BEYOND AVERAGED POLICY GRADIENTS: RLVR LEARNS DENSE PARITY FROM FINAL REWARDS

Anonymous authors
Paper under double-blind review

ABSTRACT

While reinforcement learning with verifiable rewards (RLVR) has achieved striking empirical success, a fundamental theoretical question remains open: when/how can final correctness alone provide an effective learning signal? In this work, we demonstrate that the limitation of ordinary Group Relative Policy Optimization (GRPO) in this setting is an optimization failure rather than an information deficit. Focusing on dense parity as a benchmark, we first prove that the population policy gradient underlying standard GRPO suffers from exponential decay, leaving an exponentially weak learning signal under the standard relative-reward mechanism. To resolve this, we introduce Verifier Posterior Transport (VPT), a sequential RLVR framework that executes a KL-regularized policy update after every example. Unlike GRPO, VPT guarantees a fixed, non-vanishing log-odds gain between successful and failed executions using only the exact same binary final reward. We construct a recursive transformer that realizes VPT exactly and prove that the example complexity required for target recovery scales linearly with input dimension at fixed confidence. Our results demonstrate that dense parity can be learned with linear sample complexity without intermediate supervision. Ultimately, this reveals that the bottleneck in final-reward RLVR is not whether the verifier holds sufficient signal, but how the update rule transforms that information into a learning signal.

1 INTRODUCTION

Large language models (LLMs), usually built with transformers (Vaswani et al., 2017), still struggle with tasks that require several dependent reasoning steps. To address this challenge, chain-of-thought (CoT) prompting encourages models to generate intermediate reasoning steps before giving a final answer (Wei et al., 2022). During training, these steps can be supervised using reference reasoning traces (Nye et al., 2021) or feedback on the correctness of individual steps (Lightman et al., 2024). However, obtaining reliable step-level supervision is costly, as multiple valid reasoning paths are typically feasible. Reinforcement learning with verifiable rewards (RLVR) takes another route: it trains a complete response from a checkable final outcome.

RLVR is usually trained with policy gradient methods in which policy samples complete responses, a verifier scores them, and the resulting advantages determine the update. Proximal policy optimization (PPO) is a standard choice and is commonly paired with a learned critic (Schulman et al., 2017). GRPO is widely used for reasoning models because it replaces the critic with relative advantages computed from a group of responses to the same prompt, reducing memory cost (Shao et al., 2024; Guo et al., 2025). Since a verifier can score such a group directly, GRPO provides a simple and effective RLVR training pipeline. This is the framework that we adopt in this study.

Automatic verifiers include exact answer checkers for mathematics (Shao et al., 2024), compilers and unit tests for code (Le et al., 2022; Gehring et al., 2024), and proof assistants for formal proofs (Xin et al., 2025). Although much of the RLVR literature is empirical and algorithmic – studying reasoning performance and capability boundaries (Shao et al., 2024; Guo et al., 2025; Chen et al., 2026), reward quality and data efficiency (Shao et al., 2025; Wang et al., 2026), credit assignment across samples and tokens (He et al., 2026; Tang et al., 2026; Zhai et al., 2026), and the weak learning signal of GRPO on hard problems, which is not necessarily resolved by increasing the number of completions per prompt (Noukhovitch et al., 2026) – a smaller but growing body of theoretical

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.

literature investigates RLVR training dynamics (Mroueh, 2025; Huang et al., 2026), the learnability of compositional reasoning from outcome rewards (Barzilai et al., 2026), and the ability of RLVR to learn backtracking from final rewards (Wei & Kim, 2026). However, so far insufficient attention has been paid to the distinction between the information contained in a verifier response and the way an optimizer combines that information across prompts. A main thrust of this work is to investigate this distinction in parity learning by analyzing how policy updates can preserve and integrate the final reward information.

Our work is inspired by Kim & Suzuki (2025), showing that transformers learn dense parity efficiently when CoT provides intermediate parity computations, whereas learning directly from final supervision is difficult via an averaged empirical gradient interface. We adopt the same restriction used in their analysis: the model processes each training example separately, no external procedure solves the accumulated parity equations, and information across examples can only be exploited through the optimization of the model parameters. Building on this framework, we investigate the following problem: *can a transformer trained with RLVR learn dense parity without intermediate supervision, relying solely on final correctness?*

This question highlights the distinction between the accuracy of a rollout estimate and the strength of the population signal. While methods like GRPO can use additional rollouts to reduce variance in estimation error, sampling alone cannot increase the underlying mean gradient. A single example may clearly distinguish successful outputs from failed ones, yet the corresponding gradient contributions can cancel across different examples after averaging. As a result, the verifier can be informative on individual examples even when the resulting population gradient is exponentially weak.

We adopt the same dense parity setting as Kim & Suzuki (2025). A policy chooses among $M = \binom{d}{k}$ complete parity programs and receives one bit that records correctness on the current example. Each example partitions the candidates into successes and failures, and the unknown target always succeeds. Thus, every verifier outcome carries constraint information. Nevertheless, parity orthogonality makes the population policy gradient underlying ordinary GRPO proportional to the probability $q(p)$ already assigned to the target. From a uniform policy, $q(p) = 1/M = e^{-\Theta(d)}$ in the dense regime. More rollouts estimate this weak direction more accurately, but do not make it stronger.

To avoid this obstruction, we apply a policy update with KL regularization after every example and call the resulting sequential procedure VPT. This update increases the log odds of any successful program against any failed program by exactly 2β, regardless of their current probabilities. These fixed gains accumulate. We prove that VPT identifies dense parity from $O(d)$ examples and construct a recursive transformer whose score update realizes VPT exactly from final correctness.

Dense parity provides a controlled mathematical setting in which the information supplied by a final verifier can be separated from the information preserved by the optimization rule. The same issue arises in other tasks with automatically checked outcomes. Compilers and unit tests provide this feedback for code generation and repair (Le et al., 2022; Gehring et al., 2024), proof assistants for formal theorem proving (Xin et al., 2025) and exact answer checkers for mathematical reasoning (Shao et al., 2024). The broader implication is that more rollouts may estimate a weak population direction more accurately without strengthening it; whether verifier information survives depends on how it is incorporated into the policy update.

Our contributions are as follows:

- We derive the exact population gradient underlying ordinary GRPO and prove that its norm is exponentially small for dense parity at an uninformative initialization.
- Using the identical final reward, we show that VPT produces a fixed gain in log odds for every successful program relative to every failed program, independently of the current target probability. This avoids the exponentially small target mass factor that appears in the ordinary population policy gradient.
- We realize VPT exactly with a recursive transformer and prove recovery of the target from $O(d)$ examples.
- Experiments reproduce the optimization separation in a standard trainable transformer and show that the number of examples required by the exact construction grows linearly with dimension.

1.1 Related works

RLVR and reasoning supervision. GRPO forms relative advantages from groups of responses and avoids the learned critic used in PPO (Shao et al., 2024). Empirical RLVR studies examine reasoning gains and their limits (Guo et al., 2025; Chen et al., 2026), reward quality and data efficiency (Shao et al., 2025; Wang et al., 2026), and the roles of token credit and sample polarity (He et al., 2026; Tang et al., 2026). Recent RLVR theory analyzes policy and training dynamics (Mroueh, 2025; Huang et al., 2026), compositional reasoning from outcome rewards (Barzilai et al., 2026), and backtracking from final rewards (Wei & Kim, 2026). Process supervision can expose intermediate targets (Nye et al., 2021; Lightman et al., 2024). Intermediate generation can also increase transformer computation power (Feng et al., 2023; Merrill & Sabharwal, 2024; Li et al., 2024), while intermediate parity labels can make learning efficient (Wies et al., 2022; Kim & Suzuki, 2025). Our result does not require those labels.

Parity learning and policy optimization. Parity is difficult for statistical query methods and for gradients computed at finite precision (Kearns, 1998; Shalev-Shwartz et al., 2017; Shamir, 2018), while memory can change the complexity of streaming parity learning (Raz, 2018). Our setting differs from these approaches because the learner receives only the final binary correctness reward and does not perform Gaussian elimination. KL regularization and information geometric methods have a long history in policy optimization (Peters & Schaal, 2007; Peters et al., 2010; Levine, 2018). These works study general principles for policy optimization rather than dense parity learning from a final binary correctness reward. In this setting, we show that ordinary policy gradient and VPT have sharply different learning behavior, and that the VPT update can be realized exactly by the recursive transformer.

2 Problem Setup

2.1 Notation

For a positive integer d, let $[d] := \{1, \dots, d\}$. For a finite set S, $|S|$ is its cardinality, $\mathrm{Unif}(S)$ is the uniform distribution on S, and $\Delta(S)$ is the probability simplex on S, i.e., $\Delta(S) := \left\{q : S \to [0,1] : \sum_{s\in S} q(s) = 1\right\}$, where $q(s)$ is the probability of s under q. The symbols $\mathbb{E}$ and $\mathbb{P}$ denote expectation and probability, and $\mathbf{1}\{E\}$ is the indicator of an event E. For distributions q, r on the same finite set, $D_{\mathrm{KL}}(q\|r) := \sum_A q(A) \log \frac{q(A)}{r(A)}$ is their KL divergence, with the usual zero probability convention. For sets A, B, $A\Delta B := (A \setminus B) \cup (B \setminus A)$ is their symmetric difference. Bold lowercase symbols denote vectors. We use $\|\cdot\|$ for the Euclidean norm, $\odot$ for coordinatewise multiplication, and $\mathbf{0}_m, \mathbf{1}_m$ for the zero and all ones vectors in $\mathbb{R}^m$. For a vector $\boldsymbol{x}$, $\boldsymbol{x}_{-j}$ removes coordinate j. For nonnegative functions f, g, $f(d) = O(g(d))$ means $f(d) \le Cg(d)$ for some constant $C > 0$ and all sufficiently large d, while $f(d) = \Theta(g(d))$ means $cg(d) \le f(d) \le Cg(d)$ for constants $c, C > 0$ and all sufficiently large d. Finally, $\mathrm{poly}(d)$ denotes an unspecified factor bounded by a polynomial in d. All unsubscripted logarithms are natural, while $\log_2$ is base two.

2.2 The parity problem and the program policy

Let $d \ge k \ge 2$ and let P be the collection of subsets of $[d]$ with cardinality k. As in Kim & Suzuki (2025), we identify each $A \in P$ with the parity function

$$A(\boldsymbol{x}) := \prod_{j\in A} x_j, \qquad \boldsymbol{x} \in \{-1,+1\}^d. \tag{1}$$

An unknown target $p \in P$ generates independent training examples

$$\boldsymbol{x}^t \sim \mathrm{Unif}\left(\{-1,+1\}^d\right), \qquad y^t = p(\boldsymbol{x}^t). \tag{2}$$

Thus every $A \in P$ is a candidate parity rule, and the learning problem is to identify the unknown target p. We write $M := |P| = \binom{d}{k}$. Our main regime is dense parity, where both k and $d-k$ are $\Theta(d)$. In this regime $M = e^{\Theta(d)}$.

We represent the learner by a probability distribution over the candidate parities in P. Let θ denote the model parameters, which determine a differentiable distribution q_θ on P, where

$$q_\theta(A) = \mathbb{P}\{\text{the learner selects } A\}, \qquad A \in P.$$

If the learner selects A, its prediction on an input $\boldsymbol{x}$ is deterministically given by $A(\boldsymbol{x})$. Thus, our analysis depends only on the distribution q_θ over candidate parities. Whenever $\nabla_\theta \log q_\theta(A)$ is used, we assume that q_θ has full support on P. At round t, we write q_t for the current distribution; for a parametric policy, $q_t = q_{\theta_t}$.

2.3 Final reward and sequential training

On round t, a sampled program $A \sim q_t$ predicts $A(\boldsymbol{x}^t)$. The verifier returns only whether this final prediction is correct:

$$R_t(A) := \mathbf{1}\{A(\boldsymbol{x}^t) = y^t\} \in \{0, 1\}. \tag{3}$$

For the analysis, it is convenient to recenter the same binary feedback as

$$\rho_t(A) := 2R_t(A) - 1 = y^t A(\boldsymbol{x}^t) = \prod_{j \in p\Delta A} x_j^t \in \{-1, +1\}, \tag{4}$$

Equation (4) is the basic parity identity used throughout the paper. For the target $A = p$, it gives $\rho_t(p) = 1$ on every example. For the target $A = p$, we have $\rho_t(p) = 1$ for every example. For any fixed wrong program $A \neq p$, the set $p\Delta A$ is nonempty. Since $\boldsymbol{x}^t$ is uniformly distributed on $\{-1, +1\}^d$, the product $\rho_t(A) = \prod_{j \in p\Delta A} x_j^t$ takes the values $+1$ and -1 with equal probability. Hence the target is correct on every example, whereas every fixed wrong parity is correct with probability $1/2$ on each independently sampled example.

Following the restriction used by Kim & Suzuki (2025), each policy update is formed from one example at a time. The training rule never performs Gaussian elimination or row reduction on the accumulated parity equations. Information from previous rounds is carried only by the current policy state. This distinction is important because our positive result concerns a sequential change of the policy rather than an external algorithm that solves the parity system before updating the model.

2.4 Transformer model

We use the recursive transformer with a single layer adapted from Kim & Suzuki (2025). Its fixed attention and feedforward maps compute each candidate parity, and a score head defines the policy.

Input representation. Fix a horizon n and $A = \{j_1, \ldots, j_k\} \in P$. For every $j \in [d]$, let

$$\boldsymbol{x}_j = (x_j^1, \ldots, x_j^n)^\top \in \{-1, +1\}^n. \tag{5}$$

Set $L = \lceil \log_2 k \rceil$ and $K = 2^L$. The vectors $\boldsymbol{x}_{j_1}, \ldots, \boldsymbol{x}_{j_k}$ are the leaves of a balanced tree, padded by $K - k$ copies of $\mathbf{1}_n$. Each of its $2K - 1$ nodes has token

$$\boldsymbol{p}_r = \begin{pmatrix} \boldsymbol{h}_r \\ \boldsymbol{e}_r \end{pmatrix}, \qquad \boldsymbol{h}_r = \begin{cases} \boldsymbol{x}_{j_r}, & 1 \le r \le k, \\ \mathbf{1}_n, & k < r \le K, \\ \mathbf{0}_n, & K < r \le 2K - 1, \end{cases} \tag{6}$$

where $\boldsymbol{e}_r \in \mathbb{R}^{2K-1}$ is the r-th standard basis vector, serving as the absolute one-hot node position.

Recursive parity computation. At each level, each parent combines the values of its two children. Using the absolute position vectors, we choose fixed query, key and value matrices W_Q, W_K and W_V so that, under the fixed tree attention structure, a parent node r attends only to its two children, indexed by a and b, with equal weights $1/2$. The value matrix W_V selects the value coordinates. Hence

$$\mathrm{Attn}(\boldsymbol{p}_r) = \frac{\boldsymbol{h}_a + \boldsymbol{h}_b}{2}. \tag{7}$$

We then construct a fixed coordinatewise function ϕ such that

$$\phi\left(\frac{a+b}{2}\right) = ab \quad (a, b \in \{-1, +1\}), \qquad \phi\left(\frac{\boldsymbol{h}_a + \boldsymbol{h}_b}{2}\right) = \boldsymbol{h}_a \odot \boldsymbol{h}_b. \tag{8}$$

Thus one transformer layer replaces the two child values by their coordinatewise product. Applying the same fixed transformer layer recursively for L levels gives the root value

$$\boldsymbol{h}_A = \left(\prod_{j \in A} x_j^1, \ldots, \prod_{j \in A} x_j^n \right)^\top = \big(A(\boldsymbol{x}^1), \ldots, A(\boldsymbol{x}^n)\big)^\top . \tag{9}$$

Program score and policy. The transformer body is fixed and $\boldsymbol{c} = (c_1, \ldots, c_n)^\top$ is trainable. We set

$$S_{\boldsymbol{c}}(A) := \boldsymbol{c}^\top \boldsymbol{h}_A = \sum_{s=1}^{n} c_s A(\boldsymbol{x}^s) \quad \text{and} \quad q_{\boldsymbol{c}}(A) := \frac{e^{S_{\boldsymbol{c}}(A)}}{\sum_{B \in P} e^{S_{\boldsymbol{c}}(B)}}. \tag{10}$$

We initialize $\boldsymbol{c} = \mathbf{0}$, giving the uniform policy. On round t, the model supplies $h_t(A) = A(\boldsymbol{x}^t)$ and only c_t is updated; The proof in Theorem 3 shows that the resulting step exactly realizes VPT using only the final reward.

2.5 Population policy gradient underlying ordinary GRPO

For a training example $(\boldsymbol{x}^t, y^t)$, define the signed reward $\rho_t(A) = y^t A(\boldsymbol{x}^t)$. The expected reward under the current program distribution is

$$J_t(\theta) := \mathbb{E}_{A \sim q_\theta}[\rho_t(A)] = \sum_{A \in P} q_\theta(A) \rho_t(A). \tag{11}$$

Its gradient admits the policy-gradient representation

$$\nabla_\theta J_t(\theta) = \sum_{A \in P} \rho_t(A) \nabla_\theta q_\theta(A) = \mathbb{E}_{A \sim q_\theta}\left[\rho_t(A) \nabla_\theta \log q_\theta(A)\right]. \tag{12}$$

Taking expectation over the input distribution gives the population objective

$$J_{\mathrm{pg}}(\theta) := \mathbb{E}_{\boldsymbol{x}, A \sim q_\theta}[\rho(A; \boldsymbol{x})], \qquad \rho(A; \boldsymbol{x}) := p(\boldsymbol{x}) A(\boldsymbol{x}). \tag{13}$$

We analyze the corresponding population gradient

$$g_{\mathrm{pg}}(\theta) := \nabla_\theta J_{\mathrm{pg}}(\theta) = \mathbb{E}_{\boldsymbol{x}, A \sim q_\theta}\left[\rho(A; \boldsymbol{x}) \nabla_\theta \log q_\theta(A)\right]. \tag{14}$$

Subtracting a baseline $b(\boldsymbol{x})$ that does not depend on the sampled program leaves this direction unchanged, since $\mathbb{E}_{A \sim q_\theta}\left[b(\boldsymbol{x}) \nabla_\theta \log q_\theta(A)\right] = b(\boldsymbol{x}) \nabla_\theta \sum_{A \in P} q_\theta(A) = 0$.

Equation (14) describes the basic population policy-gradient direction underlying GRPO; it does not include finite-group normalization and clipping. Our comparison is between this gradient, whose population direction averages over inputs, and VPT, which updates the program distribution using the current example's reward before processing the next example.

3 Main results

The results below compare two different ways of using the same final verifier signals. We first show that, in ordinary population policy gradient, averaging over inputs cancels the verifier signals associated with all incorrect parity programs before they can be incorporated into the policy, leaving only a target contribution scaled by its current probability. We then apply the VPT update to each example before the wrong-program contributions cancel under population averaging and show that the resulting sequential updates identify the target from $O(d)$ examples in the dense regime.

3.1 Ordinary population policy gradient

The key fact is parity orthogonality. For every fixed wrong program $A \neq p$, the signed reward in (4) has zero expectation over an input. Consequently, once the reward from the current example is averaged before the policy is changed, no wrong program carries a population signal that depends on the target. The only surviving term comes from the target itself.

Theorem 1 (Ordinary population gradient for dense parity). *Let $d \geq k \geq 2$ and P be the collection of subsets of $[d]$ with cardinality k. Let $p \in P$ be the target and suppose q_θ is differentiable, has full support on P and is independent of the input $\boldsymbol{x}$. Then*

$$g_{\mathrm{pg}}(\theta) = q_\theta(p)\nabla_\theta \log q_\theta(p) = \nabla_\theta q_\theta(p). \tag{15}$$

Suppose the dense regime above holds and $q_0 = q_{\theta_0}$ is uniform. If

$$\left\| \nabla_\theta \log q_\theta(A)|_{\theta=\theta_0} \right\|_2 \leq \mathrm{poly}(d) \qquad \textit{for all } A \in P,$$

then

$$\|g_{\mathrm{pg}}(\theta_0)\|_2 \leq \frac{\mathrm{poly}(d)}{\binom{d}{k}}. \tag{16}$$

In particular, for dense parity learning, this upper bound is exponentially small in d.

Remark 1. Theorem 1 also applies to the usual GRPO objective with a KL penalty relative to a reference policy at the uniform initialization. If $q_{\theta_0} = q_{\mathrm{ref}}$ is uniform on P, then $\nabla_\theta D_{\mathrm{KL}}\left(q_\theta \| q_{\mathrm{ref}}\right)\Big|_{\theta=\theta_0} = 0$. Hence, the KL term does not change the population gradient at initialization, and the full GRPO gradient has the same exponentially small target signal identified in Theorem 1. Increasing the number of responses sampled for each prompt can reduce the Monte Carlo error in estimating this gradient, but it does not increase the magnitude of the underlying population signal. This distinction is consistent with recent empirical evidence that increasing the number of completions per prompt in GRPO does not necessarily improve learning on harder problems (Noukhovitch et al., 2026). Thus, polynomially many rollouts do not remove this bottleneck in the dense parity regime.

3.2 Verifier Posterior Transport

Theorem 1 shows that averaging the verifier reward over the input distribution removes the contribution of every wrong program from the population policy gradient. This motivates a different use of the same final reward. Rather than averaging the reward over inputs before changing the policy, we update the current program distribution using the reward observed on each example and then proceed to the next example. Motivated by policy improvement with KL regularization (Peters et al., 2010), we apply the update sequentially to verifier rewards after every training example and call the resulting procedure Verifier Posterior Transport (VPT).

The distinction can be seen directly from the reward. For the target p, $\mathbb{E}_{\boldsymbol{x}}[\rho(p;\boldsymbol{x})] = 1$, whereas for every fixed wrong program $A \neq p$, $\mathbb{E}_{\boldsymbol{x}}[\rho(A;\boldsymbol{x})] = 0$. Consequently, the population policy gradient reduces to $g_{\mathrm{pg}}(\theta) = q_\theta(p)\nabla_\theta \log q_\theta(p)$, so the only remaining contribution is multiplied by the current target probability $q_\theta(p)$. Under the uniform policy, $q_\theta(p) = \frac{1}{\binom{d}{k}}$, which is exponentially small in the dense regime.

On an individual example, however, the reward has not been averaged. At round t, $\rho_t(A) \in \{-1, +1\}$ indicates whether A satisfies the current parity constraint. The target satisfies every constraint, $\rho_t(p) = 1$, while any fixed wrong program satisfies a given constraint with probability $1/2$. VPT uses these realized rewards to change the current policy before the next example is observed. In this way, the effect of each example is carried forward through the updated policy.

Let q_t be the current distribution over programs and let $\beta > 0$. At round t, VPT updates the current program distribution by

$$q_{t+1}(A) := \frac{q_t(A)e^{\beta\rho_t(A)}}{Z_t}, \qquad Z_t := \sum_{B\in P} q_t(B)e^{\beta\rho_t(B)}. \tag{17}$$

Theorem 2 (Verifier Posterior Transport). *Let $\beta > 0$ and let q_t have full support on P. The unique solution of*

$$\max_{q\in\Delta(P)} \left\{ \mathbb{E}_{A\sim q}[\rho_t(A)] - \frac{1}{\beta} D_{\mathrm{KL}}(q\|q_t) \right\} \tag{18}$$

is q_{t+1} in (17). Moreover, for every $A, B \in P$,

$$\log \frac{q_{t+1}(A)}{q_{t+1}(B)} - \log \frac{q_t(A)}{q_t(B)} = \beta\big(\rho_t(A) - \rho_t(B)\big). \tag{19}$$

In particular, if A succeeds and B fails on example t, then

$$\frac{q_{t+1}(A)}{q_{t+1}(B)} = e^{2\beta}\frac{q_t(A)}{q_t(B)}. \tag{20}$$

Thus, one such example multiplies the probability ratio of the successful program to the failed program by $e^{2\beta}$.

Remark 2. VPT does not improve by estimating the ordinary gradient more accurately. Instead, it applies an update regularized by KL divergence. When one program succeeds and another fails, their log odds change by the fixed amount 2β, independent of the current target probability. The contrast is direct: the ordinary target contribution is multiplied by $q_\theta(p)$, whereas VPT multiplies the probability ratio of a successful program to a failed program by $e^{2\beta}$. Repeating this update across examples yields the separation used in Theorem 4. Further details are provided in Appendix A.4.

3.3 Transformer realization

We use VPT to denote the sequential application of (17) to verifier rewards. For the constructed transformer, a parameter update realizes this distributional update exactly, linking VPT to concrete transformer dynamics. The VPT update can be realized by the recursive transformer in Section 2.4 with a particularly simple parameter update. We index the sequential updates by $t = 1, \dots, n$. At step t, the training sample $(\boldsymbol{x}^t, y^t)$ is incorporated into the current program distribution q_t to produce q_{t+1}.

Before the t-th training sample is incorporated, the parameter vector is $\boldsymbol{c}^{(t-1)} = \big(\beta y^1, \dots, \beta y^{t-1}, 0, \dots, 0\big)^\top$ and $q_t(A) := q_{\boldsymbol{c}^{(t-1)}}(A)$. To describe the update of the t-th coordinate, introduce a scalar u and define $\boldsymbol{c}^{(t-1)}(u) = \big(\beta y^1, \dots, \beta y^{t-1}, u, 0, \dots, 0\big)^\top$, together with $q_t(A; u) := q_{\boldsymbol{c}^{(t-1)}(u)}(A)$. Thus, $q_t(A) = q_t(A; 0)$. Let $h_t(A) := A(\boldsymbol{x}^t)$ and $\rho_t(A) = y^t h_t(A) = 2R_t(A) - 1$.

Theorem 3 (Recursive transformer realization of VPT)**.** *Let $\beta > 0$ and define*

$$u_t := \beta y^t, \tag{21}$$

Set $\boldsymbol{c}^{(t)} := \boldsymbol{c}^{(t-1)}(u_t)$. Then the resulting transformer policy $q_{t+1}(A) := q_{\boldsymbol{c}^{(t)}}(A)$ is exactly the VPT update

$$q_{t+1}(A) = \frac{q_t(A)e^{\beta\rho_t(A)}}{\sum_{B\in P} q_t(B)e^{\beta\rho_t(B)}}. \tag{22}$$

Consequently, starting from $\boldsymbol{c}^{(0)} = \mathbf{0}$, and applying this parameter update sequentially for $t = 1, \dots, n$ produces exactly the VPT iterates.

Remark 3. The sequential form of VPT is used to make explicit how each verifier outcome changes the current program distribution. For a fixed collection of training examples, these updates can also be evaluated in batch, since $q_{n+1}(A) \propto q_1(A)\exp\{\beta\sum_{t=1}^n \rho_t(A)\}$. Thus, sequential and batch evaluation give the same final VPT distribution. For the constructed transformer, the batch form corresponds to setting $\boldsymbol{c}^{(n)} = \beta(y^1, \dots, y^n)^\top$.

3.4 Learning dense parity

We now study the sequential effect of the exact VPT update. Iterating (17) gives

$$q_n(A) = \frac{q_0(A)\exp\{\beta\sum_{t=1}^n \rho_t(A)\}}{\sum_{B\in P} q_0(B)\exp\{\beta\sum_{t=1}^n \rho_t(B)\}}. \tag{23}$$

The target has cumulative score n because $\rho_t(p) = 1$ for every t. For a fixed wrong program $A \neq p$, the variables $\rho_t(A)$ are independent Rademacher signs, so its cumulative score is centered. Thus the target separates from all $M - 1$ wrong programs after order $\log M$ examples, where $M = \binom{d}{k}$.

Theorem 4 (Learning dense parity from final rewards)**.** *Let $p \in P$, let $\beta > 0$, let $\delta \in (0, 1)$, assume $M \geq 2$, and let $(\boldsymbol{x}^t, y^t)_{t=1}^n$ be independent samples from (2). Let $(q_t)_{t=0}^n$ be generated by the exact VPT update (17). Suppose q_0 has full support and define*

$$\Gamma_0 := \max_{A,B\in P}\left|\log\frac{q_0(A)}{q_0(B)}\right|. \tag{24}$$

If

$$n \geq 8\left(\log(M-1)+\log\frac{2}{\delta}\right) \quad \text{and} \quad \frac{\beta n}{2} \geq \Gamma_0 + \log\frac{M-1}{\delta}, \tag{25}$$

then, with probability at least $1-\delta$,

$$q_n(p) \geq 1-\delta. \tag{26}$$

In particular, in the dense regime defined in Section 2.2, for constant β *and* $\Gamma_0 = O(d)$*, it is sufficient to take* $n = O\big(d + \log\frac{1}{\delta}\big)$.

Corollary 5. *Assume the dense regime defined in Section 2.2, let* $\beta > 0$ *be constant, and let* $\delta \in (0,1)$. *The recursive transformer in Section 2.4, initialized with* $\boldsymbol{c}^{(0)} = \mathbf{0}$ *and trained by the VPT parameter update (21), satisfies* $q_{\boldsymbol{c}^{(n)}}(p) \geq 1-\delta$ *with probability at least* $1-\delta$ *after* $n = O\big(d + \log\frac{1}{\delta}\big)$ *training examples.*

The corollary follows immediately from Theorems 3 and 4. Together with Theorems 3 and 4, these results give a positive counterpart to Theorem 1: the supervision is still only the binary final reward in (3), but the sequential VPT update preserves the information from each example in the current policy.

4 Numerical Experiments

All experiments use the same tree tokens and final reward protocol described in Section 2.4. Full experimental settings are provided in Appendix A.1.

Experiment 1: Theorems 1 and 2 with a standard trainable transformer. The first experiment investigates whether the optimization separation appears in an ordinary transformer rather than only in the special construction used for the proof of Theorem 3. We use a standard transformer encoder with a single layer and train its input projection, attention matrices, feedforward network, output map and program scores. Both branches start from identical parameters and the uniform policy and receive exactly the same final rewards. In each training round, VPT incorporates the final reward from one new example, whereas the ordinary baseline takes one exact population gradient step. At $d = 14$ and $k = 7$, Figure 1(a) shows that after 40 rounds the target probability under the ordinary gradient changes only from 0.0002921 to 0.0002959, while VPT reaches 0.99991. Thus, the standard transformer reproduces the separation predicted by Theorems 1 and 2: averaging the ordinary update shows an extremely weak target signal, whereas incorporating the current reward through VPT produces a substantial change in program odds. Figure 1(b) validates the conclusion of Theorem 1. Across dimensions, the initial ordinary gradient norm is proportional to $1/\binom{d}{k}$, confirming the predicted exponential decay for dense parity. Consequently, Figures 1(a) and 1(b) provide numerical support for Theorems 1 and 2 in a standard trainable transformer.

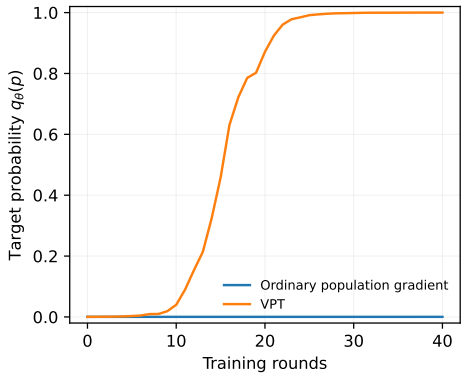
Experiment 2: Theorem 3 with the recursive transformer. Figure 1(c) tests the explicit construction on exactly the same target and examples as Figure 1(a). The constructed transformer reaches target mass 0.99949, and every policy iterate agrees with the exact VPT update. This numerical identity verifies the exact realization asserted by Theorem 3.

Experiment 3: Theorem 4 and Corollary 5. Figure 1(d) measures the number of sequential examples required by the recursive transformer to reach target mass 0.9. For $d \in \{4, 6, 8, 10, 12, 14\}$, all 20 trials per dimension reach the threshold within $8d$ examples, and the mean time is $1.781d + 2.70$. The observed linear scaling supports the $O(d)$ conclusion of Theorem 4 and its transformer consequence in Corollary 5.

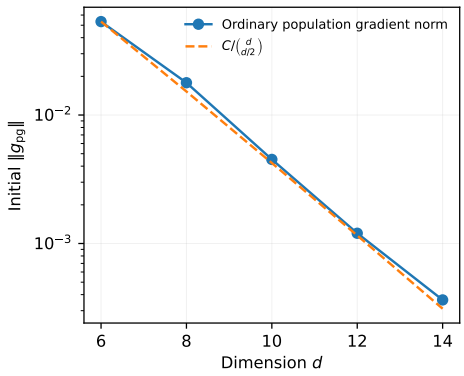
Interpretation. The exact recursive construction proves that a transformer can realize VPT, while the first experiment provides the complementary practical evidence: the same qualitative behavior arises when all components of a standard transformer are trainable. Hence, the positive phenomenon is not only an artifact of a manually specified network. The theorem gives an exact existence result, and Figure 1(a) shows that a usual transformer parameterization can approximate the same update closely enough to obtain the predicted learning behavior.

5 Conclusion

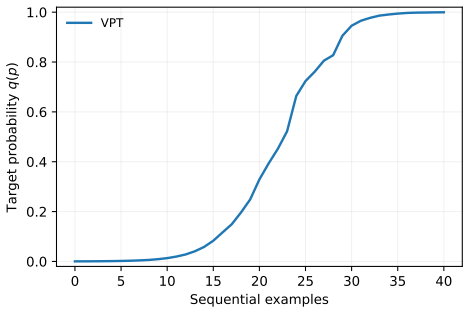
Our analysis highlights the distinction between the information contained in the verifier and the way in which the information enters optimization. For dense parity, the roadblock for ordinary GRPO



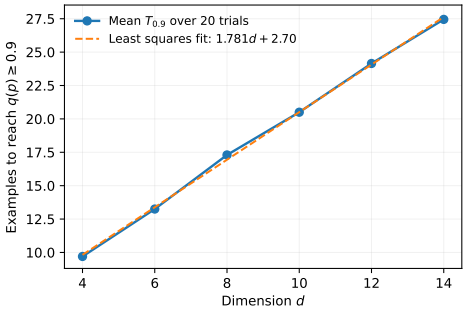
(a) Standard transformer



(b) Gradient scaling



(c) Recursive transformer



(d) Sample scaling

Figure 1: Numerical verification of the main results under the same tree token input rule and final reward protocol. (a) A standard trainable transformer reveals the sharply different behavior between the ordinary gradient and VPT predicted by Theorems 1 and 2. (b) The ordinary gradient decays as the inverse program count, as stated in Theorem 1. (c) The recursive transformer matches exact VPT, verifying Theorem 3 and (d) its sample complexity grows linearly with dimension, supporting Theorem 4 and Corollary 5. In (a), each round denotes one population-gradient update for the ordinary baseline and one example-based update for VPT; in (c), each round corresponds to one sequential training example.

comes from population averaging together with the probability scaling of ordinary policy gradient, rather than from the final correctness signal itself. The constructive result shows that the same signal can support efficient learning when it is incorporated through VPT. Our VPT result points to a broader optimization phenomenon: the exponential probability factor is not intrinsic to the verifier signal, and the same analysis shows that natural policy gradient can avoid this factor as well. The significance of this observation is not confined to parity: this is a case where the verifier contains enough information, yet averaging together with ordinary policy gradient makes the downstream learning signal exponentially weak. This issue may be especially important for difficult reasoning tasks, where correct trajectories occupy only a small fraction of the policy space and positive verifier outcomes are correspondingly rare. This observation reveals that the treatment of verifier information, rather than rollout count alone, is the central question for the theory and design of RLVR.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing of this manuscript, support literature retrieval and discovery and assist with code generation and debugging. All AI-assisted work was reviewed and verified by the authors. In particular, AI-assisted code was verified and tested for correctness and literature references identified with AI assistance were independently checked by the

authors. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REFERENCES

Daniel Barzilai, Yotam Wolf, and Ronen Basri. When is compositional reasoning learnable from verifiable rewards? *arXiv preprint arXiv:2602.07992*, 2026.

Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Advances in Neural Information Processing Systems*, 38:57654–57689, 2026.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in neural information processing systems*, 36:70757–70798, 2023.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024.

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.

Yuhang He, Haodong Wu, Siyi Liu, Hongyu Ge, Hange Zhou, Keyi Wu, Zhuo Zheng, Qihong Lin, Zixin Zhong, and Yongqi Zhang. Rethinking token-level credit assignment in rlvr: A polarity-entropy analysis. *arXiv e-prints*, pp. arXiv–2604, 2026.

Yu Huang, Zixin Wen, Yuejie Chi, Yuting Wei, Aarti Singh, Yingbin Liang, and Yuxin Chen. On the learning dynamics of rlvr at the edge of competence. *arXiv e-prints*, pp. arXiv–2602, 2026.

Michael Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM (JACM)*, 45(6):983–1006, 1998.

Juno Kim and Taiji Suzuki. Transformers provably solve parity efficiently with chain of thought. In *International Conference on Learning Representations*, 2025.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328, 2022.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations*, volume 2024, pp. 11911–11943, 2024.

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 *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations*, volume 2024, pp. 7690–7706, 2024.

Youssef Mroueh. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification. *arXiv preprint arXiv:2503.06639*, 2025.

Michael Noukhovitch, Hamish Ivison, Nathan Lambert, and Aaron Courville. Learning to solve hard problems in rl for llms by never giving up. *arXiv preprint arXiv:2609.13443*, 2026.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pp. 745–750, 2007.

Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 1607–1612, 2010.

Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. *Journal of the ACM (JACM)*, 66(1):1–18, 2018.

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

Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning*, pp. 3067–3075. PMLR, 2017.

Ohad Shamir. Distribution-specific hardness of learning neural networks. *Journal of Machine Learning Research*, 19(32):1–29, 2018.

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, et al. Spurious rewards: Rethinking training signals in rlvr. *arXiv preprint arXiv:2506.10947*, 2025.

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.

Xinyu Tang, Yuliang Zhan, Zhixun Li, Wayne Xin Zhao, Zhenduo Zhang, Zujie Wen, Zhiqiang Zhang, and Jun Zhou. Rethinking sample polarity in reinforcement learning with verifiable rewards. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2928–2954, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, et al. Reinforcement learning for reasoning in large language models with one training example. *Advances in Neural Information Processing Systems*, 38: 122721–122764, 2026.

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.

Stanley Wei and Juno Kim. Provable benefits of rlvr over sft for reasoning models: Learning to backtrack efficiently. *arXiv preprint arXiv:2606.22938*, 2026.

Noam Wies, Yoav Levine, and Amnon Shashua. Sub-task decomposition enables learning in sequence to sequence tasks. *arXiv preprint arXiv:2204.02892*, 2022.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *International Conference on Learning Representations*, volume 2025, pp. 72274–72303, 2025.

Zepeng Zhai, Meilin Chen, Jiaxuan Zhao, Junlang Qian, Lei Shen, and Yuan Lu. Rewards as labels: Revisiting rlvr from a classification perspective. *arXiv preprint arXiv:2602.05630*, 2026.

A Appendix

A.1 Experimental Details

Each trial first fixes its full training horizon n and draws $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n \in \{-1, +1\}^d$. For every candidate $A = \{j_1, \ldots, j_k\}$, we set $K = 2^{\lceil \log_2 k \rceil}$ and create $2K - 1$ tree tokens

$$\boldsymbol{p}_r = \begin{pmatrix} \boldsymbol{h}_r \\ \boldsymbol{e}_r \end{pmatrix}, \qquad \boldsymbol{h}_r = \begin{cases} \boldsymbol{x}_{j_r}, & 1 \le r \le k, \\ \mathbf{1}_n, & k < r \le K, \\ \mathbf{0}_n, & K < r \le 2K - 1, \end{cases}$$

where $\boldsymbol{x}_j = (x_j^1, \ldots, x_j^n)^\top$ and $\boldsymbol{e}_r \in \mathbb{R}^{2K-1}$ is the absolute one-hot encoding of node r. This is exactly the input representation in (5) and (6), including the constant padded leaves and internal node values initialized to zero. All experiments use this representation. Experiments 1 and 2 additionally share the identical target index and sample tensor generated under the same seed (seed $= 2026$); Experiment 3 uses the same sampling and token construction procedure independently at each dimension.

The network in Experiment 1 is intentionally more general than the explicit Transformer in Theorem 3. For each candidate program, the input representation consists of the tree tokens $(\boldsymbol{h}_r^\top, \boldsymbol{e}_r^\top)^\top \in \mathbb{R}^{n+2K-1}$ described in Section 2.4. These tokens are used directly as the Transformer input. The full sequence is passed once through a standard Transformer encoder layer with model dimension $n+2K-1$, one attention head, feedforward dimension $2(n+2K-1)$, GELU activation, normalization before each sublayer and zero dropout. The hidden state at the root token is then mapped to a scalar by a trainable linear output layer and combined with a trainable bias associated with each program. The attention matrices, feedforward network, normalization parameters, output map and program biases are all trainable, while the input tree tokens are fixed and stored as a model buffer. Thus, Experiment 1 does not impose the fixed child mask, manually specified Q, K, V, fixed parity map ϕ, or recursive computation across tree levels used in the proof of Theorem 3.

For Experiment 1, the scalar output weights, scalar output bias, and policy scores are initialized to zero. Therefore, all candidate scores are exactly zero and $q_0(A) = 1/M$ with full support, although the remaining Transformer parameters use their standard random initialization. For Experiments 2 and 3, the recursive Transformer uses the fixed child attention and fixed parity map specified in the proof of Theorem 3; the trainable score vector is initialized as $\boldsymbol{c} = \mathbf{0}$. In every experiment, the complete input matrix is held fixed during a trial. On round t, only the final signed reward vector $\rho_t(A)$ associated with row t is supplied to the update.

Fixed dimension ordinary population gradient and VPT experiment. The main run uses $d = 14$, $k = 7$, $M = 3432$ and horizon $n = 40$. The ordinary population-gradient baseline applies the update $\theta_{t+1} = \theta_t + \eta \nabla_\theta q_{\theta_t}(p)$ with learning rate $\eta = 1$. The VPT branch uses a natural-gradient implementation of the current signed-reward update with $\beta = 0.5$. We solve the resulting linear system approximately, using a small regularization term of 10^{-7} for numerical stability and at most 20 solver iterations per update. Both methods start from identical model parameters and enumerate all M candidate programs instead of sampling a finite rollout group.

For the ordinary-gradient diagnostic, we use $d \in \{6, 8, 10, 12, 14\}$, set $k = d/2$, and run each experiment for $n = 4d$ steps. The measured norms are 0.0533582, 0.0178390, 0.00451361, 0.00120190, and 0.000363809, respectively. Let $M = \binom{d}{d/2}$ denote the number of candidate parity programs. After rescaling each gradient norm by M, we obtain 1.067, 1.249, 1.137, 1.111, and 1.249. These rescaled values remain of order one and vary only moderately across d, indicating that the ordinary gradient norm scales approximately as $1/M$, as illustrated in Figure 1(b).

Exact recursive Transformer experiment. Figure 1(c) reports an experiment with $d = 14$, $k = 7$, and $M = 3432$, conducted for $n = 40$ training rounds. It uses exactly the same target parity and training samples as Figure 1(a). The fixed recursive Transformer evaluates every candidate program through $h_t(A) = A(\boldsymbol{x}^t)$. Starting from $\boldsymbol{c} = \mathbf{0}$, at round t it sets $c_t = \beta y^t$, with $\beta = 0.5$, exactly as in Theorem 3. After 40 rounds, the policy assigns probability 0.99949 to the target program.

Recursive Transformer sample scaling experiment. For Figure 1(d), we consider $d \in \{4, 6, 8, 10, 12, 14\}$ with $k = d/2$ and run 20 independent trials for each dimension. In each trial, the target parity is sampled uniformly from P, the model is initialized with $\boldsymbol{c}^{(0)} = \mathbf{0}$, and training is run for at most $n = 8d$ rounds using the input construction from Section 2.4. At round t, the model observes only the final reward for the current input and applies the exact scalar update from Theorem 3 with $\beta = 0.5$. Training stops when $q_{\boldsymbol{c}^{(n)}}(p) \geq 0.9$ or when the maximum number of rounds is reached. All 20 trials succeed at every dimension. A least-squares linear fit gives a slope of 1.781 and an intercept of 2.70.

A.2 Proof of Theorem 1

Proof. We first compute the population policy gradient exactly and then bound its norm at the uniform initialization.

Since q_θ is independent of the input $\boldsymbol{x}$, (14) gives

$$g_{\mathrm{pg}}(\theta) = \sum_{A \in P} q_\theta(A) \mathbb{E}_{\boldsymbol{x}}[\rho(A; \boldsymbol{x})] \nabla_\theta \log q_\theta(A). \tag{27}$$

For a fixed candidate $A \in P$, the signed reward satisfies

$$\rho(A; \boldsymbol{x}) = p(\boldsymbol{x}) A(\boldsymbol{x}) = \prod_{j \in p \Delta A} x_j. \tag{28}$$

If $A = p$, the product is over the empty set and equals one. Consequently,

$$\mathbb{E}_{\boldsymbol{x}}[\rho(p; \boldsymbol{x})] = 1. \tag{29}$$

If $A \neq p$, choose $j_0 \in p \Delta A$. Conditional on $\boldsymbol{x}_{-j_0}$, the product of the remaining coordinates is fixed, whereas x_{j_0} remains an independent uniform sign. Therefore,

$$\mathbb{E}[\rho(A; \boldsymbol{x}) \mid \boldsymbol{x}_{-j_0}] = \left(\prod_{j \in (p \Delta A) \setminus \{j_0\}} x_j \right) \mathbb{E}[x_{j_0} \mid \boldsymbol{x}_{-j_0}] = 0.$$

By the law of total expectation,

$$\mathbb{E}_{\boldsymbol{x}}[\rho(A; \boldsymbol{x})] = \mathbb{E}_{\boldsymbol{x}_{-j_0}} \left[\mathbb{E}[\rho(A; \boldsymbol{x}) \mid \boldsymbol{x}_{-j_0}] \right] = 0. \tag{30}$$

Combining these two cases yields

$$\mathbb{E}_{\boldsymbol{x}}[\rho(A; \boldsymbol{x})] = \mathbf{1}\{A = p\}. \tag{31}$$

Substituting (31) into (27) leaves only the term $A = p$:

$$g_{\mathrm{pg}}(\theta) = q_\theta(p) \nabla_\theta \log q_\theta(p) = \nabla_\theta q_\theta(p).$$

This proves (15).

At the uniform initialization,

$$q_{\theta_0}(A) = \frac{1}{\binom{d}{k}} \qquad \text{for every } A \in P.$$

Using (15) and the assumed score-gradient bound, we obtain

$$\|g_{\mathrm{pg}}(\theta_0)\|_2 = q_{\theta_0}(p) \left\| \nabla_\theta \log q_\theta(p)|_{\theta=\theta_0} \right\|_2 \leq \frac{\mathrm{poly}(d)}{\binom{d}{k}}. \tag{32}$$

This proves (16).

Finally, we verify that this upper bound is exponentially small in the dense regime. Let $m = \min\{k, d-k\}$. By assumption, $m = \Theta(d)$ and $m \leq d/2$. Symmetry of the binomial coefficients gives

$$\binom{d}{k} = \binom{d}{m} \geq \binom{2m}{m} \geq \frac{2^{2m}}{2m+1}.$$

The last inequality follows because $\binom{2m}{m}$ is the largest of the $2m+1$ terms in $\sum_{r=0}^{2m} \binom{2m}{r} = 2^{2m}$. Hence there exists a constant $a > 0$ such that $\binom{d}{k} \geq e^{ad}$ for all sufficiently large d. Since the numerator in (32) is polynomially bounded, there are constants $C > 0$ and $r \geq 0$ such that

$$\|g_{\mathrm{pg}}(\theta_0)\|_2 \leq C d^r e^{-ad} \leq e^{-ad/2}$$

for all sufficiently large d. This establishes the claimed exponential upper bound and completes the proof. □

A.3 Proof of Theorem 2

Proof. Fix a round t. Recall from (17) that the VPT update is

$$q_{t+1}(A) = \frac{q_t(A)e^{\beta\rho_t(A)}}{Z_t}, \qquad Z_t = \sum_{B\in P} q_t(B)e^{\beta\rho_t(B)}. \tag{33}$$

Since q_t has full support, $q_t(A) > 0$ for every $A \in P$. Also, $e^{\beta\rho_t(A)} > 0$ for every finite β. Therefore

$$q_t(A)e^{\beta\rho_t(A)} > 0, \qquad A \in P.$$

Because P is finite, the normalizing constant satisfies $0 < Z_t < \infty$. It follows from (33) that $q_{t+1}(A) > 0$ for every $A \in P$, so q_{t+1} also has full support.

We first prove that q_{t+1} is the unique optimizer of (18). Let $q \in \Delta(P)$ be arbitrary. Since q_{t+1} has full support, $D_{\mathrm{KL}}(q\|q_{t+1})$ is well defined. Using (33),

$$D_{\mathrm{KL}}(q\|q_{t+1}) = \sum_{A\in P} q(A)\log\frac{q(A)}{q_{t+1}(A)} = \sum_{A\in P} q(A)\log\left(\frac{q(A)Z_t}{q_t(A)e^{\beta\rho_t(A)}}\right). \tag{34}$$

For each $A \in P$,

$$\log\left(\frac{q(A)Z_t}{q_t(A)e^{\beta\rho_t(A)}}\right) = \log\frac{q(A)}{q_t(A)} + \log Z_t - \beta\rho_t(A).$$

Substituting this identity into (34) gives

$$\begin{aligned} D_{\mathrm{KL}}(q\|q_{t+1}) &= \sum_{A\in P} q(A)\log\frac{q(A)}{q_t(A)} + \sum_{A\in P} q(A)\log Z_t - \beta\sum_{A\in P} q(A)\rho_t(A) \\ &= D_{\mathrm{KL}}(q\|q_t) + \log Z_t - \beta\mathbb{E}_{A\sim q}[\rho_t(A)]. \end{aligned} \tag{35}$$

The second equality uses $\sum_{A\in P} q(A) = 1$, and hence $\sum_{A\in P} q(A)\log Z_t = \log Z_t$. Rearranging (35), we obtain

$$\mathbb{E}_{A\sim q}[\rho_t(A)] - \frac{1}{\beta}D_{\mathrm{KL}}(q\|q_t) = \frac{1}{\beta}\log Z_t - \frac{1}{\beta}D_{\mathrm{KL}}(q\|q_{t+1}). \tag{36}$$

The quantity $\frac{1}{\beta}\log Z_t$ is independent of the optimization variable q. Therefore, maximizing the left-hand side of (36) over $q \in \Delta(P)$ is equivalent to minimizing $D_{\mathrm{KL}}(q\|q_{t+1})$ over $q \in \Delta(P)$. By nonnegativity of KL divergence, we have

$$D_{\mathrm{KL}}(q\|q_{t+1}) \geq 0,$$

with equality if and only if

$$q(A) = q_{t+1}(A) \qquad \text{for every } A \in P.$$

Thus the unique maximizer $q^\star$ of (18) satisfies

$$q^\star(A) = q_{t+1}(A) = \frac{q_t(A)e^{\beta\rho_t(A)}}{Z_t} = \frac{q_t(A)e^{\beta\rho_t(A)}}{\sum_{B\in P} q_t(B)e^{\beta\rho_t(B)}}, \qquad A \in P. \tag{37}$$

The final expression in (37) is exactly the update defined in (17). Hence $q^\star = q_{t+1}$, proving the first claim of the theorem.

We next prove the pairwise log odds identity. Let $A, B \in P$. Applying (17) to both programs gives

$$\frac{q_{t+1}(A)}{q_{t+1}(B)} = \frac{q_t(A)e^{\beta\rho_t(A)}/Z_t}{q_t(B)e^{\beta\rho_t(B)}/Z_t} = \frac{q_t(A)e^{\beta\rho_t(A)}}{q_t(B)e^{\beta\rho_t(B)}} = \frac{q_t(A)}{q_t(B)}\exp\left\{\beta\big(\rho_t(A) - \rho_t(B)\big)\right\}. \tag{38}$$

Since both q_t and q_{t+1} have full support, all terms in (38) are positive. Taking logarithms therefore gives

$$\log\frac{q_{t+1}(A)}{q_{t+1}(B)} = \log\frac{q_t(A)}{q_t(B)} + \beta\left(\rho_t(A) - \rho_t(B)\right).$$

Rearranging,

$$\log \frac{q_{t+1}(A)}{q_{t+1}(B)} - \log \frac{q_t(A)}{q_t(B)} = \beta \left(\rho_t(A) - \rho_t(B)\right),$$

which is (19).

Finally, suppose that A succeeds and B fails on example t. Then

$$R_t(A) = 1, \qquad R_t(B) = 0.$$

Since $\rho_t(C) = 2R_t(C) - 1$ for every $C \in P$,

$$\rho_t(A) = 1, \qquad \rho_t(B) = -1.$$

Therefore

$$\beta\big(\rho_t(A) - \rho_t(B)\big) = \beta\big(1 - (-1)\big) = 2\beta.$$

Thus, on any example for which A succeeds and B fails,

$$\frac{q_{t+1}(A)}{q_{t+1}(B)} = e^{2\beta} \frac{q_t(A)}{q_t(B)}.$$

Hence, one such example multiplies the probability ratio between the successful and failed programs by $e^{2\beta}$. Repeating this effect across examples produces an exponential separation in probability ratio. □

A.4 Comparison with ordinary policy gradient

To compare the two updates directly on the same training sample, parameterize the current program distribution by logits,

$$q_t(A) = \frac{e^{z_t(A)}}{\sum_{B \in P} e^{z_t(B)}}.$$

For the t-th training sample, ordinary policy gradient maximizes the expected signed reward

$$J_t := \mathbb{E}_{A \sim q_t}\left[\rho_t(A)\right] = \sum_{A \in P} q_t(A)\rho_t(A). \tag{39}$$

Define the average signed reward under the current program distribution by

$$\overline{\rho}_t := \sum_{B \in P} q_t(B)\rho_t(B).$$

Differentiating (39) with respect to the logit $z_t(A)$ gives

$$\frac{\partial J_t}{\partial z_t(A)} = q_t(A)\left(\rho_t(A) - \overline{\rho}_t\right).$$

Therefore, one ordinary policy gradient step with learning rate η is

$$z_{t+1}(A) = z_t(A) + \eta q_t(A)\left(\rho_t(A) - \overline{\rho}_t\right). \tag{40}$$

Consider the target program p and a wrong program $A \neq p$. The target always satisfies

$$\rho_t(p) = 1.$$

Suppose that A fails on the current training sample, so that

$$\rho_t(A) = -1.$$

The probability ratio before the update is

$$\frac{q_t(p)}{q_t(A)} = e^{z_t(p) - z_t(A)}.$$

After the update,

$$\frac{q_{t+1}(p)}{q_{t+1}(A)} = e^{z_{t+1}(p) - z_{t+1}(A)}.$$

Using (40),

$$z_{t+1}(p) - z_{t+1}(A) = z_t(p) - z_t(A) + \eta q_t(p)\,(1 - \overline{\rho}_t) + \eta q_t(A)\,(1 + \overline{\rho}_t)\,.$$

Hence

$$\log \frac{q_{t+1}(p)}{q_{t+1}(A)} - \log \frac{q_t(p)}{q_t(A)} = \eta q_t(p)\,(1 - \overline{\rho}_t) + \eta q_t(A)\,(1 + \overline{\rho}_t)\,.$$

Equivalently,

$$\frac{q_{t+1}(p)}{q_{t+1}(A)} = \frac{q_t(p)}{q_t(A)} \exp\left\{\eta\left[q_t(p)\,(1 - \overline{\rho}_t) + q_t(A)\,(1 + \overline{\rho}_t)\right]\right\}. \tag{41}$$

At the uniform program distribution,

$$q_t(p) = q_t(A) = \frac{1}{\binom{d}{k}}.$$

Therefore,

$$q_t(p)\,(1 - \overline{\rho}_t) + q_t(A)\,(1 + \overline{\rho}_t) = \frac{1}{\binom{d}{k}}\,(1 - \overline{\rho}_t) + \frac{1}{\binom{d}{k}}\,(1 + \overline{\rho}_t) = \frac{2}{\binom{d}{k}}.$$

Thus (41) reduces to

$$\frac{q_{t+1}(p)}{q_{t+1}(A)} = e^{2\eta/M}\,\frac{q_t(p)}{q_t(A)}. \tag{42}$$

Since the ratio before the update is 1 at the uniform distribution,

$$1 \longmapsto e^{2\eta/M}.$$

For VPT, on the same training sample with $\rho_t(p) = 1$ and $\rho_t(A) = -1$,

$$\frac{q_{t+1}(p)}{q_{t+1}(A)} = \frac{q_t(p)e^{\beta}}{q_t(A)e^{-\beta}} = e^{2\beta}\,\frac{q_t(p)}{q_t(A)}.$$

Thus, at the uniform program distribution, the two updates give

$$\Delta_{\mathrm{PG}} = \frac{2\eta}{\binom{d}{k}}, \qquad \Delta_{\mathrm{VPT}} = 2\beta,$$

where Δ denotes the change in the log probability ratio between the target program and a wrong program that fails on the current training sample. In the dense parity regime,

$$M = \binom{d}{k} = e^{\Theta(d)},$$

so the ordinary policy gradient change is exponentially small near the uniform distribution, whereas the VPT change is independent of M.

A.5 Proof of Theorem 3

Proof. The proof has two parts. We first verify that the recursive transformer produces the feature $h_t(A) = A(\boldsymbol{x}^t)$ for every candidate program. We then compute the scalar parameter update at step t and show that the resulting program distribution is exactly the VPT update.

Step 1: the recursive transformer computes the candidate parity. Fix a candidate $A = \{j_1, \ldots, j_k\} \in P$. Let $L := \lceil \log_2 k \rceil$, $K := 2^L$ and $N := 2K - 1$. We use K leaves in the balanced reduction tree. For $r \le k$, let $\boldsymbol{h}_r^{(0)} := \boldsymbol{x}_{j_r}$, and for $k < r \le K$, let $\boldsymbol{h}_r^{(0)} := \mathbf{1}_n$, where n is the number of samples. Hence every coordinate of every leaf belongs to $\{-1, +1\}$.

For each node r, let $\boldsymbol{e}_r \in \mathbb{R}^N$ denote its absolute one-hot position, and define the token

$$\boldsymbol{p}_r = \begin{pmatrix} \boldsymbol{h}_r \\ \boldsymbol{e}_r \end{pmatrix} \in \mathbb{R}^{n+N}.$$

Let $\pi(r)$ denote the parent of a non-root node r and define $R := \sum_{r \neq \text{root}} \boldsymbol{e}_{\pi(r)} \boldsymbol{e}_r^\top$. Choose the fixed query, key and value matrices $W_Q = [0 \;\; I_N]$, $W_K = [0 \;\; R]$ and $W_V = [I_n \;\; 0]$. Then $W_Q \boldsymbol{p}_r = \boldsymbol{e}_r$, $W_K \boldsymbol{p}_r = \boldsymbol{e}_{\pi(r)}$ and $W_V \boldsymbol{p}_r = \boldsymbol{h}_r$. Now fix a parent node r and let j be one of its children. Then $\pi(j) = r$, so

$$(W_Q \boldsymbol{p}_r)^\top (W_K \boldsymbol{p}_j) = \boldsymbol{e}_r^\top \boldsymbol{e}_{\pi(j)} = \boldsymbol{e}_r^\top \boldsymbol{e}_r = 1.$$

Thus, the two children of a given parent have the same attention logit.

At level $\ell \in \{1, \ldots, L\}$, the r-th parent has two children, indexed by $2r-1$ and $2r$ at level $\ell - 1$. We use the standard masked scaled dot product attention. The fixed attention mask is

$$M_{rj}^{(\ell)} = \begin{cases} 0, & \pi(j) = r \\ -\infty, & \pi(j) \neq r \end{cases}.$$

The corresponding attention logit is

$$s_{rj}^{(\ell)} = \frac{(W_Q \boldsymbol{p}_r)^\top (W_K \boldsymbol{p}_j)}{\sqrt{N}} + M_{rj}^{(\ell)}.$$

If j is one of the two children of r, then $\pi(j) = r$ and, by the construction above, $(W_Q \boldsymbol{p}_r)^\top (W_K \boldsymbol{p}_j) = 1$. Hence the two children have the same logit $1/\sqrt{N}$. For every other node, the mask gives $s_{rj}^{(\ell)} = -\infty$. Therefore, after the softmax, the two children each receive attention weight $1/2$, while all other nodes receive weight zero.

Since $W_V \boldsymbol{p}_j = \boldsymbol{h}_j$, the attention output is

$$\boldsymbol{z}_r^{(\ell)} = \frac{\boldsymbol{h}_{2r-1}^{(\ell-1)} + \boldsymbol{h}_{2r}^{(\ell-1)}}{2}. \tag{43}$$

We now construct the fixed coordinatewise map ϕ. Define

$$\phi(z) = \begin{cases} -4z^2 - 8z - 3, & -1 \leq z \leq -\frac{1}{2} \\ 4z^2 - 1, & -\frac{1}{2} < z < \frac{1}{2} \\ -4z^2 + 8z - 3, & \frac{1}{2} \leq z \leq 1 \end{cases}.$$

Since $\phi(-1) = 1$, $\phi(0) = -1$ and $\phi(1) = 1$, for any $a, b \in \{-1, +1\}$, we have $\phi(\frac{a+b}{2}) = ab$. Applying this identity coordinatewise to (43) gives

$$\boldsymbol{h}_r^{(\ell)} = \phi\left(\boldsymbol{z}_r^{(\ell)}\right) = \boldsymbol{h}_{2r-1}^{(\ell-1)} \odot \boldsymbol{h}_{2r}^{(\ell-1)}, \tag{44}$$

where $\odot$ denotes coordinatewise multiplication.

We now identify the vector stored at every node. We claim that for every level ℓ and parent index r,

$$\boldsymbol{h}_r^{(\ell)} = \bigodot_{u=(r-1)2^\ell+1}^{r2^\ell} \boldsymbol{h}_u^{(0)}. \tag{45}$$

For $\ell = 1$, this follows directly from (44). Suppose (45) holds at level $\ell - 1$. Then

$$\boldsymbol{h}_r^{(\ell)} = \boldsymbol{h}_{2r-1}^{(\ell-1)} \odot \boldsymbol{h}_{2r}^{(\ell-1)} = \left(\bigodot_{u=(2r-2)2^{\ell-1}+1}^{(2r-1)2^{\ell-1}} \boldsymbol{h}_u^{(0)} \right) \odot \left(\bigodot_{u=(2r-1)2^{\ell-1}+1}^{2r2^{\ell-1}} \boldsymbol{h}_u^{(0)} \right) = \bigodot_{u=(r-1)2^\ell+1}^{r2^\ell} \boldsymbol{h}_u^{(0)}.$$

Thus the claim follows by induction.

At the root, $\ell = L$ and $r = 1$, so

$$\boldsymbol{h}_1^{(L)} = \bigodot_{u=1}^{K} \boldsymbol{h}_u^{(0)} = \bigodot_{r=1}^{k} \boldsymbol{x}_{j_r}, \tag{46}$$

because the padded leaves are all equal to $\mathbf{1}_n$.

The s-th coordinate of (46) is

$$\left[\boldsymbol{h}_1^{(L)}\right]_s = \prod_{r=1}^{k} x_{j_r}^s = \prod_{j\in A} x_j^s = A(\boldsymbol{x}^s).$$

Therefore the root vector is

$$\boldsymbol{h}_A = \left(A(\boldsymbol{x}^1), \ldots, A(\boldsymbol{x}^n)\right)^\top.$$

In particular, the feature used at step t is exactly

$$h_t(A) = A(\boldsymbol{x}^t). \tag{47}$$

Step 2: the scalar VPT parameter update. We index the sequential updates by $t = 1, \ldots, n$. At step t, the training sample $(\boldsymbol{x}^t, y^t)$ is incorporated into the current program distribution q_t to produce q_{t+1}.

Before the tth training sample is incorporated, write the current transformer parameter vector as

$$\boldsymbol{c}^{(t-1)} = (u_1, \ldots, u_{t-1}, 0, \ldots, 0)^\top,$$

where $u_1, \ldots, u_{t-1}$ are the parameter values obtained from the previous updates. The corresponding program distribution is

$$q_t(A) := q_{\boldsymbol{c}^{(t-1)}}(A). \tag{48}$$

To isolate the update of the tth coordinate, introduce a scalar u and define

$$\boldsymbol{c}^{(t-1)}(u) = (u_1, \ldots, u_{t-1}, u, 0, \ldots, 0)^\top.$$

Let

$$q_t(A; u) := q_{\boldsymbol{c}^{(t-1)}(u)}(A).$$

Thus

$$q_t(A) = q_t(A; 0).$$

By the softmax policy in (10), changing only the tth coordinate from 0 to u changes the score of candidate A from $S_{\boldsymbol{c}^{(t-1)}}(A)$ to

$$S_{\boldsymbol{c}^{(t-1)}}(A) + u h_t(A).$$

Therefore

$$q_t(A; u) = \frac{\exp\left\{S_{\boldsymbol{c}^{(t-1)}}(A) + u h_t(A)\right\}}{\sum_{B\in P} \exp\left\{S_{\boldsymbol{c}^{(t-1)}}(B) + u h_t(B)\right\}} = \frac{q_t(A) e^{u h_t(A)}}{Z_t(u)}, \tag{49}$$

where

$$Z_t(u) := \sum_{B\in P} q_t(B) e^{u h_t(B)}.$$

Using (4) and (47),

$$\rho_t(A) = y^t h_t(A). \tag{50}$$

By the parameter update in (21),

$$u_t = \beta y^t. \tag{51}$$

The updated transformer parameter vector is

$$\boldsymbol{c}^{(t)} = \boldsymbol{c}^{(t-1)}(u_t).$$

Hence, using (49),

$$q_{t+1}(A) := q_{\boldsymbol{c}^{(t)}}(A) = q_t(A; u_t) = \frac{q_t(A) e^{\beta y^t h_t(A)}}{\sum_{B\in P} q_t(B) e^{\beta y^t h_t(B)}} = \frac{q_t(A) e^{\beta \rho_t(A)}}{\sum_{B\in P} q_t(B) e^{\beta \rho_t(B)}}. \tag{52}$$

The last equality follows from (50). Equation (52) is exactly the VPT update (22).

Step 3: sequential application. The preceding calculation applies at every step $t = 1, \dots, n$. Starting from

$$\boldsymbol{c}^{(0)} = \mathbf{0},$$

all transformer scores are zero. Hence

$$q_1(A) = \frac{1}{\binom{d}{k}}$$

for every $A \in P$, which is the uniform initial program distribution.

Suppose that after the first $t-1$ training samples have been incorporated, the transformer parameter vector is $\boldsymbol{c}^{(t-1)}$ and the corresponding program distribution as in (48)

$$q_t(A) = q_{\boldsymbol{c}^{(t-1)}}(A)$$

is exactly the current VPT iterate.

At step t, (52) of Step 2 shows that the scalar parameter update produces $\boldsymbol{c}^{(t)}$ and changes the transformer policy from q_t to

$$q_{t+1}(A) = \frac{q_t(A) e^{\beta \rho_t(A)}}{\sum_{B \in P} q_t(B) e^{\beta \rho_t(B)}}.$$

This is exactly the next VPT iterate. Therefore, by induction over $t = 1, \dots, n$, every transformer policy produced by the sequential parameter updates coincides with the corresponding exact VPT iterate. This proves Theorem 3.

□

A.6 Proof of Theorem 4

Proof. We first describe the cumulative reward of the target and of every fixed wrong parity, and then use Theorem 2 to convert this reward separation into a probability bound.

For any $A \in P$, (4) gives

$$\rho_t(A) = y^t A(\boldsymbol{x}^t) = \prod_{j \in p \Delta A} x_j^t. \tag{53}$$

For the target $A = p$, the symmetric difference is empty, so

$$\rho_t(p) = 1 \qquad \text{for every } t. \tag{54}$$

Consequently the target cumulative score after n rounds is exactly

$$\sum_{t=1}^{n} \rho_t(p) = n.$$

Now fix one wrong program $A \neq p$. Set $S := p \Delta A$, which is nonempty, and choose $j_0 \in S$. Conditional on all coordinates of $\boldsymbol{x}^t$ except $x_{j_0}^t$,

$$\rho_t(A) = x_{j_0}^t \prod_{j \in S \setminus \{j_0\}} x_j^t.$$

The second factor is fixed under this conditioning, while $x_{j_0}^t$ is an independent uniform sign. Therefore

$$\mathbb{P}\{\rho_t(A) = 1 \mid \boldsymbol{x}_{-j_0}^t\} = \mathbb{P}\{\rho_t(A) = -1 \mid \boldsymbol{x}_{-j_0}^t\} = \frac{1}{2}.$$

Averaging over $\boldsymbol{x}_{-j_0}^t$ gives

$$\mathbb{P}\{\rho_t(A) = 1\} = \mathbb{P}\{\rho_t(A) = -1\} = \frac{1}{2}. \tag{55}$$

Thus $\rho_t(A)$ is a Rademacher random variable. Moreover, the examples $\boldsymbol{x}^1, \dots, \boldsymbol{x}^n$ are independent, so for this fixed A the random variables $\rho_1(A), \dots, \rho_n(A)$ are independent. No independence across different wrong programs is needed below.

Define

$$U_n(A) := \sum_{t=1}^{n} \rho_t(A). \tag{56}$$

By Hoeffding's inequality for a sum of n independent random variables taking values in $[-1, 1]$ with mean zero, for every $u > 0$,

$$\mathbb{P}\{U_n(A) \geq u\} \leq \exp\left\{-\frac{u^2}{2n}\right\}. \tag{57}$$

There are $M - 1$ wrong programs. Applying the union bound to (57),

$$\mathbb{P}\left\{\max_{A \neq p} U_n(A) \geq u\right\} \leq \sum_{A \neq p} \mathbb{P}\{U_n(A) \geq u\} \leq (M-1)\exp\left\{-\frac{u^2}{2n}\right\}. \tag{58}$$

Choose

$$u := \sqrt{2n\left(\log(M-1) + \log\frac{2}{\delta}\right)}. \tag{59}$$

Substituting (59) into (58), we have

$$\mathbb{P}\left\{\max_{A \neq p} U_n(A) \geq u\right\} \leq (M-1)\exp\left\{-\log(M-1) - \log\frac{2}{\delta}\right\} = \frac{\delta}{2}. \tag{60}$$

Let

$$L := \log(M-1) + \log\frac{2}{\delta}.$$

By assumption $n \geq 8L$ in (25) of Theorem 4, we get $L \leq n/8$. Then we have $u^2 = 2nL \leq \frac{n^2}{4}$, which implies $u \leq n/2$. It follows from (60) that the event

$$\mathcal{E} := \left\{\max_{A \neq p} U_n(A) \leq \frac{n}{2}\right\} \tag{61}$$

has probability at least $1 - \delta/2$.

We now work on the event $\mathcal{E}$. The pairwise log odds identity in Theorem 2, applied to A and the target p, gives at round t

$$\log\frac{q_{t+1}(A)}{q_{t+1}(p)} - \log\frac{q_t(A)}{q_t(p)} = \beta(\rho_t(A) - \rho_t(p)).$$

Summing this equality from $t = 1$ to n telescopes the left hand side

$$\log\frac{q_n(A)}{q_n(p)} - \log\frac{q_0(A)}{q_0(p)} = \beta\sum_{t=1}^{n}(\rho_t(A) - \rho_t(p)) = \beta(U_n(A) - n), \tag{62}$$

where the last equality uses (54) and (56). Rearranging,

$$\log\frac{q_n(A)}{q_n(p)} = \log\frac{q_0(A)}{q_0(p)} + \beta(U_n(A) - n). \tag{63}$$

By the definition of Γ_0 in (24) of Theorem 4,

$$\log\frac{q_0(A)}{q_0(p)} \leq \Gamma_0.$$

On $\mathcal{E}$, $U_n(A) \leq n/2$ for every wrong A. Hence

$$\log\frac{q_n(A)}{q_n(p)} \leq \Gamma_0 + \beta\left(\frac{n}{2} - n\right) = \Gamma_0 - \frac{\beta n}{2}. \tag{64}$$

By assumption $\frac{\beta n}{2} \geq \Gamma_0 + \log\frac{M-1}{\delta}$ in (25) of Theorem 4 and (64), we deduce that, for every $A \neq p$, $\log\frac{q_n(A)}{q_n(p)} \leq -\log\frac{M-1}{\delta}$, which implies

$$\frac{q_n(A)}{q_n(p)} \leq \frac{\delta}{M-1}. \tag{65}$$

Summing (65) over the $M-1$ wrong programs, we have that

$$\frac{1-q_n(p)}{q_n(p)} = \sum_{A \neq p} \frac{q_n(A)}{q_n(p)} \leq (M-1)\frac{\delta}{M-1} = \delta. \tag{66}$$

Equation (66) implies $1 - q_n(p) \leq \delta q_n(p)$, so $q_n(p) \geq \frac{1}{1+\delta}$. Since $\delta > 0$,

$$\frac{1}{1+\delta} - (1-\delta) = \frac{\delta^2}{1+\delta} \geq 0$$

and therefore

$$q_n(p) \geq 1 - \delta.$$

This conclusion holds on $\mathcal{E}$, whose probability is at least $1 - \delta/2$ by (60). In particular, it holds with probability at least $1 - \delta$, as claimed.

Finally, we verify the stated sample scaling. In the dense regime, $\log M = \Theta(d)$. Condition (25) requires

$$n = O\left(\log M + \log \frac{1}{\delta}\right) = O\left(d + \log \frac{1}{\delta}\right).$$

For constant β and $\Gamma_0 = O(d)$, Condition (25) requires

$$n \geq \frac{2}{\beta}\left(\Gamma_0 + \log(M-1) + \log \frac{1}{\delta}\right) = O\left(d + \log \frac{1}{\delta}\right).$$

Choosing the constant in the $O(\cdot)$ notation large enough satisfies both conditions. This proves the final statement of Theorem 4. □

A.7 Proof of Corollary 5

Proof. At initialization $\boldsymbol{c}^{(n)} = \mathbf{0}$, (10) gives $S_{\boldsymbol{c}^{(n)}}(A) = 0$ for every $A \in P$. Hence

$$q_0(A) = \frac{1}{\binom{d}{k}} \qquad \text{for every } A \in P,$$

so q_0 has full support and $\Gamma_0 = 0$. By Theorem 3, every VPT parameter update (21) produces exactly the corresponding VPT update. Therefore, the sequence of transformer policies satisfies all assumptions of Theorem 4 with $\Gamma_0 = 0$. In the dense regime and for constant β, Theorem 4 gives $q_{\boldsymbol{c}^{(n)}}(p) \geq 1 - \delta$ with probability at least $1 - \delta$ after $n = O\left(d + \log \frac{1}{\delta}\right)$ training examples. This is exactly the claim of Corollary 5. □