.

REFERENCES

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.

Hisham Abdullah Alyahya, Haidar Khan, Yazeed Alnumay, M Saiful Bari, and Bulent Yener. Zerosumeval: An extensible framework for scaling llm evaluation with inter-model competition. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 340–350, 2025.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Frédéric Berdoz, Leonardo Rugli, and Roger Wattenhofer. Can ai agents agree?, 2026. URL https://arxiv.org/abs/2603.01213.

Michael R Bock, Kara Molisee, Zachary Ozer, and Sumit Shah. Taxcalcbench: Evaluating frontier models on the tax calculation task. *arXiv preprint arXiv:2507.16126*, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. Finqa: A dataset of numerical reasoning over financial data, 2022. URL https://arxiv.org/abs/2109.00122.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Langyuan Cui, Chun Kai Ling, and Hwee Tou Ng. Game of thought: Robust information seeking with large language models using game theory. *arXiv preprint arXiv:2602.01708*, 2026.

Ewen Denney, Ganesh Pai, and Ibrahim Habli. Dynamic safety cases for through-life safety assurance. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pp. 587–590. IEEE, 2015.

Homayoon Dezfuli, Allan Benjamin, Christopher Everett, Martin Feather, Peter Rutledge, Dev Sen, and Robert Youngblood. Nasa system safety handbook. volume 2: System safety concepts, guidelines, and implementation examples. Technical report, 2015.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*, 2024.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38, 2023.

Guangyu Jiang, Mahdi Imani, Nathaniel D Bastian, and Tian Lan. Agentic ai for cyber defense: Llm-guided hierarchical multi-agent reinforcement learning. In *MILCOM 2025-2025 IEEE Military Communications Conference (MILCOM)*, pp. 1518–1523. IEEE, 2025.

Yeonsung Jung, Trilok Padhi, Sina Shaham, Dipika Khullar, Joonhyun Jeong, Ninareh Mehrabi, and Eunho Yang. Co-evolving agents: Learning from failures as hard negatives, 2026. URL https://arxiv.org/abs/2511.22254.

Tim Kelly and Rob Weaver. The goal structuring notation–a safety argument notation. In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, volume 6. Citeseer Princeton, NJ, 2004.

Benjamin Kempinski, Ian Gemp, Kate Larson, Marc Lanctot, Yoram Bachrach, and Tal Kachman. Game of thoughts: Iterative reasoning in game-theoretic domains with large language models. 2025.

Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R Bowman, Tim Rocktäschel, and Ethan Perez. Debating with more persuasive llms leads to more truthful answers. *arXiv preprint arXiv:2402.06782*, 2024.

Jakub Grudzien Kuba, Mengting Gu, Qi Ma, Yuandong Tian, Vijai Mohan, and Jason Chen. Language self-play for data-free training, 2025. URL https://arxiv.org/abs/2509.07414.

Austen Liao, Nicholas Tomlin, and Dan Klein. Efficacy of language model self-play in non-zero-sum games, 2024. URL https://arxiv.org/abs/2406.18872.

Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. *arXiv preprint arXiv:2502.20379*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

John E Lincoln. Overview of the us fda gmps: Good manufacturing practice (gmp)/quality system (qs) regulation (21 cfr part 820). *Journal of validation technology*, 18(3):17, 2012.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, et al. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *arXiv preprint arXiv:2506.24119*, 2025.

Yang Liu, Peng Sun, and Hang Li. Large language models as agents in two-player games, 2024. URL https://arxiv.org/abs/2402.08078.

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

John F Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

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. *Advances in neural information processing systems*, 36:68539–68551, 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, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10): 1095–1100, 1953.

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

Sizhe Tang, Rongqian Chen, and Tian Lan. Agent alpha: Tree search unifying generation, exploration and evaluation for computer-use agents. *arXiv preprint arXiv:2602.02995*, 2026.

Zhichao Wang. Gift: Group-relative implicit fine tuning integrates grpo with dpo and una. *arXiv preprint arXiv:2510.23868*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yihong Wu, Liheng Ma, Lei Ding, Muzhi Li, Xinyu Wang, Kejia Chen, Zhan Su, Zhanguang Zhang, Chenyang Huang, Yingxue Zhang, et al. It takes two: Your grpo is secretly dpo. *arXiv preprint arXiv:2510.00977*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Fei Xu Yu, Gina Adam, Nathaniel D Bastian, and Tian Lan. Optimizing prompt sequences using monte carlo tree search for llm-based optimization. *arXiv preprint arXiv:2508.05995*, 2025a.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025b. URL https://arxiv.org/abs/2503.14476.

Zuyuan Zhang, Mahdi Imani, and Tian Lan. Modeling other players with bayesian beliefs for games with incomplete information. *arXiv preprint arXiv:2405.14122*, 2024.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

A Related Work

RL-based post-training and GRPO. RL-based post-training commonly optimizes a KL-regularized objective against a reference policy, often using PPO-style updates or variants (Schulman et al., 2017; Ouyang et al., 2022). GRPO is a prominent alternative that avoids explicit value-function training by normalizing rewards within groups of samples, which can improve stability and throughput for verifiable reward settings (Shao et al., 2024). Our work adopts GRPO as the primary optimization primitive and uses KL regularization to bound policy drift.

Preference optimization and GRPO–DPO connections. Direct Preference Optimization (DPO) optimizes policies from preference pairs via an implicit reward view, offering a supervised-like alternative to RL-style updates (Rafailov et al., 2023). Recent work connects GRPO to contrastive preference optimization and shows that pairwise grouping, or 2-GRPO, can retain key optimization behavior (Wu et al., 2025). Related reward-matching perspectives further connect group-normalized RL objectives with implicit-reward formulations from preference optimization (Wang, 2025). Our setting naturally induces paired actions for both roles: SAC/no-SAC for the verifier and keep/revise for the generator. This allows us to construct counterfactual dual-action GRPO updates that resemble preference optimization, while still using explicit verifiable rewards rather than human preference labels.

Multi-agent critique, verification, and iterative refinement. A broad literature explores self-critique, debate, and multi-agent refinement for improving reliability, often via prompting and heuristic selection (Madaan et al., 2023; Shinn et al., 2023; Du et al., 2024; Khan et al., 2024; Lifshitz et al., 2025; Zhang et al., 2024; Tang et al., 2026; Jiang et al., 2025). We treat verification as an explicit decision policy (SAC vs. no-SAC) trained with verifiable signals, and we train the generator to respond to interventions through revision decisions.

LLMs as agents in two-player games. Recent works have framed LLM reasoning and post-training through two-player or game-theoretic interactions, drawing on classical notions such as Nash equilibrium and stochastic games, (Nash Jr, 1950; Shapley, 1953). Liu et al. (2024) interpret LLM training and interaction as a language-based two-player game, where each agent has its own policy, reward, and best-response dynamics. Kempinski et al. (2025) study iterative self-play reasoning in strategic domains and show that naive LLM action selection can be highly exploitable, while best-response-style refinement improves robustness. Cui et al. (2026) formulate information-seeking as a two-player zero-sum extensive-form game and approximate Nash-equilibrium strategies through game-theoretic search. These works support a game-theoretic view of generator–verifier interaction, but our method differs by grounding the game in structured output components, with paired SAC/no-SAC and keep/revise actions trained using verifiable counterfactual rewards.

B Limitations

DPA-GRPO is evaluated on two benchmarks with programmatic correctness signals: TaxCalcBench TY24 for structured tax-form completion and FinQA for multi-step numerical reasoning over finan-

cial tables and text. Both benchmarks nevertheless provide programmatic correctness signals, and the current experiments do not establish robustness to systematically noisy, incomplete, or model-generated supervision. In particular, errors that reverse the relative ordering of paired actions can direct the update toward a lower-quality action, while missing supervision reduces the number and coverage of valid paired updates. We discuss these effects and possible confidence-weighted extensions in Appendix F.

Our theoretical analysis is local and relies on stochastic-approximation and game-theoretic assumptions; it does not imply global convergence or guarantee that finite-step LLM training reaches a Nash equilibrium. Empirically, performance may also depend on model capacity, reward design, SAC quality, and the availability of useful revision opportunities.

DPA-GRPO incurs additional computation relative to generator-only training because each decision unit requires generation and verification, with revision invoked when an SAC is raised. Our measurements quantify this overhead in Appendix E.4; the use of separate LoRA adapters over a shared frozen backbone keeps the additional memory footprint small. Future work can further reduce computational cost through more efficient verification, sampling, and revision strategies.

From a broader-impact perspective, structured SACs can improve auditability by exposing claim–argument–evidence records for verifier interventions. These records make the basis of a challenge easier to inspect, but they should not be interpreted as formal guarantees of correctness or safety. In high-stakes settings, model outputs should remain subject to appropriate independent validation and domain-specific review.

C Algorithm

Algorithm 1 DPA-GRPO training for the generator–verifier game.

Require: Policies $\pi_{G,\phi}, \pi_{V,\theta}$, oracle S, reference policies $\pi_{G,\mathrm{ref}}, \pi_{V,\mathrm{ref}}$, KL coefficient β, step sizes η_k

for iteration $k = 1, 2, \ldots$ **do**
  Sample a batch of task instances and decision units.
  for each decision context $m_t = (\tau, u_t, o_{<t})$ **do**
    Sample initial proposal $x_t \sim \pi^x_{G,\phi}(\cdot \mid m_t)$ and compute $S_x \leftarrow S(x_t; \tau, u_t)$.
    Set $c_{\mathrm{sac}} \leftarrow 1 - S_x$.
    Sample verifier action $y_t \sim \pi_{V,\theta}(\cdot \mid m_t, x_t)$.
    Construct verifier pair $\{\mathrm{SAC}, \mathrm{NS}\}$ and compute counterfactual paired rewards $R_V(\mathrm{SAC}) = c_{\mathrm{sac}},\ R_V(\mathrm{NS}) = 1 - c_{\mathrm{sac}}$.
    if $y_t = \mathrm{SAC}$ **then**
      Sample candidate revision $z_t \sim \pi^z_{G,\phi}(\cdot \mid m_t, x_t, y_t)$ and compute $S_z \leftarrow S(z_t; \tau, u_t)$.
      Construct generator pair $\{\mathrm{KEEP}, \mathrm{REVISE}\}$ and compute counterfactual paired rewards $R_G(\mathrm{KEEP}) = S_x,\ R_G(\mathrm{REVISE}) = S_z$.
      Sample revision action $a_t \sim \pi^a_{G,\phi}(\cdot \mid m_t, x_t, y_t, z_t)$.
      Set $o_t \leftarrow x_t$ if $a_t = \mathrm{KEEP}$, and $o_t \leftarrow z_t$ if $a_t = \mathrm{REVISE}$.
    else
      Set $z_t \leftarrow \varnothing$, $a_t \leftarrow \mathrm{KEEP}$, and $o_t \leftarrow x_t$.
    end if
    Store decision-unit transition, paired rewards, and log-probabilities.
  end for
  Update θ using GRPO on verifier SAC/NS paired rewards with β-KL anchoring to $\pi_{V,\mathrm{ref}}$.
  Update ϕ using GRPO on the candidate-revision group and the paired $\{\mathrm{KEEP}, \mathrm{REVISE}\}$ action group, with β-KL anchoring to $\pi_{G,\mathrm{ref}}$.
end for

D Theory Guarantees

This section contains theorem-specific technical assumptions and proofs for Theorems 2 and 3.

D.1 Proof of Local Nash Necessary Conditions

The main text states the local non-degeneracy assumption because it is central to the interpretation of the case taxonomy. We add here the KL-specific assumption required to transfer reward-improving directions to the regularized objectives.

Assumption 4 (KL-dominance for reward-improving deviations)**.** For every local direction identified by Lemma 1 or Lemma 2 that strictly improves the corresponding unregularized reward, the first-order reward gain is not smaller than the first-order increase in the corresponding KL penalty. Equivalently, the same direction improves the full KL-regularized objective.

Proof of Theorem 2. Suppose, for contradiction, that (ϕ^*, θ^*) is a local Nash equilibrium but the verifier assigns nonzero probability to a pointwise reward-suboptimal action on a positive-measure set of contexts. By Lemma 1, there exists a local direction $\Delta\theta$ such that $D_\theta \widetilde{J}_v(\theta^*; \phi^*)[\Delta\theta] > 0$. By Assumption 4, the same direction also increases J_v, contradicting local Nash optimality of θ^*.

Now suppose the generator assigns nonzero probability to a strictly suboptimal KEEP/REVISE action on a positive-measure set of SAC contexts. By Lemma 2, there exists a local direction $\Delta\phi$ such that $D_\phi \widetilde{J}_f(\phi^*; \theta^*)[\Delta\phi] > 0$. By Assumption 4, the same direction also increases J_f, contradicting local Nash optimality of ϕ^*. Therefore both conclusions must hold. □

Case-taxonomy consequence. Theorem 2 rules out positive equilibrium mass on verifier mistakes and strictly suboptimal SAC responses. Case 5a can remain admissible because a wrong proposal followed by correct SAC and successful revision can be reward-consistent for both players. A stronger Case-1-only result requires an additional assumption, such as an explicit intervention/revision cost or a dominance condition that producing a correct proposal is strictly preferred to relying on the SAC–revision pathway.

D.2 Proof of Convergence Statement

For the convergence result, we use standard stochastic-approximation assumptions.

Assumption 5 (Smoothness and compactness)**.** The parameter domains for ϕ and θ are compact, or the iterates are projected to a compact set. The objectives J_f and J_v are continuously differentiable and have Lipschitz gradients on this set.

Assumption 6 (Bounded noise)**.** Rewards and GRPO advantages are uniformly bounded. The stochastic gradient noise terms have bounded second moments and form martingale-difference sequences with respect to the training filtration.

Assumption 7 (Step sizes)**.** The learning rates satisfy the Robbins–Monro conditions

$$\sum_{t=1}^{\infty} \eta_t = \infty, \qquad \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

Assumption 8 (Controlled replay and stale-sample bias)**.** Let b_t^f and b_t^v denote bias terms due to replay, stale log-probabilities, and finite group normalization. We assume

$$\sum_{t=1}^{\infty} \eta_t \left(\|b_t^f\| + \|b_t^v\| \right) < \infty.$$

Assumption 9 (Infinite visitation)**.** Each role is updated infinitely often, and each decision branch appearing in the objectives is visited infinitely often. In particular, the SAC branch has nonzero visitation frequency along the training trajectory.

The alternating GRPO iterates can be written as

$$\phi_{t+1} = \phi_t + \eta_t \left(\nabla_\phi J_f(\phi_t; \theta_t) + M_{t+1}^f + b_t^f \right),$$

$$\theta_{t+1} = \theta_t + \eta_t \left(\nabla_\theta J_v(\theta_t; \phi_{t+1}) + M_{t+1}^v + b_t^v \right),$$

where M_{t+1}^f and M_{t+1}^v are martingale-difference noise terms and b_t^f, b_t^v collect finite-sample, replay, and stale-log-probability bias.

Proof sketch of Theorem 3. By Assumptions 6 and 7, the martingale-difference noise terms average out asymptotically. Assumption 8 ensures that replay, stale log-probabilities, and finite group-normalization effects do not accumulate enough to alter the limiting ODE.

The update is block-coordinate: the verifier update uses ϕ_{t+1} instead of ϕ_t. Since $\phi_{t+1}-\phi_t = O(\eta_t)$ and the gradients are Lipschitz by Assumption 5, this introduces an $O(\eta_t)$ perturbation inside the verifier gradient and hence an $O(\eta_t^2)$ perturbation in the parameter update, which is summable by Assumption 7. Therefore the interpolated process tracks the ODE

$$\dot{\phi} = \nabla_\phi J_f(\phi;\theta), \qquad \dot{\theta} = \nabla_\theta J_v(\theta;\phi).$$

Standard stochastic-approximation results imply convergence to an internally chain transitive set of this ODE. Any isolated asymptotically stable limit point of the ODE is stationary, yielding $\nabla_\phi J_f(\phi;\theta) = 0$ and $\nabla_\theta J_v(\theta;\phi) = 0$. □

E Experimental Details and Additional Results

E.1 Implementation Details

The TaxCalcBench and FinQA data and evaluation protocols are described in Section 4.1. We evaluate Qwen3-4B and Qwen3-8B backbones and train all methods using LoRA adapters with AdamW. For DPA-GRPO, the generator and verifier use separate LoRA adapters over the same backbone family.

LoRA configuration. We use LoRA rank $r = 8$, scaling parameter $\alpha = 32$, and dropout 0.05. Separate generator and verifier adapters are applied to the attention and MLP linear projections.

Optimization. We use AdamW with learning rate 10^{-5} and weight decay 0.01. DPA-GRPO uses a REINFORCE-style policy-gradient update with a KL anchor to a frozen reference policy. The KL coefficient is $\beta = 0.04$, applied uniformly across role and branch updates.

Sampling and paired-action groups. The paired-action group size is 2. For the verifier, the paired actions are $\{\mathrm{SAC}, \mathrm{NS}\}$, and for the generator on SAC branches, the paired actions are $\{\mathrm{KEEP}, \mathrm{REVISE}\}$. We use temperature 0.2 and top-$p = 0.9$ for initial proposal and verifier-action sampling. Candidate revisions are sampled with temperature 0.7 using best-of-K revision generation with $K = 5$ candidates, where candidates are filtered for parse validity and then scored by oracle correctness.

Training schedule. We train for up to 30 steps and evaluate every 5 steps.

Hardware. Experiments are implemented in Python 3 and run on a Linux machine with an AMD EPYC 7513 32-Core Processor CPU and a single NVIDIA RTX A6000 GPU. We use mixed-precision training with bf16 autocast and gradient checkpointing.

E.2 Verifier Intervention Statistics

Table 3: Held-out verifier intervention statistics.

Benchmark	Intervention rate	Intervention precision↑	FPR↓
TaxCalcBench	17.4%	78.8%	9.0%
FinQA	21.0%	100.0%	0.0%

We report three statistics used to characterize verifier intervention behavior in Table 3. The *intervention rate* is the fraction of evaluated decision units for which the verifier raises an SAC. *Intervention precision* is the fraction of raised SACs for which the initial proposal is incorrect. The *false-positive rate* (FPR) is the fraction of initially correct proposals for which the verifier raises an SAC.

Table 4: Training-time accuracy on three common training samples. These examples are illustrative training diagnostics and are not part of the held-out evaluation.

Method	Backbone	Train-A↑	Train-B↑	Train-C↑
Untrained				
Zero-shot	Qwen3-4B	0.53	0.32	0.26
Zero-shot	Qwen3-8B	0.84	0.42	0.42
Generator-only RL				
GRPO	Qwen3-4B	0.74	0.47	0.32
DAPO	Qwen3-4B	0.74	0.37	0.21
GSPO	Qwen3-4B	0.74	0.37	0.21
GRPO	Qwen3-8B	0.84	0.63	0.32
DAPO	Qwen3-8B	0.84	0.63	0.53
GSPO	Qwen3-8B	0.84	**0.68**	**0.58**
Ours				
DPA-GRPO	Qwen3-4B	0.74	0.58	0.37
DPA-GRPO	Qwen3-8B	**0.90**	**0.68**	**0.58**

E.3 Additional Training-Time Diagnostics

Table 4 reports accuracy on three common training samples that appear across the trained runs. These examples are included as representative diagnostics of optimization behavior rather than as evidence of held-out generalization. DPA-GRPO matches or exceeds the generator-only baselines across these training samples.

E.4 Computational Cost Details

Table 5: Training-cost comparison. Token counts are normalized per decision unit.

Benchmark	Training SAC rate	Generator-only tokens/unit	DPA-GRPO tokens/unit	Token multiplier	Wall-clock multiplier
FinQA	32.5%	1,387	4,319	3.11×	2.15×
TaxCalcBench	28.0%	5,067	13,613	2.69×	3.8×*

*The TaxCalcBench runs use different numbers of decision units per optimizer step, so the raw per-step wall-clock multiplier is not directly comparable with the matched FinQA measurement.

DPA-GRPO introduces additional computation because each decision unit requires both generation and verification, with revision invoked selectively. We report training-time and inference-time computational measurements for the generator-only GRPO and DPA-GRPO configurations in Table 5. Token counts are computed using the Qwen3-4B tokenizer. The generator and verifier use separate LoRA adapters over a shared frozen backbone.

On the matched FinQA runs, DPA-GRPO uses approximately $2.15\times$ the training wall-clock time and $3.11\times$ the generated tokens of generator-only GRPO. On TaxCalcBench, the token multiplier is $2.69\times$; the raw wall-clock multiplier is not directly comparable because the runs use different numbers of decision units per optimizer step.

At inference time, every decision unit requires an initial generator proposal and a verifier decision. A candidate revision and subsequent KEEP/REVISE decision are generated only when the verifier raises an SAC. If p_{SAC} denotes the intervention rate, the expected number of model calls per decision unit is

$$\mathbb{E}[N_{\mathrm{calls}}] = 2 + 2p_{\mathrm{SAC}}. \tag{1}$$

Using the held-out intervention rates of 17.4% on TaxCalcBench and 21.0% on FinQA gives approximately 2.35 and 2.42 model calls per decision unit, respectively, compared with the four-call worst case when every proposal is revised.

Both generator-only GRPO and DPA-GRPO use 10 cases per optimizer step, on-policy training, and a 2,048-token generation budget per role. Each generator and verifier LoRA adapter has approximately 16.5 million trainable parameters, corresponding to about 0.41% of the 4B backbone.

Adding the verifier adapter and optimizer state increases the measured training-memory footprint by approximately 0.2 GiB over a total footprint of roughly 26–27 GiB.

E.5 Paired Uncertainty Analysis

Table 6: Paired bootstrap confidence intervals for selected accuracy differences on TaxCalcBench and FinQA.

Comparison	Difference	Paired bootstrap 95% CI
TaxCalcBench TY24		
DPA-GRPO 4B – GRPO 4B	+1.3 pp	$[-1.3, +4.6]$
DPA-GRPO 4B – Zero-shot 4B	+11.8 pp	$[+0.7, +23.0]$
DPA-GRPO 8B – DAPO 8B	+2.0 pp	$[-2.0, +7.9]$
DPA-GRPO 8B – Zero-shot 8B	+3.9 pp	$[-1.3, +10.5]$
FinQA		
DPA-GRPO 4B – GRPO 4B	+8.0 pp	$[+2.0, +14.0]$
DPA-GRPO 4B – Zero-shot 4B	+8.0 pp	$[+2.0, +14.0]$
DPA-GRPO 4B – GSPO 4B	+9.0 pp	$[+3.0, +15.0]$

For TaxCalcBench, all methods are evaluated on the same eight held-out tax returns. We therefore use a paired case-level bootstrap, drawing eight returns with replacement while retaining all 19 evaluated decision units associated with each sampled case. DPA-GRPO and baseline outcomes are resampled together, preserving their pairing at the case level.

For FinQA, all methods are evaluated on the same fixed 100-question subset of the official test split. We instead perform paired bootstrap resampling directly over these 100 questions. In both benchmarks, we report percentile 95% confidence intervals from 10,000 bootstrap resamples with seed 42. The resulting intervals are summarized in Table 6.

On FinQA, GRPO and zero-shot generation have identical per-question correctness on the evaluated subset, yielding the same paired difference and confidence interval relative to DPA-GRPO. DPA-GRPO is correct on nine questions missed by either baseline and incorrect on one question answered correctly by the baseline. GSPO is correct on one fewer question, producing the slightly larger +9.0 percentage-point difference.

E.6 Comparison of Structured and Unstructured Feedback

Table 7: Matched comparison of verifier feedback formats on FinQA. Both variants use the same checkpoint, evaluation examples, decoding settings, and revision procedure.

Feedback format	Post-revision accuracy↑	95% CI
Unstructured critique	35.0%	$[26.0, 44.0]$
Structured claim–argument–evidence SAC	**37.0%**	$[28.0, 46.0]$

To examine whether the claim–argument–evidence structure contributes beyond generic natural-language feedback, we conduct a matched feedback-format comparison on FinQA in Table 7. We hold the model checkpoint, evaluation examples, decoding settings, and revision procedure fixed, and vary only the feedback provided to the generator. Both variants begin from the same 28.0% initial proposal accuracy.

Both feedback formats improve over the shared 28.0% initial proposal: unstructured critique reaches 35.0%, while structured SAC feedback reaches 37.0%. The observed difference between the two feedback formats is small relative to the uncertainty in this evaluation, so we do not interpret this comparison as establishing a clear accuracy advantage for the claim–argument–evidence structure.

We use the structured SAC representation primarily as an explicit and auditable interface for verifier interventions: the claim identifies the affected component, the argument states the relevant rule or dependency, and the evidence grounds the intervention in the task context.

F Analysis of Oracle Noise and Partial Supervision

The paired objective depends on the relative reward ordering of the two actions within a decision context. To illustrate the effect of oracle errors, suppose that the correct paired update direction is g and that the oracle independently reverses the pairwise preference with probability e. The expected update is

$$(1 - e)g + e(-g) = (1 - 2e)g. \tag{2}$$

Under this simplified independent-noise model, the correctly directed signal is attenuated when $e < 0.5$, vanishes at $e = 0.5$, and reverses when $e > 0.5$. Perturbations that preserve the relative ordering of the two actions can change the magnitude of the update without reversing its direction.

Partial supervision creates a different limitation. When a decision unit cannot be scored reliably, no valid paired update can be constructed for that unit, reducing the amount and coverage of available training supervision. For confidence-valued oracles, the binary correctness values can instead be replaced by continuous scores, with low-confidence pairs down-weighted or omitted.

G Illustrative line-level interactions

Each example corresponds to one *decision unit*: a single Form 1040 line in one TaxCalcBench case. The *Generator* emits the line JSON, while the *Verifier* emits a structured assessment that is mapped to `NO_SAC` or `SAC`. When an SAC is raised, the Generator produces a candidate revision and subsequently chooses `KEEP` or `REVISE`. Oracle tags $S_x, S_z \in \{0, 1\}$ indicate whether the initial proposal and candidate revision match the gold value and are used only for training-time scoring.

Case 1. *Case* `single-w2-multiple-1099int-withholding-schedule-b`, *line* 1a (AGI).

> **Generator:** AGI = 87,451; rationale: wages with no adjustments.
> **Verifier:** NO_SAC; line finding correct; cites W-2 Box 1.
> **Revision:** not considered (no SAC).
> **Oracle:** S_x=1 (draft matches gold).

Reading: both roles agree the line is fine; the Verifier does not intervene. This is the desirable "Generator correct, Verifier silent" mass that Table 1 tracks as Case 1.

Case 4. *Case* `single-w2-unemployment-1099g`, *line* 1a (AGI).

> **Generator:** AGI = 131,131; rationale ties the amount to ``prior-year AGI'' from the intake header.
> **Verifier:** NO_SAC; line finding still marked correct in the JSON (false reassurance).
> **Revision:** not considered.
> **Oracle:** S_x=0 (draft does *not* match gold on this line).

Reading: the draft is wrong, but the Verifier never raises SAC, so no repair loop can run. This is Case 4, the dominant failure mode of the initial joint policy.

Example 5a. *Case* `single-multiple-w2-excess-social-security-tax-same-ein`, *line* 15 (taxable income).

> **Generator (draft):** amount 146,000 while the text argues AGI − standard deduction = 147,097 (numeric inconsistency).

Verifier: SAC; line finding wrong; disputes arithmetic.
Generator (revise): amount 147,097, aligned with stated AGI and standard deduction.
Generator decision: REVISE (commit the revised line).
Oracle: S_x=0, S_z=1; justified SAC.


```
{
  "claim_id": "linewise-claim",
  "overall_verdict": "reject",
  "overall_confidence": 0.9,
  "line_findings": [
    {
      "form": "1040",
      "line": "15",
      "claim": "Taxable income calculated as AGI minus standard deduction",
      "verdict": "wrong",
      "confidence": 0.9,
      "arguments": ["The taxable income should be calculated as AGI minus the standard deduction.
      The AGI is $161,697, and the standard deduction is $14,600.
      The correct taxable income is $161,697 - $14,600 = $147,097, not $146,000."],
      "evidence": [
        { "type": "input_field", "location": "return_data.w2[0].wages.value", "value": 116022 },
        { "type": "input_field", "location": "return_data.w2[1].wages.value", "value": 45675 },
        { "type": "input_field", "location": "context_so_far[0].amount", "value": 161697.0 },
        { "type": "input_field", "location": "context_so_far[2].amount", "value": 14600.0 }
      ]
    }
  ]
}
```


Case 5a SAC.

Reading: the Verifier correctly flags a real error; the revised line strictly improves over the draft ($S_z > S_x$ in the usual partial ordering); the Generator accepts the repair. Taxonomy bucket 5a: wrong draft, valid critique, revision fixes the line.

Example 6a. *Case* `single-w2-multiple-1099int-withholding-schedule-b`, *line* 11 (tax liability).

Generator (draft): liability 12,222.30; rationale mixes bracket steps and contradicts the headline number.
Verifier: SAC; line finding wrong; argues bracket / tax-table mistake.
Generator (revise): proposes *different* liability 11,080.34 (still not gold).
Generator decision: KEEP (retain the original draft, discard revision).
Oracle: S_x=S_z=0; justified SAC; Generator choice is counterfactually optimal among {draft, revision}.

Reading: the Verifier is right to be suspicious, but swapping in the revision would not improve strict correctness; KEEP implements "skip a bad revision." This is Case 6a and requires the verifier intervention and generator KEEP/REVISE mechanism.