ACES: WHO TESTS THE TESTS? LEAVE-ONE-OUT AUC CONSISTENCY FOR CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Selecting LLM-generated code candidates using LLM-generated tests is challenging because the tests themselves may be incorrect. Existing methods treat all tests equally or use ad-hoc heuristics to filter unreliable tests. Yet determining test correctness requires knowing which codes are correct, creating a *circular dependency*. Our key insight is that we need not determine test correctness at all: *test votes should rank, not merely count*. What matters is not how many codes pass a test, but whether it can *distinguish* correct from incorrect code. We break the circular dependency via leave-one-out evaluation: hold out one test, rank codes by their aggregate scores on all remaining tests, and measure whether the held-out test's pass/fail pattern agrees with this ranking. We formalize this agreement as the leave-one-out AUC (LOO-AUC) and, under a class-conditional model, derive an exact link between the expected pass-rate-corrected score and each test's ability to separate correct from incorrect code. Building on this, we propose **ACES** (**A**UC **C**onsist**E**ncy **S**coring) with two complementary variants: ACES-C provides closed-form weights with a population-level oracle-approximation bound under a mild assumption on average test quality; ACES-O drops this assumption and iteratively optimizes a differentiable LOO-AUC objective. Both operate solely on the binary pass matrix with negligible overhead, and achieve state-of-the-art Pass@k on multiple code generation benchmarks.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong capabilities in code generation (Chen et al., 2021; Guo et al., 2024; Liu et al., 2025; Du et al., 2025), yet individual generations are not always correct. A promising strategy is to scale test-time computation (Snell et al., 2025; Brown et al., 2024; Wu et al., 2025): generate many candidate solutions together with test cases, and select the best candidates based on test execution (Li et al., 2022; Chen et al., 2023). The central challenge is that *neither the code nor the tests are guaranteed to be correct*: we need reliable tests to judge code quality, and reliable code to judge test quality, but have neither.

Existing methods either treat all tests equally via majority voting or rely on heuristics: CodeT (Chen et al., 2023) scores consensus sets by size without weighting individual tests, while MBR-exec (Minimum Bayes Risk) (Shi et al., 2022) and SRank (To et al., 2024) require pairwise output comparison beyond the binary pass matrix. More heavyweight approaches co-evolve code and tests via reinforcement learning (Wang et al., 2025) or evolutionary search (Li et al., 2025b), requiring substantially more computation. Yet the *circular dependency* between code and test quality remains: to our knowledge, no existing method offers formal guarantees for identifying reliable tests without knowing which codes are correct.

What should such a criterion measure? We observe that code selection is fundamentally a ranking problem: we must order candidates so that correct solutions appear near the top. Our key insight is that, for this ranking task, a test's value lies not in its correctness but in its ability to *distinguish* correct code from incorrect code: a trivially correct test that all codes pass offers no ranking signal, while a demanding test that separates candidates is valuable even if imperfect. *Test votes should rank, not merely count.* Under uniform counting, however, this distinction is lost: easy tests that most codes pass dilute the ranking signal, while tests that favor incorrect codes actively corrupt it.

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.

How can we measure a test's ability to distinguish codes without knowing which codes are correct? We break the circular dependency via *leave-one-out evaluation among tests themselves*. Hold out one test; the remaining tests, aggregated using their weights, rank the code candidates. When this ranking favors correct code, agreement with the held-out test provides evidence of informativeness. Contradiction suggests a potentially misleading test; lack of correlation suggests little ranking signal. This evaluation requires no knowledge of code correctness; it exploits only the internal structure of the pass matrix.

We formalize this as the leave-one-out AUC (LOO-AUC): the area under the ROC curve with the remaining tests' aggregate scores as predictions and each held-out test's pass/fail column as the label. A test contributes to ranking to the extent that correct codes are more likely to pass it than incorrect codes; we call this difference the test's *discriminative power*. This quantity is latent, since code correctness is unknown. Yet LOO-AUC requires no knowledge of code correctness. Under a class-conditional model, we link population LOO-AUC to discriminative power (Theorem 3) and prove an exact expectation identity for the pass-rate-corrected empirical score (Theorem 6). This provides the theoretical basis for principled non-uniform test weighting.

Building on this, we propose *ACES* (AUC Consistency Scoring) with two complementary variants. *ACES-C* applies the pass-rate correction in closed form under uniform weighting, with a population-level oracle-approximation bound under a mild assumption on average test quality (Corollary 8). *ACES-O* instead iteratively optimizes the test weights through a differentiable LOO-AUC objective, without requiring this assumption. Both operate solely on the binary pass matrix with negligible overhead. The two are complementary: the average-quality assumption holds for the majority of tasks, and in this regime ACES-C's one-shot correction targets the oracle weights; for more challenging tasks where it fails, ACES-O's iterative optimization remains effective.

Our main contributions are as follows:

- **Theoretical foundation.** We introduce a population LOO-AUC identity (Theorem 3) and an exact expectation formula for the empirical quality score (Theorem 6). The expected score's sign distinguishes informative from misleading tests under the stated conditions.
- **Algorithms.** Building on this identity, we propose two lightweight algorithms. ACES-C provides closed-form weights with a population-level oracle-approximation bound (Corollary 8); ACES-O iteratively optimizes test weights via a differentiable LOO-AUC objective without requiring the average-quality assumption.
- **Empirical results.** We show that ACES advances the state-of-the-art in Pass@k on multiple benchmarks, with ACES-O leading as a standalone method and ACES-C excelling as a plug-and-play execution-only scorer when combined with other pre-filtering mechanisms.

2 Theoretical Foundations

We formalize code ranking as a weighted voting problem over the pass matrix and develop the theoretical tools that motivate ACES.

2.1 Problem Setup

Given a programming problem, an LLM generates n candidate solutions $C = \{c_1, \ldots, c_n\}$ and m test cases $T = \{t_1, \ldots, t_m\}$. Executing every candidate on every test yields the *pass matrix*

$$B \in \{0,1\}^{n \times m}, \qquad B_{ij} = \mathbf{1}[c_i \text{ passes } t_j]. \tag{1}$$

Each code has an unknown correctness label $y_i \in \{0,1\}$; we write $n^+ = |\{i : y_i = 1\}|$ and $n^- = n - n^+$. Test quality is also unknown: some tests may have incorrect expected outputs. Since test reliability varies, we parameterize code ranking by a weight vector over tests: $w \in \Delta^m = \{w \in \mathbb{R}_+^m : \sum_j w_j = 1\}$ induces scores $s_i(w) = \sum_{j=1}^m w_j B_{ij}$. Codes are ranked by $s_i(w)$ in descending order (ties broken uniformly at random). We evaluate with Pass@k, the probability that at least one correct code appears in the top k:

$$\text{Pass@}k(w) = P\left(\min_{i:\, y_i = 1} \text{rank}(c_i) \le k \right). \tag{2}$$

Table 1: Vote types for a test t_j on a correct-incorrect pair (c^+, c^-). Each test casts a vote $h_j = B_{c^+,j} - B_{c^-,j}$ on their ordering; the score difference $s_{c^+} - s_{c^-} = \sum_j w_j h_j$ is the weighted aggregate.

Type	Condition	h_j	Effect
Informative	c^+ pass, c^- fail	$+1$	Raises $s_{c^+} - s_{c^-}$
Uninformative	Both same	0	No effect
Misleading	c^- pass, c^+ fail	-1	Lowers $s_{c^+} - s_{c^-}$

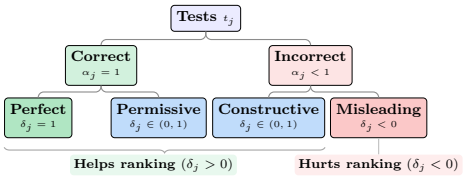


Figure 1: **For ranking, only δ_j matters.** Taxonomy of tests by correctness (α_j) and discriminative power (δ_j).

The simplest baseline, *majority voting*, sets $w_{\text{unif}} = (1/m, \ldots, 1/m)$, ranking codes by total passes.

Pairwise analysis. To understand how w affects ranking, consider a correct code c^+ and an incorrect code c^-. Each test t_j casts a *vote* $h_j = B_{c^+,j} - B_{c^-,j} \in \{+1, 0, -1\}$ on their ordering (Table 1), and the score difference $s_{c^+}(w) - s_{c^-}(w) = \sum_j w_j h_j$ is the weighted sum of these votes: code ranking is a *weighted voting problem*. Averaging over all such pairs, the probability that w places the correct code above the incorrect one defines the *true AUC*:

$$A(w) \;=\; P(s_{c^+}(w) > s_{c^-}(w)) \;+\; \frac{1}{2}\,P(s_{c^+}(w) = s_{c^-}(w)). \tag{3}$$

Since $A(w)$ depends on unknown code labels, it cannot be computed directly and appears only in the analysis. Under uniform weighting, misleading tests contribute as much as informative ones, actively lowering $A(w)$ and corrupting the ranking. Constant columns of B do not affect rankings and are removed in computation; their empirical quality scores are defined as zero.

2.2 Discriminative Power and Pass@k Bound

For analysis, fix labels with $n^+, n^- > 0$ and let $\pi = n^+/n$. We assume independent candidate rows and, within each row, conditionally independent test outcomes with the class-conditional rates below. The fixed-weight analysis takes w independently of the evaluated matrix B.

Definition 1 (Discriminative Power)**.** *For each test t_j, define the* class-conditional pass rates *and* discriminative power*:*

$$\alpha_j = P(B_{ij} = 1 \mid y_i = 1), \qquad \beta_j = P(B_{ij} = 1 \mid y_i = 0), \qquad \delta_j = \alpha_j - \beta_j.$$

A test is informative *($\delta_j > 0$),* uninformative *($\delta_j = 0$), or* misleading *($\delta_j < 0$) according to whether correct codes are more, equally, or less likely to pass it than incorrect ones.*

Figure 1 refines this classification by test correctness: correct tests ($\alpha_j = 1$) have $\delta_j \geq 0$, while incorrect tests ($\alpha_j < 1$) may be constructive ($\delta_j > 0$) or misleading ($\delta_j < 0$). Prior correctness-based filters discard constructive tests together with misleading ones, losing valuable ranking signal; for ranking, only δ_j matters. Equivalently, $\delta_j = \mathbb{E}[h_j]$ is the expected pairwise vote. Pass@k is controlled by a signal-to-noise ratio determined by w and $\{\delta_j\}$:

Theorem 2 (Pass@k Bound)**.** *Define the* mean signal *and* signal-to-noise ratio *of weights w:*

$$M(w) \;:=\; \sum_j w_j \delta_j, \qquad R(w) \;:=\; \frac{M(w)^2}{\sum_j w_j^2}.$$

For any fixed w with $M(w) > 0$ and any $k \geq 1$:

$$\textit{Pass@}k(w) \;\geq\; 1 - \frac{n^-}{k}\,\exp\!\left(-\frac{R(w)}{2}\right), \tag{4}$$

where $n^- = n - n^+$ is the number of incorrect codes. Over non-negative weights with $M(w) > 0$, $R(w)$ is maximized by $w_j^* \propto \max(0,\ \delta_j)$.

The proof is in Appendix A.2. The bound improves exponentially with $R(w)$, but the oracle-optimal weights w^* require knowing the unknown discriminative powers δ_j.

Majority voting baseline. Writing $\bar{\delta} := (1/m)\sum_j \delta_j > 0$, uniform weights give $R(w_{\text{unif}}) = m\bar{\delta}^2$ and hence:

$$\text{Pass@}k(w_{\text{unif}}) \;\geq\; 1 - \frac{n^-}{k}\,\exp\!\left(-\frac{m\bar{\delta}^2}{2}\right). \tag{5}$$

Non-uniform weights can improve this bound by increasing $R(w)$ beyond $m\bar{\delta}^2$, but the optimal weights w^* are unknown in practice.

2.3 Estimating Test Quality for Ranking: LOO-AUC Identity

The discriminative powers δ_j are unknown, but the pass matrix B provides m surrogate label vectors, one per test column. An informative test's column correlates with the unknown y, while a misleading test's column anti-correlates. To distinguish the two, one can hold out test t_j, rank codes by their aggregate scores on the remaining tests, and measure whether t_j's pass/fail pattern agrees with that ranking. Formally, the *leave-one-out scores* and *LOO-AUC* are:

$$S_i^{(-j)} = \sum_{j' \neq j} w_{j'}\, B_{ij'}, \qquad \text{LOO-AUC}_j(w) = \text{AUC}\Big(S^{(-j)},\; B_{:,j}\Big), \tag{6}$$

where $S^{(-j)}$ serves as scores and $B_{:,j}$ as binary labels. Both are computed from the pass matrix B alone; **no knowledge of y is required**. LOO-AUC_j measures the *consistency* between test t_j and the remaining tests: the probability that, among codes distinguished by t_j, those passing it are ranked higher by the remaining tests. A high value ($\text{LOO-AUC}_j > 1/2$) indicates agreement with the consensus ranking; a low value ($\text{LOO-AUC}_j < 1/2$) indicates disagreement. For analysis, $\mathcal{A}_j(w)$ denotes the corresponding *population* LOO-AUC: compare independent passer and failer draws from the class mixture with correct-code probability π.

Theorem 3 (LOO-AUC Identity)**.** *Let $A^{(-j)}(w)$ denote the true AUC using all tests except t_j. For fixed w and marginal pass rate $p_j = \pi\alpha_j + (1-\pi)\beta_j \in (0,1)$:*

$$\mathcal{A}_j(w) - \frac{1}{2} \;=\; c_j(w)\cdot\delta_j, \qquad c_j(w) \;:=\; \frac{\pi(1-\pi)}{p_j(1-p_j)}\left(A^{(-j)}(w) - \frac{1}{2}\right), \tag{7}$$

The proof is in Appendix A.3. For $p_j \in \{0,1\}$, $\delta_j = 0$; set $\mathcal{A}_j = 1/2$ and $c_j = 0$ by convention. The coefficient $c_j(w)$ has two test-dependent factors: the inverse pass-rate variance $1/[p_j(1-p_j)]$ and the leave-one-out ranking quality $A^{(-j)}(w) - 1/2$. Section 3 uses this identity to construct empirical weights and establishes an exact expectation identity for their underlying quality scores.

3 ACES: AUC Consistency Scoring

We use the LOO-AUC Identity (Theorem 3) to construct non-uniform test weights that improve Pass@k through two complementary approaches: ACES-C applies a closed-form pass-rate correction, while ACES-O optimizes weights via differentiable LOO-AUC.

3.1 ACES-C: Closed-Form Weighting

The LOO-AUC Identity (Theorem 3) gives $\mathcal{A}_j(w) - 1/2 = c_j(w)\cdot\delta_j$. When $c_j(w) > 0$, the population LOO-AUC excess shares the sign of δ_j, linking consistency to whether a test is informative or misleading. This holds whenever $A^{(-j)}(w) > 1/2$, i.e., the remaining tests rank correct codes higher, the natural regime since the complement would mean the aggregate ranking is no better than random. To guarantee this for all j under uniform weights, we introduce:

Assumption 4. *The test pool has sufficiently positive average discriminative power:*

$$\bar{\delta} \;:=\; \frac{1}{m}\sum_{j=1}^{m}\delta_j \;>\; 2\sqrt{\frac{\ln 2}{m}}.$$

This is our only assumption beyond the model in Section 2.2; the threshold is $O(1/\sqrt{m})$, and it requires informative signal only *on average*, permitting arbitrarily many misleading tests.

Proposition 5 (Structure of $c_j(w_{\text{unif}})$)**.** *Under Assumption 4, $A^{(-j)}(w_{\text{unif}}) > 1/2$ for all j, so $c_j(w_{\text{unif}}) > 0$ whenever $0 < p_j < 1$. Moreover, $|A^{(-j)} - A^{(-k)}| \le \exp(-(m\bar{\delta}-1)^2/[2(m-1)])$.*

The proof and quantitative bound are in Appendix A.5. Under uniform weights, the variation in $A^{(-j)}(w_{\text{unif}}) - 1/2$ across tests vanishes as m grows at fixed positive $\bar{\delta}$. The factor $\pi(1-\pi)$ is shared by all tests, whereas $1/[p_j(1-p_j)]$ introduces a test-dependent scaling. Multiplying the population LOO-AUC excess by $p_j(1-p_j)$ removes this factor, motivating the empirical correction:

$$w_j \;=\; \max\!\left(0,\ \text{LOO-AUC}_j(w_{\text{unif}}) - \frac{1}{2}\right) \cdot \hat{p}_j(1-\hat{p}_j), \tag{8}$$

where $\hat{p}_j = \frac{1}{n}\sum_{i=1}^{n} B_{ij}$ is the empirical pass rate of test j; constant columns receive zero weight. The $\max(0,\cdot)$ ensures non-negative weights, assigning zero weight to tests with $\text{LOO-AUC}_j \le 1/2$ and filtering out those identified as misleading. Since only relative magnitudes affect the induced ranking, normalization is unnecessary. The corrected empirical score has the following exact expectation under our model.

Theorem 6 (Expected ACES-C Quality Score)**.** *Define $q_j := (\text{LOO-AUC}_j(w_{\text{unif}}) - \frac{1}{2})\,\hat{p}_j(1-\hat{p}_j)$, with $q_j = 0$ for constant columns. Under the model in Section 2.2:*

$$\mathbb{E}[q_j] \;=\; \pi(1-\pi)\left(A^{(-j)}(w_{\text{unif}}) - \frac{1}{2}\right) \cdot \delta_j.$$

Under Assumption 4, $\text{sign}(\mathbb{E}[q_j]) = \text{sign}(\delta_j)$; Proposition 5 bounds the variation in the proportionality factor across tests.

Proof sketch. Multiplication by $\hat{p}_j(1-\hat{p}_j)$ cancels the empirical AUC's passing–failing pair count, leaving an ordered-pair sum scaled by n^{-2}. Conditional independence and within-class exchangeability give zero expected contribution from same-class pairs. The two cross-class orientations together contribute $\delta_j(A^{(-j)}(w_{\text{unif}}) - 1/2)$; each has n^+n^- pairs, yielding the factor $\pi(1-\pi)$. Proposition 5 gives the sign under Assumption 4; full details are in Appendix A.4. □

ACES-C clips the empirical score: $w_j = \max(0,\, q_j)$ (Eq. 8). Under Assumption 4, its population counterpart $\bar{w}_j = \max(0,\, \mathbb{E}[q_j])$ is positive precisely when $\delta_j > 0$ and satisfies $R(\bar{w}) \ge \rho^2 R^*$ (Corollary 8), with $\rho \to 1$ as m grows for fixed positive $\bar{\delta}$.

3.2 ACES-O: Optimized Weighting

ACES-C is an efficient closed-form method with a population-level guarantee under Assumption 4. When the assumption may fail, ACES-O jointly optimizes the weights and their induced ranking:

$$\max_{w \in \Delta^m} \quad J(w) \;=\; \sum_{j=1}^{m} w_j \left(\text{LOO-AUC}_j(w) - \frac{1}{2}\right), \tag{9}$$

where $\text{LOO-AUC}_j(w)$ depends on w through the leave-one-out scores $S^{(-j)}(w)$ (Eq. 6). The simplex constraint leaves the induced ranking unchanged but prevents unbounded weights and ill-defined optimization. ACES-C evaluates LOO-AUC once under uniform weights, whereas ACES-O reevaluates it under the current weights w. Unlike ACES-C's hard clipping, ACES-O keeps weights positive rather than permanently excluding low-consistency tests. As the leave-one-out ranking changes, their LOO-AUC values can rise above $1/2$, allowing initially missed informative tests to regain weight.

Theoretical motivation. For fixed w, the population counterpart of J satisfies

$$J_{\text{pop}}(w) := \sum_j w_j(\mathcal{A}_j(w) - \tfrac{1}{2}) \;=\; \sum_j w_j\, c_j(w)\, \delta_j. \tag{10}$$

For $0 < p_j < 1$ and $c_j(w) > 0$, informative tests contribute positively to this objective and misleading tests negatively; constant population columns contribute zero. Assumption 4 ensures these signs at uniform weights, providing a theoretically justified starting point. The identity motivates

optimizing empirical consistency without knowing δ_j. Reevaluating tests under the updated ranking allows their weights and the ranking to adapt together, rather than fixing test quality after one uniform-weight evaluation. Since the AUC is non-differentiable, we optimize a logistic surrogate via gradient ascent; pseudocode, pre-filtering, and hyperparameters are in Appendix B. Appendix C.1 illustrates the weighting mechanisms at different average discriminative powers $\bar{\delta}$: ACES-C suffices in the higher-signal example, while ACES-O recovers initially missed informative tests in the lower-signal example.

4 EXPERIMENTS

We evaluate ACES-C and ACES-O on three code generation benchmarks, focusing on: (1) comparison with existing reranking methods, (2) empirical validation of Assumption 4, and (3) analysis of robustness and test quality detection under LOO-AUC-based weighting.

4.1 SETUP

Benchmarks and generation. We evaluate on HumanEval (Chen et al., 2021) (164 problems), HumanEval$^+$ (Liu et al., 2023) (164 problems, stricter tests), and MBPP (Austin et al., 2021) (427 problems). We use candidate solutions and tests from Huang et al. (2024), generated by GPT-3.5-Turbo, with approximately 200 candidates and 500 tests per problem. We report Pass@k for $k \in \{1, 2, 5\}$. For reranking methods, Pass@k follows Eq. 2; for direct inference baselines, Pass@k is the unbiased estimator of Chen et al. (2021), averaging over random k-subsets of candidates.

Baselines. We compare with post-hoc methods using code×test execution: Majority Voting (w_{unif}), CodeT (Chen et al., 2023) (consensus-set scoring on the pass matrix), and MBR-exec (Shi et al., 2022) (pairwise output comparison beyond the pass matrix); and methods requiring additional information: SC+Spec (Huang et al., 2024) (execution + specification consistency voting), Self-collaborate (Dong et al., 2024) (multi-agent LLM calls), MPSC (Huang et al., 2024) (multi-perspective consistency voting; Uniform variant, using only inter-consistency), and $\mathcal{DS}^3$ (Liu et al., 2026) (static analysis). We also include direct inference from GPT-3.5-Turbo, GPT-4 (Achiam et al., 2023), DeepSeek-Coder (Guo et al., 2024), WizardCoder (Luo et al., 2024), and CodeLlama (Roziere et al., 2023). All post-hoc methods share GPT-3.5-Turbo candidates and tests.

Implementation details of ACES-C and ACES-O are in Appendix B; results with additional generation models and benchmarks are in Appendix C.

4.2 MAIN RESULTS

In Table 2, ACES-O achieves the best Pass@k in nearly all execution-only comparisons using only the binary pass matrix. Combining ACES with static analysis yields the best reranking results across all three benchmarks.

Execution-only methods. Among execution-only methods, ACES-O achieves the best Pass@1 on all three benchmarks. On HumanEval, ACES-O reaches 84.15% Pass@1 versus 81.71% for $\mathcal{DS}^3$, which leverages additional static-analysis signals beyond the pass matrix. On HumanEval$^+$, which uses more comprehensive ground-truth tests for the same problems, ACES-O reaches 74.39% Pass@1, again outperforming $\mathcal{DS}^3$ (72.56%) despite using only the pass matrix. The gain over Majority Voting widens relative to HumanEval (+4.88% vs. +3.66% Pass@1), showing that principled test weighting remains effective under more comprehensive correctness evaluation. On MBPP, ACES-O leads all execution-only methods (72.37% Pass@1) but trails $\mathcal{DS}^3$ (75.88%), whose orthogonal static-analysis signals are complementary. For Pass@1, ACES surpasses all baselines on HumanEval and HumanEval$^+$ using only the pass matrix; on MBPP, only $\mathcal{DS}^3$ (static analysis) ranks higher.

Combination with static analysis. $\mathcal{DS}^3$ operates in two stages: heuristic pre-filtering of candidates, followed by static-analysis scoring (Pylint and cyclomatic complexity). We reuse its first-stage filtering and combine the second-stage static-analysis scores with ACES scores via a weighted sum (details in Appendix B). ACES-C + $\mathcal{DS}^3$ improves over $\mathcal{DS}^3$ alone on all three benchmarks ($+3.66$/$+4.88$/$+0.23$ Pass@1 on HumanEval/HumanEval$^+$/MBPP); ACES-O + $\mathcal{DS}^3$

Table 2: Pass@k (%) on HumanEval, HumanEval$^+$, and MBPP. Post-hoc methods rerank GPT-3.5-Turbo candidates (n≈200, m≈500). Subscripts denote the change from GPT-3.5-Turbo direct inference. **Bold**/underline: best/second-best within each group. Shaded : ours.

	HumanEval			HumanEval$^+$			MBPP		
Method	Pass@1	Pass@2	Pass@5	Pass@1	Pass@2	Pass@5	Pass@1	Pass@2	Pass@5
Direct inference									
GPT-3.5-Turbo	68.38	76.24	83.15	58.75	66.58	73.96	66.80	72.34	76.60
GPT-4	81.48	86.31	90.46	70.52	75.48	79.54	71.26	74.27	76.99
DeepSeek-Coder	79.30	–	–	–	–	–	70.00	–	–
WizardCoder	73.20	–	–	–	–	–	61.20	–	–
CodeLlama	62.20	–	–	–	–	–	62.20	–	–
Post-hoc reranking using only code × test execution									
MBR-exec	72.96$_{+4.6}$	76.47$_{+0.2}$	79.00$_{-4.2}$	62.12$_{+3.4}$	67.08$_{+0.5}$	71.38$_{-2.6}$	70.79$_{+4.0}$	73.14$_{+0.8}$	**74.85**$_{-1.8}$
CodeT	78.05$_{+9.7}$	78.05$_{+1.8}$	78.30$_{-4.9}$	67.87$_{+9.1}$	68.75$_{+2.2}$	69.65$_{-4.3}$	71.90$_{+5.1}$	71.95$_{-0.4}$	72.02$_{-4.6}$
Majority Voting	80.49$_{+12.1}$	82.93$_{+6.7}$	83.54$_{+0.4}$	69.51$_{+10.8}$	73.17$_{+6.6}$	76.83$_{+2.9}$	68.62$_{+1.8}$	70.49$_{-1.9}$	72.83$_{-3.8}$
ACES-C	82.93$_{+14.6}$	82.93$_{+6.7}$	82.93$_{-0.2}$	71.34$_{+12.6}$	71.95$_{+5.4}$	74.39$_{+0.4}$	71.19$_{+4.4}$	71.43$_{-0.9}$	72.37$_{-4.2}$
ACES-O	**84.15**$_{+15.8}$	**85.98**$_{+9.7}$	**86.59**$_{+3.4}$	**74.39**$_{+15.6}$	**75.61**$_{+9.0}$	**79.88**$_{+5.9}$	**72.37**$_{+5.6}$	**73.54**$_{+1.2}$	73.77$_{-2.8}$
Post-hoc reranking using additional information									
SC+Spec	73.86$_{+5.5}$	73.93$_{-2.3}$	74.10$_{-9.1}$	63.50$_{+4.8}$	64.70$_{-1.9}$	65.67$_{-8.3}$	71.70$_{+4.9}$	71.73$_{-0.6}$	71.82$_{-4.8}$
Self-collaborate	74.40$_{+6.0}$	–	–	–	–	–	68.20$_{+1.4}$	–	–
MPSC	74.17$_{+5.8}$	77.02$_{+0.8}$	78.53$_{-4.6}$	65.05$_{+6.3}$	69.76$_{+3.2}$	71.72$_{-2.2}$	69.34$_{+2.5}$	70.06$_{-2.3}$	71.85$_{-4.8}$
$\mathcal{DS}^3$	81.71$_{+13.3}$	82.32$_{+6.1}$	82.32$_{-0.8}$	72.56$_{+13.8}$	73.78$_{+7.2}$	75.00$_{+1.0}$	75.88$_{+9.1}$	**77.28**$_{+4.9}$	78.45$_{+1.9}$
ACES-C + $\mathcal{DS}^3$	**85.37**$_{+17.0}$	**85.98**$_{+9.7}$	**87.20**$_{+4.1}$	**77.44**$_{+18.7}$	**78.66**$_{+12.1}$	**81.10**$_{+7.1}$	76.11$_{+9.3}$	**77.28**$_{+4.9}$	**78.69**$_{+2.1}$
ACES-O + $\mathcal{DS}^3$	83.54$_{+15.2}$	83.54$_{+7.3}$	85.98$_{+2.8}$	75.00$_{+16.3}$	76.22$_{+9.6}$	79.88$_{+5.9}$	**76.58**$_{+9.8}$	**77.28**$_{+4.9}$	**78.69**$_{+2.1}$

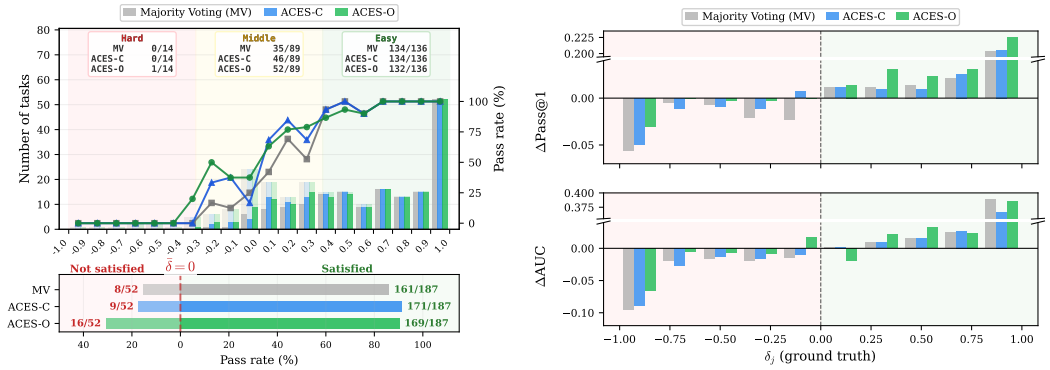
similarly improves, including a gain of $+0.70$ on MBPP, confirming that the two methods capture orthogonal signals. Interestingly, ACES-C + $\mathcal{DS}^3$ outperforms ACES-O + $\mathcal{DS}^3$ on HumanEval and HumanEval$^+$ (e.g., 85.37% vs. 83.54% Pass@1 on HumanEval), showing that closed-form weighting integrates effectively with this pre-filtered setting. In practice, ACES-O extracts more signal for standalone reranking, while ACES-C's closed-form weights integrate more effectively with high-quality pre-filtering and require no optimization.

Comparison of ACES variants. ACES-C and ACES-O are alternative weighting schemes rather than cumulative components. ACES-C improves Pass@1 over Majority Voting on all three benchmarks (e.g., 82.93% vs. 80.49% on HumanEval), showing that closed-form LOO-AUC weighting captures meaningful test-quality signal; ACES-O outperforms ACES-C across all reported benchmark–k settings. Further analyses, including finer Pass@k grids, selection vs. weighting, hyperparameter sensitivity, and computational cost, are in Appendix C.

4.3 Analysis

We restrict this analysis to non-trivial tasks ($0 < n^+ < n$; details in Table 7) and non-constant tests ($0 < \hat{p}_j < 1$), since neither trivial tasks nor constant test columns carry ranking signal.

Assumption satisfaction. Figure 2a examines performance across average test quality on MBPP, which has the most non-trivial tasks and the lowest Assumption 4 satisfaction rate among the three benchmarks. We bin tasks by $\bar{\delta}$ and use the simplified threshold $\bar{\delta} > 0$ for visual clarity (the formal threshold $2\sqrt{\ln 2/m}$ is small for $m \approx 500$; see Appendix C.4 for the formal analysis). The assumption is well satisfied on all benchmarks; detailed satisfaction rates are in Table 7. The performance gains concentrate in the Middle region, where informative and misleading tests coexist: ACES-O passes 52/89 tasks versus 46/89 for ACES-C and 35/89 for Majority Voting. In the Easy region, all methods achieve near-perfect pass rates ($\geq 97\%$), leaving little room for improvement; in the Hard region, no method passes more than 1 of 14 tasks, suggesting the pass matrix alone is insufficient when average test quality is very low and complementary signals such as static analysis may be needed (cf. Section 4.2). The bottom panel shows that both ACES variants outperform Majority Voting both when $\bar{\delta} > 0$ and when $\bar{\delta} \leq 0$; in the latter group, ACES-O substantially



(a) MBPP Pass@1 across average test quality $\bar{\delta}$. Bottom: pass rate when $\bar{\delta} > 0$ vs. $\bar{\delta} \leq 0$.

(b) Impact of each δ_j bin on Pass@1 (top) and AUC (bottom). Positive: helpful; negative: harmful.

Figure 2: **Empirical analysis on MBPP at Pass@1. (a)** Tasks binned by $\bar{\delta}$; bars show pass/fail counts, lines show pass rate; bottom panel compares pass rates for $\bar{\delta} > 0$ vs. $\bar{\delta} \leq 0$. **(b)** Performance impact of each δ_j bin upon removing its tests; full results in Appendix C.4.

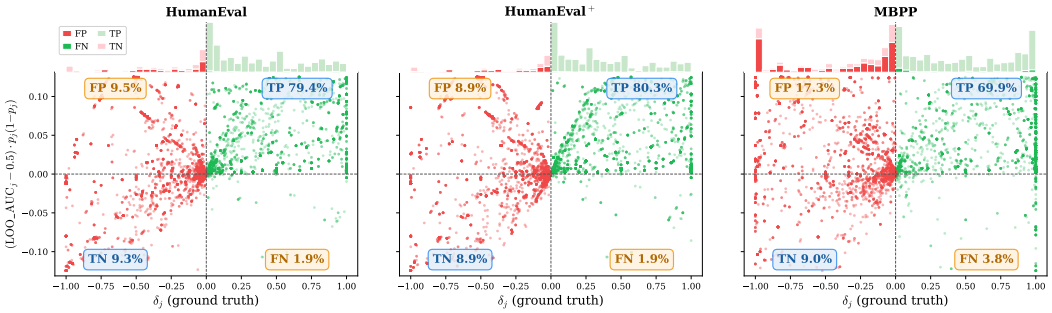


Figure 3: **Test quality detection.** ACES-C quality score $(\text{LOO-AUC}_j - \frac{1}{2})\,\hat{p}_j(1-\hat{p}_j)$ vs. ground-truth discriminative power δ_j for all non-trivial tests across three benchmarks. Green/red points denote informative/misleading tests ($\delta_j > 0/\delta_j < 0$); top marginals give per-bin TP/FP/FN/TN (saturated: errors; light: correct), and quadrants give TP/FP/FN/TN shares by quality-score sign (summing to 100%).

outperforms ACES-C (16/52 vs. 9/52), showing that iterative optimization is particularly valuable in this regime.

Impact of test quality on performance. To understand *why* ACES-O outperforms, we measure each δ_j bin's impact on performance as the drop incurred by removing its tests (Figure 2b). The key finding is an asymmetric dependence on test quality. In the misleading-test region ($\delta_j < 0$), both ACES variants are more robust: the most misleading bin ($\delta_j \approx -0.9$) reduces Pass@1 by 0.056 for Majority Voting but by only 0.049 for ACES-C and 0.030 for ACES-O (46% less sensitive). This is consistent with the ACES weights having already down-weighted misleading tests, so that their presence or absence has little effect on the final ranking. Conversely, informative tests ($\delta_j > 0$) contribute more to ACES-O than to Majority Voting (e.g., 0.030 vs. 0.012 at $\delta_j \approx 0.3$), because the optimization has up-weighted these high-quality tests. This asymmetry shows that LOO-AUC-based weighting effectively separates informative from misleading tests in the pass matrix.

Test quality detection. Figure 3 compares ACES-C quality scores with ground-truth δ_j across all benchmarks. Under the model and Assumption 4, Theorem 6 gives $\text{sign}(\mathbb{E}[q_j]) = \text{sign}(\delta_j)$; empirically, score signs identify at least 94.8% of informative tests. The top marginals show FP/FN errors cluster near $\delta_j \approx 0$, where weak discriminative power limits their ranking impact. MBPP exceeds HumanEval in FP fraction (17.3% vs. 9.5%) and misleading-test prevalence (FP+TN: 26.3% vs. 18.8%); because errors cluster near $\delta_j = 0$, this gap lies mainly among borderline tests.

5 Related Work

Code selection with generated tests. Post-hoc selection methods rerank candidates from code LLMs (Lozhkov et al., 2024; Hui et al., 2024) using execution-based signals. CodeT (Chen et al., 2023) groups candidates into consensus sets by their binary pass profiles and scores each set using solution multiplicity and passed-test counts, operating entirely on the pass matrix. Several methods exploit execution outputs beyond the pass matrix: MBR-exec (Shi et al., 2022) applies minimum Bayes risk decoding (Eikema & Aziz, 2022) via pairwise output comparison, SRank (To et al., 2024) clusters candidates by output equivalence, and ALGO (Zhang et al., 2023a) verifies against LLM-synthesized oracle programs. S^* (Li et al., 2025a) uses LLM-guided tournament-style selection with adaptively generated inputs. Others incorporate non-execution signals: Coder-Reviewer (Zhang et al., 2023b) uses generation log-probabilities, and LEVER (Ni et al., 2023) trains a verifier on code and execution features; RankEF (Sun et al., 2024) learns ranking from execution feedback. MPSC (Huang et al., 2024) adds specification-level consistency voting, and $\mathcal{DS}^3$ (Liu et al., 2026) augments execution with static analysis. CURE (Wang et al., 2025) jointly improves code and test generation via reinforcement learning, while CoCoEvo (Li et al., 2025b) co-evolves programs and tests through LLM-based search; self-debugging (Chen et al., 2024) iteratively refines candidates using execution feedback rather than selecting among them. On the test-quality side, ConVerTest (Taherkhani et al., 2026) applies self-consistency filtering (Wang et al., 2023a) before voting, and TestCase-Eval (Yang et al., 2025) evaluates LLM-generated test reliability; DOCE (Li et al., 2024) further examines how execution-based strategies compose. Across these approaches, no formal criterion has been established for identifying test quality from the pass matrix alone; ACES fills this gap with a provable criterion for distinguishing informative from misleading tests using only binary execution outcomes.

Ranking and noisy evaluation. Code selection via weighted voting is a bipartite ranking problem (Clémençon et al., 2008; Agarwal et al., 2005; Liu, 2010). AUC surrogate consistency is well studied (Gao & Zhou, 2015); ACES-O uses sigmoid smoothing for optimization. Recent extensions address partial (Wang et al., 2023b), weakly supervised (Xie et al., 2024), and multi-label (Lukasik et al., 2025) AUC optimization; our setting fits naturally into the last framework, where each test acts as a noisy annotator (Dawid & Skene, 1979). RankBoost (Freund et al., 2003) combines rankers using preference feedback; ACES infers test weights from internal consistency. The noisy-comparisons literature provides minimax rates (Shah & Wainwright, 2018) and noise tolerance results (Haddad, 2022). Nguyen et al. (2024) establish identifiability in multi-annotator noisy-label models with instance-dependent outliers; ACES estimates test quality without external correctness labels. The verifier paradigm (Cobbe et al., 2021; Lightman et al., 2024) and LLM-as-judge evaluation (Zheng et al., 2023) score candidates via trained or prompted models; ACES requires no additional training. Our δ_j mirrors the item discrimination index from classical test theory (Lord & Novick, 2008), adapted to noisy, machine-generated tests; the key difference is our self-referential LOO-AUC mechanism, which replaces the external criterion classical theory assumes.

6 Conclusion

We presented ACES (AUC Consistency Scoring), built on the insight that *test votes should rank, not merely count*. By evaluating tests against each other via leave-one-out AUC, we break the circular dependency between code and test quality assessment without any external supervision. Our population LOO-AUC identity and empirical-score expectation formula link test consistency to discriminative power, motivating test-quality estimation from the pass matrix alone. Building on this, ACES-C provides closed-form weights with a population-level oracle-approximation bound; ACES-O complements it by iteratively optimizing differentiable LOO-AUC without the average-quality assumption. Both use only the binary pass matrix with negligible overhead; ACES-O leads nearly all main-table execution-only comparisons, and combining ACES with static analysis yields the best reranking results across benchmarks.

The LOO-AUC framework opens several research directions. Incorporating correlations among LLM-generated tests could yield tighter bounds and stronger weighting schemes. More broadly, the principle of assessing evaluator quality via internal consistency extends naturally to other noisy-evaluator settings, including LLM-as-judge ensembles, crowdsourced annotation, and verification with process reward models.

AI USE STATEMENT

We used AI tools to aid and polish the writing of this manuscript, including grammar correction and improvements to clarity and fluency. The authors developed the core method, conducted the experiments, and take full responsibility for the scientific content and final manuscript.

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.

Shivani Agarwal, Thore Graepel, Ralf Herbrich, Sariel Har-Peled, and Dan Roth. Generalization bounds for the area under the roc curve. *Journal of Machine Learning Research*, 6(14):393–425, 2005.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *Arxiv Preprint Arxiv:2108.07732*, 2021.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *Arxiv Preprint Arxiv:2407.21787*, 2024.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *The Eleventh International Conference on Learning Representations*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *Arxiv Preprint Arxiv:2107.03374*, 2021.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024.

Stéphan Clémençon, Gábor Lugosi, and Nicolas Vayatis. Ranking and empirical minimization of u-statistics. *The Annals of Statistics*, pp. 844–874, 2008.

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, 2021. *Arxiv Preprint Arxiv:2110.14168*, 2021.

Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Journal of the Royal Statistical Society: Series C (applied Statistics)*, 28 (1):20–28, 1979.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via ChatGPT. *Acm Transactions on Software Engineering and Methodology*, 33(7), 2024.

Yali Du, Hui Sun, and Ming Li. Post-incorporating code structural knowledge into pretrained models via icl for code translation. *IEEE Transactions on Software Engineering*, 51(11):3038–3055, 2025.

Bryan Eikema and Wilker Aziz. Sampling-based approximations to minimum Bayes risk decoding for neural machine translation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 10978–10993. Association for Computational Linguistics, 2022.

Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4(Nov):933–969, 2003.

Wei Gao and Zhi-Hua Zhou. On the consistency of AUC pairwise optimization. In *International Joint Conference on Artificial Intelligence*, pp. 939–945, 2015.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. DeepSeek-Coder: When the large language model meets programming–the rise of code intelligence. *Arxiv Preprint Arxiv:2401.14196*, 2024.

Dany Haddad. Noise tolerance of learning to rank under class-conditional label noise. *Arxiv Preprint Arxiv:2208.02126*, 2022.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Baizhou Huang, Shuai Lu, Xiaojun Wan, and Nan Duan. Enhancing large language models in coding through multi-perspective self-consistency. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers)*, pp. 1429–1450. Association for Computational Linguistics, 2024.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-Coder technical report. *Arxiv Preprint Arxiv:2409.12186*, 2024.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *Findings of the Association for Computational Linguistics: Emnlp 2025*, pp. 15964–15978. Association for Computational Linguistics, 2025a.

Haau-Sing Li, Patrick Fernandes, Iryna Gurevych, and André FT Martins. DOCE: Finding the sweet spot for execution-based code generation. *Arxiv Preprint Arxiv:2408.13745*, 2024.

Kefan Li, Yuan Yuan, Hongyue Yu, Tingyu Guo, and Shijie Cao. CoCoEvo: Co-evolution of programs and test cases to enhance code generation. *Ieee Transactions on Evolutionary Computation*, pp. 1–1, 2025b.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

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*, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-Seventh Conference on Neural Information Processing Systems*, 2023.

Ren-Biao Liu, Anqi Li, Chaoding Yang, Hui Sun, and Ming Li. Revisiting Chain-of-Thought in code generation: Do language models need to learn reasoning before coding? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 38809–38826. PMLR, 13–19 Jul 2025.

Ren-Biao Liu, Jiang-Tian Xue, Chao-Zeng Ma, Hui Sun, Xin-Ye Li, and Ming Li. Dynamic-Static synergistic selection method for candidate code solutions with generated test cases. In *Proceedings of the Aaai Conference on Artificial Intelligence*, pp. 32096–32104, 2026.

Tie-Yan Liu. Learning to rank for information retrieval. In *Proceedings of the 33rd International Acm Sigir Conference on Research and Development in Information Retrieval*, pp. 904. Association for Computing Machinery, 2010.

Frederic M Lord and Melvin R Novick. *Statistical Theories of Mental Test Scores*. IAP, 2008.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *Arxiv Preprint Arxiv:2402.19173*, 2024.

Michal Lukasik, Lin Chen, Harikrishna Narasimhan, Aditya Krishna Menon, Wittawat Jitkrittum, Felix X. Yu, Sashank J. Reddi, Gang Fu, Mohammadhossein Bateni, and Sanjiv Kumar. Bipartite ranking from multiple labels: On loss versus label aggregation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 41074–41102. PMLR, 13–19 Jul 2025.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. WizardCoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*, 2024.

Tri Nguyen, Shahana Ibrahim, and Xiao Fu. Noisy label learning with instance-dependent outliers: Identifiability via crowd wisdom. In *Advances in Neural Information Processing Systems*, 2024.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. LEVER: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 26106–26128. PMLR, 2023.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *Arxiv Preprint Arxiv:2308.12950*, 2023.

Nihar B. Shah and Martin J. Wainwright. Simple, robust and optimal ranking from pairwise comparisons. *Journal of Machine Learning Research*, 18(199):1–38, 2018.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3533–3546. Association for Computational Linguistics, 2022.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Zhihong Sun, Yao Wan, Jia Li, Hongyu Zhang, Zhi Jin, Ge Li, and Chen Lyu. Sifting through the chaff: On utilizing execution feedback for ranking the generated code candidates. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 229–241. Association for Computing Machinery, 2024.

Hamed Taherkhani, Alireza DaghighFarsoodeh, Mohammad Chowdhury, Hung Viet Pham, and Hadi Hemmati. Consistency meets verification: Enhancing test generation quality in large language models without ground-truth solutions. *Arxiv Preprint Arxiv:2602.10522*, 2026.

Hung Quoc To, Minh Huynh Nguyen, and Nghi D. Q. Bui. Functional overlap reranking for neural code generation. In *Findings of the Association for Computational Linguistics: Acl 2024*, pp. 3686–3704. Association for Computational Linguistics, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023a.

Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. CURE: Co-evolving coders and unit testers via reinforcement learning. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems*, 2025.

Zitai Wang, Qianqian Xu, Zhiyong Yang, Yuan He, Xiaochun Cao, and Qingming Huang. Optimizing partial area under the top-k curve: Theory and practice. *Ieee Transactions on Pattern Analysis and Machine Intelligence*, 45(4):5053–5069, 2023b.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. LeetCodeDataset: A temporal dataset for robust evaluation and efficient training of code LLMs. *arXiv preprint arXiv:2504.14655*, 2025.

Zheng Xie, Yu Liu, Hao-Yuan He, Ming Li, and Zhi-Hua Zhou. Weakly supervised auc optimization: A unified partial auc approach. *Ieee Transactions on Pattern Analysis and Machine Intelligence*, 46(7):4780–4795, 2024.

Zheyuan Yang, Zexi Kuang, Xue Xia, and Yilun Zhao. Can LLMs generate high-quality test cases for algorithm problems? TestCase-eval: A systematic evaluation of fault coverage and exposure. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (volume 2: Short Papers)*, pp. 1050–1063. Association for Computational Linguistics, 2025.

Kexun Zhang, Danqing Wang, Jingtao Xia, William Yang Wang, and Lei Li. ALGO: Synthesizing algorithmic programs with generated oracle verifiers. In *Thirty-Seventh Conference on Neural Information Processing Systems*, 2023a.

Tianyi Zhang, Tao Yu, Tatsunori Hashimoto, Mike Lewis, Wen-tau Yih, Daniel Fried, and Sida Wang. Coder reviewer reranking for code generation. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 41832–41846. PMLR, 2023b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-Seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. DeepSeek-Coder-V2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*, 2024.

A Proofs and Theoretical Details

This section provides complete proofs for all theoretical results in the main text: the Pass@k bound (A.2), the population LOO-AUC identity (A.3), the expected quality-score identity (A.4), the structure of $c_j(w_{\text{unif}})$ (A.5), and the oracle approximation guarantee (A.6).

A.1 Proof Preliminaries

We summarize the notation and model properties used throughout the proofs.

Notation. Throughout the appendix, we use the following notation from the main text:

- $\alpha_j = P(B_{ij} = 1 \mid y_i = 1)$, $\beta_j = P(B_{ij} = 1 \mid y_i = 0)$: class-conditional pass rates of test t_j.
- $\delta_j = \alpha_j - \beta_j$: discriminative power of test t_j (Definition 1).
- $C^+ = \{c_i : y_i = 1\}$, $C^- = \{c_i : y_i = 0\}$: sets of correct and incorrect codes.
- $n^+ = |C^+|$, $n^- = |C^-|$, $\pi = n^+/n$: counts and fraction of correct codes.
- $p_j := \pi\alpha_j + (1-\pi)\beta_j$: marginal pass rate in the reference class mixture with correct-code probability π.
- $\hat{p}_j = n^{-1}\sum_i B_{ij}$: empirical pass rate; $\text{LOO-AUC}_j(w)$ is empirical LOO-AUC and $\mathcal{A}_j(w)$ its population counterpart.

Model properties. We condition on fixed labels $y_1, \ldots, y_n$ with $n^+, n^- > 0$ and use a fixed pool of m tests. As assumed in Section 2.2, rows are independent; within each row, test outcomes are independent Bernoulli variables with rates α_j or β_j according to its label. Thus rows within a class are identically distributed, while different tests may have different rates. All expectations are over these test outcomes, conditional on the labels; fixed-weight results require w to be independent of this evaluated matrix.

Computation and model scope. The expectation in Theorem 6 includes all realizations of the original fixed test pool, assigning $q_j = 0$ to empirical constant columns. Deleting such columns when computing uniform-weight LOO-AUC only removes a common score offset and, if renormalized, applies a positive scale factor, leaving the other empirical AUCs and quality scores unchanged. The guarantees concern the fixed pool before data-dependent candidate pre-filtering.

Hoeffding's inequality. Several proofs below rely on the following classical concentration inequality, which we state here for reference.

Lemma 7 (Hoeffding's inequality (Hoeffding, 1963))**.** *If $Z_1, \ldots, Z_m$ are independent random variables with $Z_j \in [a_j, b_j]$, then for any $t > 0$:*

$$P\left(\sum_{j=1}^{m} Z_j - \mathbb{E}\left[\sum_{j=1}^{m} Z_j\right] \le -t\right) \;\le\; \exp\left(-\frac{2\,t^2}{\sum_{j=1}^{m}(b_j - a_j)^2}\right).$$

A.2 Proof of Theorem 2 (Pass@k Bound)

Theorem (2, restated)**.** *Define the* mean signal *and* signal-to-noise ratio *of weights w:*

$$M(w) \;:=\; \sum_j w_j \delta_j, \qquad R(w) \;:=\; \frac{M(w)^2}{\sum_j w_j^2}.$$

For any fixed w independent of B, with $M(w) > 0$, and any $k \ge 1$:

$$\textit{Pass@}k(w) \;\ge\; 1 - \frac{n^-}{k}\,\exp\left(-\frac{R(w)}{2}\right),$$

where $n^- = n - n^+$ is the number of incorrect codes.

Proof. The proof proceeds in two steps: first we reduce Pass@k to a pairwise error probability, then we bound that probability using Hoeffding's inequality.

Step I: Reducing Pass@k to pairwise comparisons. Let $c^* = \arg\max_{c \in C^+} s_c(w)$ be the highest-scoring correct code, and define

$$R_{c^*} = |\{c^- \in C^- : s_{c^-}(w) \geq s_{c^*}(w)\}|,$$

i.e., the number of incorrect codes that score at least as high as the best correct code. If Pass@k fails (no correct code in the top k), then all k top-scoring codes are incorrect and each scores $\geq s_{c^*}(w)$, so $R_{c^*} \geq k$. We now bound $\mathbb{E}[R_{c^*}]$. For any fixed correct code c^+ (not necessarily c^*) and any incorrect code c^-, the event $\{s_{c^-} \geq s_{c^*}\}$ implies $\{s_{c^-} \geq s_{c^+}\}$ (since c^* has the highest score among correct codes). Therefore:

$$P(s_{c^-} \geq s_{c^*}) \leq P(s_{c^-} \geq s_{c^+}).$$

By exchangeability, the right-hand side is the same for every pair (c^+, c^-). Summing over all n^- incorrect codes:

$$\mathbb{E}[R_{c^*}] \leq n^- \cdot P(s_{c^-} \geq s_{c^+}).$$

Applying Markov's inequality:

$$P(\text{Pass@}k \text{ fails}) \leq P(R_{c^*} \geq k) \leq \frac{n^- \cdot P(s_{c^-} \geq s_{c^+})}{k}.$$

It remains to bound the pairwise error probability $P(s_{c^-} \geq s_{c^+})$.

Step II: Bounding pairwise error via Hoeffding's inequality. Fix a pair (c^+, c^-) and define the per-test vote $h_j = B_{c^+,j} - B_{c^-,j} \in \{-1, 0, +1\}$. Then the score difference is:

$$X = s_{c^+}(w) - s_{c^-}(w) = \sum_{j=1}^{m} w_j \, h_j.$$

By the assumed row and class-conditional test independence, the outcomes for this pair are independent across tests. This means $\{h_j\}_{j=1}^m$ are mutually independent random variables with:

$$\mathbb{E}[h_j] = \alpha_j - \beta_j = \delta_j, \qquad \mathbb{E}[X] = \sum_j w_j \delta_j = M(w) > 0.$$

Each $w_j h_j \in [-w_j, w_j]$. Applying Lemma 7 with $Z_j = w_j h_j$ (so $a_j = -w_j$, $b_j = w_j$, $b_j - a_j = 2w_j$) and $t = M(w)$:

$$\begin{aligned} P(s_{c^-} \geq s_{c^+}) = P(X \leq 0) &\leq \exp\left(-\frac{2\,M(w)^2}{\sum_j (2w_j)^2}\right) \\ &= \exp\left(-\frac{M(w)^2}{2\sum_j w_j^2}\right) = \exp\left(-\frac{R(w)}{2}\right). \end{aligned}$$

Combining. Substituting the Hoeffding bound into Step I:

$$P(\text{Pass@}k \text{ fails}) \leq \frac{n^-}{k} \exp\left(-\frac{R(w)}{2}\right),$$

and therefore:

$$\text{Pass@}k(w) = 1 - P(\text{Pass@}k \text{ fails}) \geq 1 - \frac{n^-}{k} \exp\left(-\frac{R(w)}{2}\right).$$

Step III: Oracle-optimal weights. We optimize over non-negative weights with $M(w) > 0$. Since $R(w) = M(w)^2 / \sum_j w_j^2$ is scale-invariant, we may normalize $\sum_j w_j^2 = 1$, so that maximizing $R(w)$ is equivalent to maximizing the positive sum $\sum_j w_j \delta_j$. For $w_j \geq 0$, any test with $\delta_j \leq 0$ contributes non-positively to $\sum_j w_j \delta_j$, so the optimum requires $w_j^* = 0$ for all such tests. Among the remaining tests ($\delta_j > 0$), by Cauchy–Schwarz:

$$\sum_{j:\,\delta_j > 0} w_j \delta_j \leq \sqrt{\sum_{j:\,\delta_j > 0} w_j^2} \cdot \sqrt{\sum_{j:\,\delta_j > 0} \delta_j^2},$$

with equality iff $w_j \propto \delta_j$. Under the normalization $\sum_j w_j^2 = 1$, this gives $w_j^* \propto \max(0,\ \delta_j)$ and $R(w^*) = \sum_{j:\delta_j>0} \delta_j^2$. When $\bar{\delta} > 0$, $\sum_{j:\delta_j>0} \delta_j \geq m\bar{\delta} > 0$; Cauchy–Schwarz with $g = |\{j : \delta_j > 0\}| \leq m$ gives:

$$R^* \;=\; \sum_{j:\,\delta_j>0} \delta_j^2 \;\geq\; \frac{(m\bar{\delta})^2}{m} \;=\; m\bar{\delta}^2 \;=\; R(w_{\mathrm{unif}}). \qquad \square$$

A.3 Proof of Theorem 3 (LOO-AUC Identity)

Theorem (3, restated)**.** *For fixed w and $0 < p_j < 1$,*

$$\mathcal{A}_j(w) - \frac{1}{2} = \frac{\pi(1-\pi)\delta_j}{p_j(1-p_j)} \left(A^{(-j)}(w) - \frac{1}{2} \right).$$

Proof. Define a population draw by first sampling $Y \sim \mathrm{Bernoulli}(\pi)$ and then sampling a test-outcome row from the class-conditional model. Let c_{p} and c_{f} be independent draws from this population conditional on passing and failing test j, respectively, and write $H(x) = \mathbf{1}[x > 0] + \frac{1}{2}\mathbf{1}[x = 0]$. By definition,

$$\mathcal{A}_j(w) = \mathbb{E}\Big[H\Big(S_{c_{\mathrm{p}}}^{(-j)} - S_{c_{\mathrm{f}}}^{(-j)} \Big) \Big].$$

Conditional independence ensures that, given the true class, conditioning on test j does not change the distribution of the remaining scores. Consequently the conditional comparison probabilities for class pairs $(1,0)$, $(0,1)$, and equal classes are $A^{(-j)}(w)$, $1 - A^{(-j)}(w)$, and $1/2$, respectively. The equal-class result follows from exchangeability and $H(x) + H(-x) = 1$.

Let r_{10} and r_{01} be the probabilities of the first two class pairs. Bayes' rule and independence of the two draws give

$$r_{10} = \frac{\pi\alpha_j}{p_j} \frac{(1-\pi)(1-\beta_j)}{1-p_j}, \qquad r_{01} = \frac{(1-\pi)\beta_j}{p_j} \frac{\pi(1-\alpha_j)}{1-p_j},$$
$$r_{10} - r_{01} = \frac{\pi(1-\pi)\delta_j}{p_j(1-p_j)}.$$

Conditioning on these class pairs therefore yields

$$\mathcal{A}_j(w) - \frac{1}{2} = (r_{10} - r_{01}) \left(A^{(-j)}(w) - \frac{1}{2} \right),$$

which is the claimed identity. If $p_j \in \{0, 1\}$, both classes have the same deterministic outcome and $\delta_j = 0$; the convention $\mathcal{A}_j = 1/2$, $c_j = 0$ gives zero contribution. $\square$

A.4 Proof of Theorem 6 (Expected ACES-C Quality Score)

Proof. Fix the labels and use $w = w_{\mathrm{unif}}$ independently of B. Let $N_j^+ = \sum_i B_{ij}$ and $N_j^- = n - N_j^+$, so $\hat{p}_j(1-\hat{p}_j) = N_j^+ N_j^- / n^2$. For a non-constant column, the empirical AUC is the average of $H(S_i^{(-j)} - S_k^{(-j)})$ over its passing–failing candidate pairs. Thus, with the same H as above,

$$q_j = \frac{1}{n^2} \sum_{i \neq k} B_{ij}(1 - B_{kj}) \left[H\Big(S_i^{(-j)} - S_k^{(-j)} \Big) - \frac{1}{2} \right]. \tag{11}$$

This equality also holds for a constant column: every summand is zero, agreeing with the definition $q_j = 0$.

Write $a_j = A^{(-j)}(w_{\mathrm{unif}}) - 1/2$. The held-out outcomes (B_{ij}, B_{kj}) are independent of the remaining scores conditional on their fixed labels. For a same-class ordered pair, the expected bracket in Eq. 11 is zero by exchangeability. For a pair with labels $(1,0)$ it is a_j, giving expected summand $\alpha_j(1-\beta_j)a_j$; for $(0,1)$ it is $-a_j$, giving $-\beta_j(1-\alpha_j)a_j$. There are exactly n^+n^- ordered pairs of

each cross-class type, hence

$$\mathbb{E}[q_j] = \frac{n^+n^-}{n^2}\big[\alpha_j(1-\beta_j) - \beta_j(1-\alpha_j)\big]a_j$$
$$= \pi(1-\pi)\,\delta_j\left(A^{(-j)}(w_{\text{unif}}) - \frac{1}{2}\right).$$

Under Assumption 4, Proposition 5 gives $a_j > 0$, establishing the sign statement. □

A.5 Proof of Proposition 5 (Structure of $c_j(w_{\text{unif}})$)

Proposition (5, restated)**.** *Under Assumption 4, $A^{(-j)}(w_{\text{unif}}) > 1/2$ for every j, so $c_j(w_{\text{unif}}) > 0$ whenever $0 < p_j < 1$. Moreover, $|A^{(-j)} - A^{(-k)}| \le \exp(-(m\bar{\delta}-1)^2/[2(m-1)])$.*

Proof. Under uniform weights $w_k = 1/m$, removing test j gives $w_k^{(-j)} = 1/m$ for $k \neq j$ and $w_j^{(-j)} = 0$. The mean signal and sum of squared weights are:

$$M^{(-j)} \;=\; \frac{1}{m}\sum_{k\neq j}\delta_k \;=\; \frac{m\bar{\delta}-\delta_j}{m}, \qquad \sum_k (w_k^{(-j)})^2 \;=\; \frac{m-1}{m^2}.$$

Therefore:

$$R^{(-j)}(w_{\text{unif}}) \;=\; \frac{M^{(-j)\,2}}{\sum_k (w_k^{(-j)})^2} \;=\; \frac{(m\bar{\delta}-\delta_j)^2}{m-1}.$$

Since $|\delta_j| \le 1$ (each $\delta_j = \alpha_j - \beta_j$ is a difference of probabilities) and Assumption 4 implies $m\bar{\delta} > 2\sqrt{m\ln 2} > 1$, the minimum of $R^{(-j)}(w_{\text{unif}})$ over $\delta_j \in [-1,1]$ is attained at $\delta_j = 1$:

$$R^{(-j)}(w_{\text{unif}}) \;\ge\; \frac{(m\bar{\delta}-1)^2}{m-1}.$$

We claim this lower bound exceeds $2\ln 2$. Let $a = m\bar{\delta}$. From Assumption 4, $m\bar{\delta}^2 > 4\ln 2$, so $m < a^2/(4\ln 2)$ and $2(m-1)\ln 2 = 2m\ln 2 - 2\ln 2 < a^2/2 - 2\ln 2$. Therefore:

$$(a-1)^2 - 2(m-1)\ln 2 \;>\; (a-1)^2 - \frac{a^2}{2} + 2\ln 2 \;=\; \frac{(a-2)^2}{2} + (2\ln 2 - 1) \;>\; 0,$$

where the last step uses $2\ln 2 > 1$. Hence $R^{(-j)}(w_{\text{unif}}) > 2\ln 2$ for every j.

To connect this to the AUC, recall from Section A.2 that for the score difference $X = s_{c^+}(w) - s_{c^-}(w)$, the Hoeffding bound gives $P(X \le 0) \le \exp(-R(w)/2)$. Since the true AUC satisfies

$$1 - A(w) \;=\; P(X<0) + \frac{1}{2}P(X=0) \;\le\; P(X \le 0),$$

we obtain $1 - A(w) \le \exp(-R(w)/2)$ for any w with $M(w) > 0$. Applying this to the leave-one-out ranking:

$$1 - A^{(-j)}(w_{\text{unif}}) \;\le\; \exp\!\left(-\frac{R^{(-j)}(w_{\text{unif}})}{2}\right) \;<\; \exp(-\ln 2) \;=\; \frac{1}{2}.$$

Therefore $A^{(-j)}(w_{\text{unif}}) > 1/2$ for every j, giving $c_j(w_{\text{unif}}) > 0$ for $0 < p_j < 1$.

Approximate constancy of $A^{(-j)}$. The bound above gives

$$1 - A^{(-j)}(w_{\text{unif}}) \;\le\; \exp\!\left(-\frac{(m\bar{\delta}-1)^2}{2(m-1)}\right)$$

for every j. Since all $A^{(-j)}(w_{\text{unif}}) \in (1/2,\,1]$, for any $a \ge b > 1/2$ we have $a - b \le 1 - b = \max(1-a,\,1-b)$, so

$$\left|A^{(-j)} - A^{(-k)}\right| \;\le\; \max\!\Big(1 - A^{(-j)},\; 1 - A^{(-k)}\Big) \;\le\; \exp\!\left(-\frac{(m\bar{\delta}-1)^2}{2(m-1)}\right).$$

Similarly, $1 - A(w_{\text{unif}}) \le \exp(-m\bar{\delta}^2/2)$. For fixed positive $\bar{\delta}$, both bounds vanish exponentially with m, so the leave-one-out ranking qualities approach a common value of one. □

A.6 Oracle Approximation Guarantee

Corollary 8. *Under Assumption 4, let $a_j = A^{(-j)}(w_{\mathrm{unif}}) - 1/2$ and $\rho = a_{\min}/a_{\max}$. Define the* population ACES-C weights $\bar{w}_j := \max(0, \mathbb{E}[q_j])$*, where q_j is the quality score from Theorem 6. Since $a_j > 0$, this gives $\bar{w}_j = \pi(1-\pi)\, a_j \max(0, \delta_j)$. Then:*

$$R(\bar{w}) \;\geq\; \rho^2\, R^*, \qquad \rho \;\geq\; 1 - 2\exp\!\left(-\frac{(m\bar{\delta}-1)^2}{2(m-1)}\right),$$

where $R^* = \sum_{j:\,\delta_j>0} \delta_j^2$ is the oracle signal-to-noise ratio (Theorem 2). For fixed positive $\bar{\delta}$, $\rho \to 1$ as m grows, so the population weights approach the oracle signal-to-noise ratio.

Proof. By definition and Theorem 6:

$$\bar{w}_j \;=\; \pi(1-\pi)\, a_j \max(0,\ \delta_j),$$

where $a_j > 0$ by Proposition 5. Only tests with $\delta_j > 0$ receive positive population weight, so:

$$M(\bar{w}) \;=\; \sum_{j:\,\delta_j>0} a_j\, \delta_j^2 \cdot \pi(1-\pi), \qquad \sum_j \bar{w}_j^2 \;=\; \sum_{j:\,\delta_j>0} a_j^2\, \delta_j^2 \cdot \pi^2(1-\pi)^2.$$

Since R is scale-invariant, $\pi(1-\pi)$ cancels:

$$R(\bar{w}) \;=\; \frac{\left(\sum_{j:\delta_j>0} a_j\, \delta_j^2\right)^2}{\sum_{j:\delta_j>0} a_j^2\, \delta_j^2} \;\geq\; \frac{a_{\min}^2 \left(\sum_{j:\delta_j>0} \delta_j^2\right)^2}{a_{\max}^2 \sum_{j:\delta_j>0} \delta_j^2} \;=\; \rho^2\, R^*.$$

For the bound on ρ: by Appendix A.5,

$$1 - A^{(-j)} \;\leq\; \exp\!\left(-\frac{(m\bar{\delta}-1)^2}{2(m-1)}\right) \quad \text{for all } j.$$

Since $a_{\max} \leq 1/2$ (because $A^{(-j)} \leq 1$) and $a_{\min} \geq 1/2 - \exp\!\big(-(m\bar{\delta}-1)^2/[2(m-1)]\big)$:

$$\rho \;=\; \frac{a_{\min}}{a_{\max}} \;\geq\; \frac{1/2 - \exp\!\big(-(m\bar{\delta}-1)^2/[2(m-1)]\big)}{1/2} \;=\; 1 - 2\exp\!\left(-\frac{(m\bar{\delta}-1)^2}{2(m-1)}\right). \qquad \square$$

B ALGORITHM DETAILS

This section provides pseudocode for both ACES variants, implementation details, and the combination procedure with $\mathcal{DS}^3$ referenced in Section 4. The pseudocode operates on the non-constant columns retained in computation.

B.1 ACES-C

Algorithm 1 implements ACES-C (Section 3.1). Under uniform initial weights, the leave-one-out scores reduce to unweighted row sums with one column removed; since the AUC is scale-invariant, normalization is unnecessary. The algorithm has no tunable parameters: it computes the LOO-AUC for each test, applies the pass-rate correction (Eq. 8), and returns the weighted scores in a single pass.

Algorithm 1 ACES-C (Closed-Form Weighting)

Require: Pass matrix $B \in \{0,1\}^{n \times m}$
Ensure: Code scores $s_1, \ldots, s_n$
for $j = 1$ **to** m **do**
  Compute $S_i^{(-j)} = \sum_{j' \neq j} B_{ij'}$ for all i {LOO scores; $\propto$ Eq. (6) under w_{unif}}
  Compute $\text{LOO-AUC}_j(w_{\text{unif}}) = \text{AUC}(S^{(-j)}, B_{:,j})$ {AUC is scale-invariant}
end for
$\hat{p}_j \leftarrow \frac{1}{n} \sum_i B_{ij}$ for all j {empirical pass rate}
Set $w_j \leftarrow \max(0,\ \text{LOO-AUC}_j(w_{\text{unif}}) - 1/2) \cdot \hat{p}_j(1 - \hat{p}_j)$ for all j {Eq. (8)}
return $s_i = \sum_j w_j B_{ij}$ for all i

B.2 ACES-O

Algorithm 2 implements ACES-O (Section 3.2), which maximizes the objective $J(w) = \sum_j w_j \left(\text{LOO-AUC}_j(w) - 1/2\right)$ via gradient ascent. Since the exact LOO-AUC involves indicator functions and is therefore non-differentiable, we replace it with a logistic surrogate. For each test t_j, let $\mathcal{P}_j = \{i : B_{ij} = 1\}$ and $\mathcal{N}_j = \{i : B_{ij} = 0\}$ denote the sets of codes that pass and fail t_j, respectively. The surrogate LOO-AUC is:

$$\widehat{\text{LOO-AUC}}_j(w) \;=\; \frac{1}{|\mathcal{P}_j|\,|\mathcal{N}_j|} \sum_{i \in \mathcal{P}_j} \sum_{k \in \mathcal{N}_j} \sigma\Big(\gamma\big(S_i^{(-j)}(w) - S_k^{(-j)}(w)\big)\Big), \tag{12}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the logistic function and $\gamma > 0$ is a sharpness parameter controlling the surrogate tightness. As $\gamma \to \infty$, $\sigma(\gamma x)$ approaches the step function and the surrogate recovers the exact LOO-AUC. Related AUC theory studies the consistency of pairwise logistic losses (Gao & Zhou, 2015); here, sigmoid smoothing enables gradient computation via automatic differentiation. The weights are parameterized as $w = \text{softmax}(\ell)$ with learnable logits $\ell \in \mathbb{R}^m$; this ensures $w \in \Delta^m$ throughout optimization.

Algorithm 2 ACES-O (Optimized Weighting)

Require: Pass matrix $B \in \{0,1\}^{n \times m}$, sharpness γ, learning rate η, steps T
Ensure: Code scores $s_1, \ldots, s_n$
Initialize logits $\ell \leftarrow \mathbf{0} \in \mathbb{R}^m$
for $t = 1$ **to** T **do**
  $w \leftarrow \mathrm{softmax}(\ell)$
  for $j = 1$ **to** m **do**
    $S_i^{(-j)}(w) \leftarrow \sum_{j' \neq j} w_{j'} B_{ij'}$ for all i {leave-one-out scores}
    Compute $\widehat{\mathrm{LOO\text{-}AUC}}_j(w)$ via logistic surrogate (Eq. 12)
  end for
  $\ell \leftarrow \ell + \eta \nabla_\ell \sum_j w_j \widehat{\mathrm{LOO\text{-}AUC}}_j(w)$ {gradient ascent (Adam in practice)}
end for
$w \leftarrow \mathrm{softmax}(\ell)$
return $s_i = \sum_j w_j B_{ij}$ for all i

We optimize the smooth, non-convex surrogate for a fixed number of steps. Under Assumption 4, Proposition 5 establishes the population sign property at uniform initialization, motivating this starting point. Empirically, we observe stable convergence across benchmarks (Appendix C.9).

B.3 Implementation Details

For ACES-O, we adopt a two-stage pipeline: first pre-filter to the top-K candidates by majority vote, then optimize weights on this shortlist. Pre-filtering removes obviously low-quality candidates and reduces the per-step cost from $O(nm^2)$ to $O(Km^2)$. ACES-C requires no pre-filtering and runs on all n candidates directly. ACES-O uses Adam with surrogate sharpness γ=10, learning rate η=0.01, and T=90 steps on HumanEval (η=0.05, T=90 on HumanEval$^+$ and MBPP), with top-K=24 pre-filtering (K=32 on MBPP).

B.4 Combination with $\mathcal{DS}^3$

As described in Section 4.2, we combine ACES with $\mathcal{DS}^3$ (Liu et al., 2026). We first summarize $\mathcal{DS}^3$'s scoring mechanism, then describe the combined formulation.

$\mathcal{DS}^3$ scoring. $\mathcal{DS}^3$ first applies a *pre-filtering* stage that sanitizes candidate code (AST-based extraction of the longest valid code block followed by dependency-driven dead-code elimination) and validates test inputs (AST-based argument extraction with signature-conformance checking). This removes malformed candidates and invalid tests, reducing the sets from n to $n' \leq n$ and from m to $m' \leq m$. On the filtered set, $\mathcal{DS}^3$ computes two signals. A *static quality vector* $\mathbf{q} \in [0,1]^{n'}$ scores each candidate by combining a normalized Pylint score (code quality) with a normalized cyclomatic-complexity score (lower complexity yields a higher score). A *pairwise consensus matrix* $P \in [0,1]^{n' \times n'}$ captures behavioral similarity:

$$P_{ik} \;=\; \frac{1}{m'} \sum_{j=1}^{m'} \mathbf{1}\big[\mathrm{exec}(c_i, t_j) = \mathrm{exec}(c_k, t_j)\big],$$

where $\mathrm{exec}(c_i, t_j)$ denotes the full output of candidate c_i on test t_j. The final $\mathcal{DS}^3$ score is $\mathbf{q}_* = P \cdot \mathbf{q}$: each candidate receives a consensus-weighted sum of static quality across all candidates, favoring solutions that are both well-structured and behaviorally consistent.

Combined formulation. ACES and $\mathcal{DS}^3$ capture complementary signals: ACES computes adaptive test weights from the pass matrix to produce per-candidate scores $s_i(w) = \sum_j w_j B_{ij}$, while $\mathcal{DS}^3$ combines static code quality with pairwise output consensus. We reuse $\mathcal{DS}^3$'s pre-filtering to obtain a cleaned candidate set of size n', then combine both signals on the filtered candidates via

$$\mathbf{r} \;=\; \big(\alpha\, I_{n'} + (1-\alpha)\, P\big) \cdot \big(\beta\, \mathbf{s}(w) + (1-\beta)\, \mathbf{q}\big), \tag{13}$$

where $I_{n'}$ is the identity matrix and $\alpha, \beta \in [0, 1]$. The coefficient β blends the per-candidate scoring signal: ACES scores $\mathbf{s}(w)$ at $\beta = 1$ and static quality $\mathbf{q}$ at $\beta = 0$. The coefficient α controls the aggregation: $\alpha = 1$ scores each candidate independently, while $\alpha = 0$ applies pairwise consensus weighting as in $\mathcal{DS}^3$. Setting $\alpha = \beta = 0$ recovers the original $\mathcal{DS}^3$ score $P \cdot \mathbf{q}$. Both $\mathbf{s}(w)$ and $\mathbf{q}$ lie in $[0, 1]^{n'}$ by construction, so no additional normalization is required. All $\mathcal{DS}^3$ components (pre-filtering, static scoring, pairwise consensus matrix, and all associated hyperparameters) use the official implementation and default settings of Liu et al. (2026).

C Additional Experimental Results

This section provides supplementary experiments: an illustrative walkthrough (C.1), additional models and benchmarks (C.2), extended Pass@k curves (C.3), assumption verification (C.4), selection vs. weighting analysis (C.5), sensitivity analyses (C.6–C.8), convergence (C.9), hyperparameter sensitivity (C.10), runtime (C.11), vote statistics (C.12), and generation prompts (C.13).

C.1 Illustrative Example

To illustrate the weighting mechanisms at different average discriminative powers, we construct two 8×10 instances, each with three correct codes, five incorrect codes, and ten tests ordered by δ_j. The *Easy* case has a higher mean signal ($\bar{\delta} = 0.16$), whereas the *Hard* case has a lower mean signal ($\bar{\delta} = 0.04$) and more strongly misleading tests. Figure 4 summarizes their pass matrices, inferred test weights, and resulting rankings; Tables 3 and 4 report the complete numerical data.

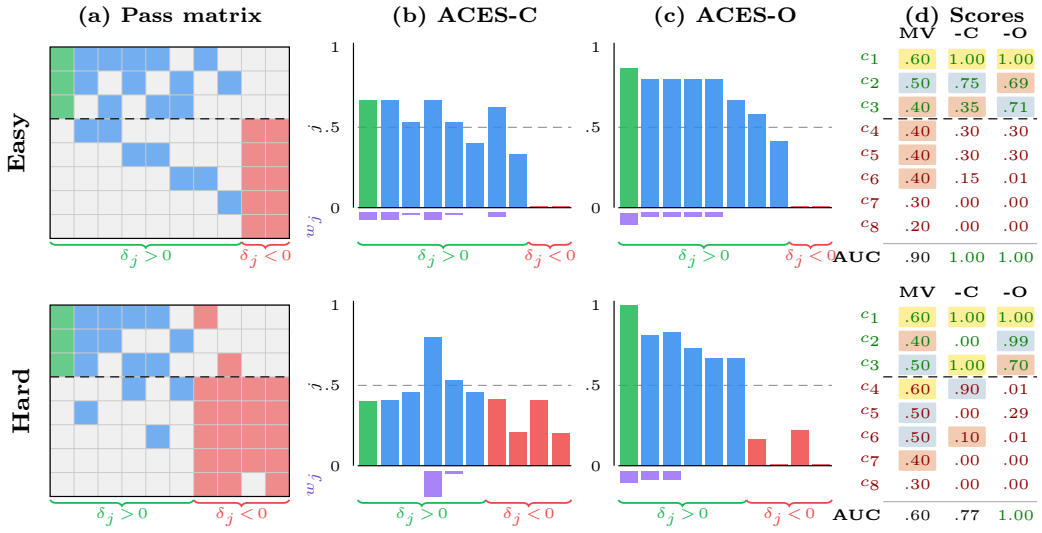


Figure 4: ACES on two constructed 8×10 instances (Easy: top; Hard: bottom). Test colors: green = perfect, blue = constructive/permissive, red = misleading. (a) Pass matrix. (b),(c) LOO-AUC$_j$ (upper) and weights w_j (lower, purple). (d) Scores; gold/silver/bronze mark the top-3 codes.

The two cases illustrate how closed-form reweighting and iterative refinement improve the ranking under different mixtures of informative and misleading tests. We now trace each case in the same sequence: MV $\rightarrow$ ACES-C $\rightarrow$ ACES-O.

Easy case (Table 3). Eight of ten tests are informative ($\delta_j > 0$) and only two are misleading, so the signal-to-noise ratio is favorable. *MV* assigns uniform weight to all tests and achieves AUC $= 0.90$; however, the lowest correct code c_3 ties with three incorrect ones at vote $= 4$, leaving these correct–incorrect pairs unresolved. *ACES-C* computes LOO-AUC under uniform weights and finds that 6 of the 8 informative tests have LOO-AUC$_j > 1/2$; it assigns non-zero weight to these 6 via Eq. (8), concentrating weight on the most discriminative tests (t_1, t_2, t_4 each receive $w = 0.25$). This breaks the MV tie and achieves AUC $= 1.00$: all three correct codes now rank above all five incorrect ones. *ACES-O* also reaches AUC $= 1.00$, further concentrating weight on the single strongest test (t_1, $w = 0.40$). When test quality is high, the closed-form ACES-C already suffices for perfect separation.

Hard case (Table 4). By contrast, six tests are informative and four are misleading, with all four misleading tests having strong discriminative power in the wrong direction ($|\delta_j| \geq 0.67$). *MV* achieves only AUC $= 0.60$: the incorrect code c_4 receives vote $= 6$, tying the best correct code c_1. *ACES-C* faces a challenge: the strong misleading tests corrupt the uniform-weight ranking, causing only 2 of 6 informative tests (t_4 and t_5) to have LOO-AUC$_j > 1/2$. Since ACES-C computes

Table 3: Easy case: 8×10 pass matrix with 8 informative and 2 misleading tests ($\bar{\delta} = 0.16$). Tests are sorted by δ_j descending. MV AUC: 0.90; ACES-C AUC: 1.00; ACES-O AUC: 1.00.

	correct ($\alpha_j = 1$)	**incorrect** ($\alpha_j < 1$)											
	perf.	**constructive** ($0 < \delta_j < 1$)							**misleading** ($\delta_j < 0$)				
	t_1	t_2	t_3	t_4	t_5	t_7	t_6	t_8	t_9	t_{10}	**MV**	**-C**	**-O**
✓ c_1	1	1	1	1	1	0	1	0	0	0	.60	1.00	1.00
✓ c_2	1	1	0	1	0	1	0	1	0	0	.50	.75	.69
✓ c_3	1	0	1	0	1	1	0	0	0	0	.40	.35	.71
× c_4	0	1	1	0	0	0	0	0	1	1	.40	.30	.30
× c_5	0	0	0	1	1	0	0	0	1	1	.40	.30	.30
× c_6	0	0	0	0	0	1	1	0	1	1	.40	.15	.01
× c_7	0	0	0	0	0	0	0	1	1	1	.30	.00	.00
× c_8	0	0	0	0	0	0	0	0	1	1	.20	.00	.00
α_j	1.00	.67	.67	.67	.67	.67	.33	.33	.00	.00			
β_j	.00	.20	.20	.20	.20	.20	.20	.20	1.00	1.00			
δ_j	+1.0	+.47	+.47	+.47	+.47	+.47	+.13	+.13	−1.0	−1.0			
LOO_C	.67	.67	.53	.67	.53	.40	.63	.33	.00	.00			
w_j^C	.25	.25	.05	.25	.05	.00	.15	.00	.00	.00			
LOO_O	.87	.80	.80	.80	.80	.67	.58	.42	.00	.00			
w_j^O	.40	.14	.15	.14	.15	.00	.00	.00	.00	.00			

Table 4: Hard case: 8×10 pass matrix with 6 informative and 4 misleading tests ($\bar{\delta} = 0.04$). Under uniform weights, only 2/6 informative tests have $\text{LOO-AUC}_j > 1/2$; ACES-O's co-evolution recovers all 6. MV AUC: 0.60; ACES-C AUC: 0.77; ACES-O AUC: 1.00.

	correct ($\alpha_j = 1$)		**incorrect** ($\alpha_j < 1$)										
	perf.	**perm.**	**constructive**				**misleading** ($\delta_j < 0$)						
	t_1	t_2	t_3	t_4	t_5	t_6	t_8	t_9	t_7	t_{10}	**MV**	**-C**	**-O**
✓ c_1	1	1	1	1	1	0	1	0	0	0	.60	1.00	1.00
✓ c_2	1	1	1	0	0	1	0	0	0	0	.40	.00	.99
✓ c_3	1	1	0	1	1	0	0	1	0	0	.50	1.00	.70
× c_4	0	0	0	1	0	1	1	1	1	1	.60	.90	.01
× c_5	0	1	0	0	0	0	1	1	1	1	.50	.00	.29
× c_6	0	0	0	0	1	0	1	1	1	1	.50	.10	.01
× c_7	0	0	0	0	0	0	1	1	1	1	.40	.00	.00
× c_8	0	0	0	0	0	0	1	1	0	1	.30	.00	.00
α_j	1.00	1.00	.67	.67	.67	.33	.33	.33	.00	.00			
β_j	.00	.20	.00	.20	.20	.20	1.00	1.00	.80	1.00			
δ_j	+1.0	+.80	+.67	+.47	+.47	+.13	−.67	−.67	−.80	−1.0			
LOO_C	.40	.41	.46	.80	.53	.46	.42	.21	.41	.20			
w_j^C	.00	.00	.00	.90	.10	.00	.00	.00	.00	.00			
LOO_O	1.00	.81	.83	.73	.67	.67	.17	.00	.22	.00			
w_j^O	.40	.29	.30	.01	.00	.00	.00	.00	.00	.00			

weights in closed form from these initial estimates, it can only assign weight to these two tests. This improves AUC to 0.77, but the incorrect c_4 remains ranked third. *ACES-O* overcomes this limitation through iterative co-evolution: improved weights produce a better ranking, which raises the LOO-AUC of initially misidentified tests, enabling further weight refinement. At convergence, all six informative tests have $\text{LOO-AUC}_j > 1/2$, and ACES-O concentrates weight on the three strongest (t_1, t_3, t_2), achieving AUC = 1.00. This demonstrates the value of iterative optimization when misleading tests are prevalent.

Taken together, these controlled cases make the complementarity between ACES-C and ACES-O explicit. We next move from mechanism-focused examples to broader evaluations across generation models and benchmarks.

C.2 Additional Generation Models and Benchmarks

We evaluate ACES on three additional code generation models beyond GPT-3.5-Turbo: Qwen2.5-Coder-7B and 14B (Hui et al., 2024), and DeepSeek-Coder-V2-16B (Zhu et al., 2024). In addition to HumanEval and HumanEval$^+$, we include LeetCodeDataset (Xia et al., 2025) as a harder benchmark. For each model, we generate candidates and test cases following the same protocol as Section 4.1. Table 5 reports results on all tasks; Table 6 reports results on non-trivial tasks ($0 < n^+ < n$), excluding tasks where all or no candidates are correct.

Table 5: Pass@k (%) with additional generation models on all tasks. Each model generates its own candidates and test cases following the protocol of Section 4.1. Subscripts denote the change from each model's zero-shot baseline. **Bold**/underline: best/second-best among reranking methods within each model. Shaded : ours.

	HumanEval			HumanEval$^+$			LeetCodeDataset		
Method	Pass@1	Pass@2	Pass@5	Pass@1	Pass@2	Pass@5	Pass@1	Pass@2	Pass@5
Qwen2.5-Coder-7B									
Zero-shot	81.69	87.21	91.27	75.63	81.38	85.62	14.64	18.82	24.50
Majority Voting	89.02$_{+7.3}$	90.24$_{+3.0}$	90.24$_{-1.0}$	80.49$_{+4.9}$	84.15$_{+2.8}$	84.15$_{-1.5}$	14.04$_{-0.6}$	15.79$_{-3.0}$	19.74$_{-4.8}$
ACES-C	89.63$_{+7.9}$	90.24$_{+3.0}$	90.24$_{-1.0}$	81.71$_{+6.1}$	84.15$_{+2.8}$	**84.76$_{-0.9}$**	**17.54$_{+2.9}$**	**18.86$_{+0.0}$**	20.18$_{-4.3}$
ACES-O	**90.24$_{+8.6}$**	**90.85$_{+3.6}$**	**90.85$_{-0.4}$**	**82.93$_{+7.3}$**	**84.76$_{+3.4}$**	**84.76$_{-0.9}$**	15.35$_{+0.7}$	16.67$_{-2.2}$	**20.61$_{-3.9}$**
Qwen2.5-Coder-14B									
Zero-shot	87.74	91.45	94.50	79.40	82.97	85.96	20.54	24.94	30.48
Majority Voting	92.07$_{+4.3}$	92.68$_{+1.2}$	92.68$_{-1.8}$	84.76$_{+5.4}$	**85.37$_{+2.4}$**	**87.80$_{+1.8}$**	23.25$_{+2.7}$	26.32$_{+1.4}$	31.14$_{+0.7}$
ACES-C	92.68$_{+4.9}$	92.68$_{+1.2}$	**93.29$_{-1.2}$**	84.76$_{+5.4}$	84.76$_{+1.8}$	86.59$_{+0.6}$	**28.07$_{+7.5}$**	**29.82$_{+4.9}$**	**32.89$_{+2.4}$**
ACES-O	**93.29$_{+5.6}$**	**93.29$_{+1.8}$**	**93.29$_{-1.2}$**	**85.37$_{+6.0}$**	**85.37$_{+2.4}$**	87.20$_{+1.2}$	21.49$_{+1.0}$	24.12$_{-0.8}$	28.95$_{-1.5}$
DeepSeek-Coder-V2-16B									
Zero-shot	77.74	82.26	86.50	71.05	75.30	79.03	20.68	24.25	28.14
Majority Voting	84.76$_{+7.0}$	85.98$_{+3.7}$	86.59$_{+0.1}$	76.83$_{+5.8}$	77.44$_{+2.1}$	78.05$_{-1.0}$	17.11$_{-3.6}$	20.18$_{-4.1}$	24.12$_{-4.0}$
ACES-C	**87.20$_{+9.5}$**	**87.20$_{+4.9}$**	**87.20$_{+0.7}$**	**78.66$_{+7.6}$**	**78.66$_{+3.4}$**	**78.66$_{-0.4}$**	**25.00$_{+4.3}$**	**25.88$_{+1.6}$**	**26.75$_{-1.4}$**
ACES-O	85.98$_{+8.2}$	86.59$_{+4.3}$	**87.20$_{+0.7}$**	78.05$_{+7.0}$	78.05$_{+2.8}$	**78.66$_{-0.4}$**	19.74$_{-0.9}$	21.05$_{-3.2}$	24.56$_{-3.6}$

Table 6: Pass@k (%) with additional generation models on non-trivial tasks ($0 < n^+ < n$), excluding tasks where all or no candidates are correct. Subscripts denote the change from each model's zero-shot baseline on the same set of tasks. **Bold**/underline: best/second-best among reranking methods within each model. Shaded : ours.

	HumanEval			HumanEval$^+$			LeetCodeDataset		
Method	Pass@1	Pass@2	Pass@5	Pass@1	Pass@2	Pass@5	Pass@1	Pass@2	Pass@5
Qwen2.5-Coder-7B									
Zero-shot	76.98	86.03	92.68	75.03	84.46	91.42	31.02	42.24	57.49
Majority Voting	89.00$_{+12.0}$	91.00$_{+5.0}$	91.00$_{-1.7}$	83.00$_{+8.0}$	89.00$_{+4.5}$	89.00$_{-2.4}$	29.41$_{-1.6}$	34.12$_{-8.1}$	44.71$_{-12.8}$
ACES-C	90.00$_{+13.0}$	91.00$_{+5.0}$	91.00$_{-1.7}$	85.00$_{+10.0}$	89.00$_{+4.5}$	**90.00$_{-1.4}$**	**38.82$_{+7.8}$**	**42.35$_{+0.1}$**	45.88$_{-11.6}$
ACES-O	**91.00$_{+14.0}$**	**92.00$_{+6.0}$**	**92.00$_{-0.7}$**	**87.00$_{+12.0}$**	**90.00$_{+5.5}$**	**90.00$_{-1.4}$**	32.94$_{+1.9}$	36.47$_{-5.8}$	**47.06$_{-10.4}$**
Qwen2.5-Coder-14B									
Zero-shot	68.41	80.36	90.16	67.05	77.32	85.91	32.83	42.87	55.50
Majority Voting	82.35$_{+13.9}$	84.31$_{+4.0}$	84.31$_{-5.9}$	82.46$_{+15.4}$	**84.21$_{+6.9}$**	**91.23$_{+5.3}$**	39.00$_{+6.2}$	46.00$_{+3.1}$	57.00$_{+1.5}$
ACES-C	84.31$_{+15.9}$	84.31$_{+4.0}$	**86.27$_{-3.9}$**	82.46$_{+15.4}$	82.46$_{+5.1}$	87.72$_{+1.8}$	**50.00$_{+17.2}$**	**54.00$_{+11.1}$**	**61.00$_{+5.5}$**
ACES-O	**86.27$_{+17.9}$**	**86.27$_{+5.9}$**	**86.27$_{-3.9}$**	**84.21$_{+17.2}$**	**84.21$_{+6.9}$**	89.47$_{+3.6}$	35.00$_{+2.2}$	41.00$_{-1.9}$	52.00$_{-3.5}$
DeepSeek-Coder-V2-16B									
Zero-shot	64.38	75.63	86.15	61.77	72.63	82.19	39.14	49.74	61.25
Majority Voting	81.82$_{+17.4}$	84.85$_{+9.2}$	86.36$_{+0.2}$	76.56$_{+14.8}$	78.12$_{+5.5}$	79.69$_{-2.5}$	28.57$_{-10.6}$	37.66$_{-12.1}$	49.35$_{-11.9}$
ACES-C	**87.88$_{+23.5}$**	**87.88$_{+12.3}$**	**87.88$_{+1.7}$**	**81.25$_{+19.5}$**	**81.25$_{+8.6}$**	**81.25$_{-0.9}$**	**51.95$_{+12.8}$**	**54.55$_{+4.8}$**	**57.14$_{-4.1}$**
ACES-O	84.85$_{+20.5}$	86.36$_{+10.7}$	**87.88$_{+1.7}$**	79.69$_{+17.9}$	79.69$_{+7.1}$	**81.25$_{-0.9}$**	36.36$_{-2.8}$	40.26$_{-9.5}$	50.65$_{-10.6}$

Three findings emerge from these results.

At least one ACES variant improves Pass@1 over Majority Voting on every model–benchmark pair. The improvement is most pronounced at k=1, where ranking quality is most critical. On HumanEval, the best ACES variant gains $+1.2$ to $+2.4$ points over MV across the three models; on HumanEval$^+$, the gain ranges from $+0.6$ to $+2.4$ points. The two variants exhibit complementary strengths: ACES-O leads ACES-C on HumanEval and HumanEval$^+$ for both Qwen models, whereas ACES-C leads for DeepSeek-Coder-V2-16B and on LeetCodeDataset across all three models.

On LeetCodeDataset, ACES-C is consistently strongest and corrects MV failures on the harder pools. Majority Voting falls below zero-shot for Qwen2.5-Coder-7B (14.04% vs. 14.64%) and DeepSeek-Coder-V2-16B (17.11% vs. 20.68%), while improving over zero-shot for Qwen2.5-Coder-14B (23.25% vs. 20.54%). ACES-C reaches 17.54%, 28.07%, and 25.00% Pass@1, respectively, exceeding both MV and zero-shot for every model; its gain over MV grows from $+3.5$ points on Qwen2.5-Coder-7B to $+4.8$ on Qwen2.5-Coder-14B and $+7.9$ on DeepSeek-Coder-V2-16B. ACES-O is less stable: it improves over MV for Qwen2.5-Coder-7B and DeepSeek-Coder-V2-16B, but trails MV for Qwen2.5-Coder-14B (21.49% vs. 23.25%). The consistent ACES-C lead, together with ACES-O's regression on Qwen2.5-Coder-14B, shows that the closed-form variant is more robust on these LeetCodeDataset candidate pools, whereas iterative optimization does not uniformly improve over MV in this regime.

Non-trivial tasks amplify the improvements. When trivial tasks are excluded (Table 6), all methods operate on tasks where the ranking actually matters, and the ACES advantage becomes more visible. On LeetCodeDataset non-trivial tasks, ACES-C improves over MV by $+9.4$ points (Qwen2.5-Coder-7B), $+11.0$ points (Qwen2.5-Coder-14B), and $+23.4$ points (DeepSeek-Coder-V2-16B) at Pass@1. On HumanEval non-trivial tasks, the corresponding gains are $+1.0$, $+2.0$, and $+6.1$ points, respectively.

C.3 Pass@k vs. k

Figure 5 extends the main results (combined with $\mathcal{DS}^3$ pre-filtering) to $k = 1, \ldots, 20$. Both ACES variants substantially outperform Majority Voting across all benchmarks and k, with the largest gains at small k where ranking quality matters most. On HumanEval and HumanEval$^+$, ACES-C + $\mathcal{DS}^3$ leads at small k (e.g., $+2.44\%$ Pass@1 on HumanEval$^+$), reflecting its closed-form weighting that concentrates on a few highly discriminative tests for sharp top-1 discrimination; the gap between the two variants narrows as k grows and they converge at larger k. On MBPP, ACES-O + $\mathcal{DS}^3$ slightly leads at small k, consistent with MBPP's higher fraction of misleading tests (Section 4.3) making iterative optimization more valuable; beyond k=2 the two variants are nearly identical. The overall improvement scales with benchmark difficulty: $\sim$6–8% on MBPP (most misleading tests) vs. $\sim$3–5% on HumanEval, confirming the greater benefit of principled test weighting when test quality is more heterogeneous.

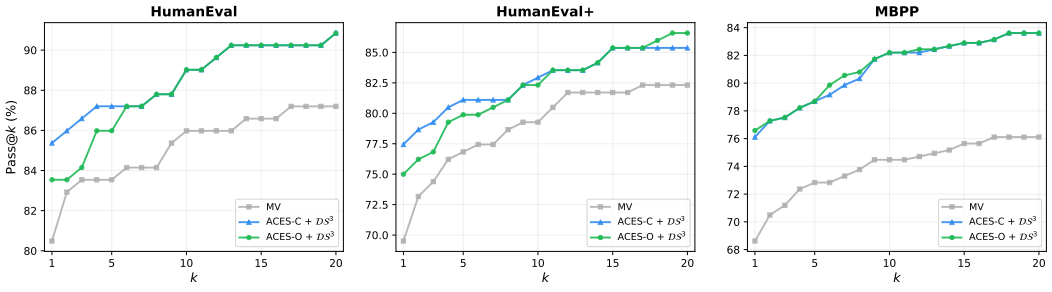


Figure 5: Pass@k vs. k ($k = 1, \ldots, 20$) on all three benchmarks, combined with $\mathcal{DS}^3$ pre-filtering. ACES-C + $\mathcal{DS}^3$ leads at small k on HumanEval/HumanEval$^+$; the two variants converge at larger k. On MBPP they perform comparably throughout.

C.4 Assumption Satisfaction Analysis

We verify Assumption 4 empirically on all three benchmarks. For each task, we remove trivial tasks (all codes correct or all incorrect) and constant-column tests (all codes pass or all fail), then compute $\bar{\delta} = (1/m)\sum_j \delta_j$ over the remaining m non-constant tests. Assumption 4 formally requires $\bar{\delta} > 2\sqrt{\ln 2/m}$, where the threshold is task-specific since each task has a different number of non-constant tests m. Table 7 reports the formal satisfaction rate. The figures in this section bin tasks by $\bar{\delta}$ using the simplified threshold $\bar{\delta} > 0$ for visual clarity. Tasks are binned with width 0.1 (left-open right-closed) and further divided into three equal-width regions (*Hard* / *Middle* / *Easy*).

Table 7: Assumption 4 satisfaction across benchmarks. The threshold $2\sqrt{\ln 2/m}$ is evaluated per task using its number of non-constant tests m.

Benchmark	Non-trivial tasks	$\bar{\delta} > 2\sqrt{\ln 2/m}$ (%)	$\bar{\delta}$ range
HumanEval	118	83.1	$[-0.42,\ 1.00]$
HumanEval$^+$	123	82.1	$[-0.42,\ 1.00]$
MBPP	239	71.1	$[-0.90,\ 1.00]$

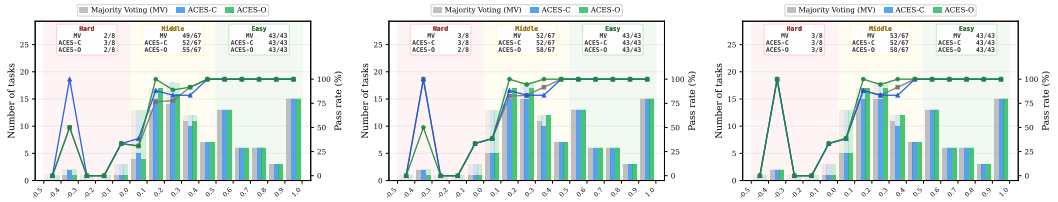
The assumption is satisfied for the majority of non-trivial tasks on all benchmarks. MBPP has a higher proportion of hard tasks, consistent with its greater fraction of misleading tests.

Figures 6–8 show the full per-benchmark results for Pass@k ($k \in \{1, 2, 5\}$), extending the MBPP Pass@1 analysis in Section 4.3 to all benchmarks and k values. The key findings from the main text generalize consistently across all three benchmarks and k values:

(i) The Middle region is where ACES provides the most value. In the Easy region (high $\bar{\delta}$), all methods achieve near-perfect pass rates since most tests are informative and even uniform weighting works well. The performance gains of ACES concentrate in the Middle region, where informative and misleading tests coexist and principled test weighting becomes critical.

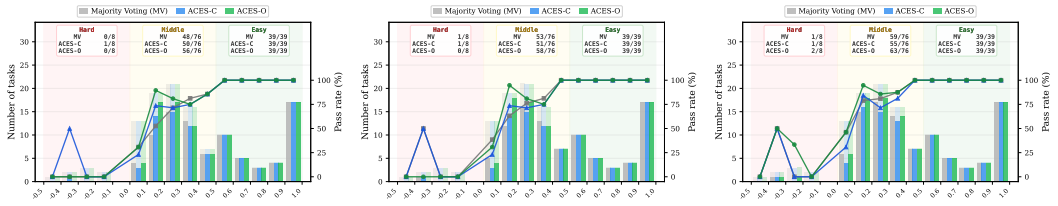
(ii) ACES-O and ACES-C have complementary strengths across difficulty regimes. At low $\bar{\delta}$, ACES-O's iterative optimization is more effective at identifying and up-weighting the few reliable tests, giving it a clear advantage over ACES-C. At high $\bar{\delta}$, ACES-C matches or approaches ACES-O without requiring iterative optimization.

(iii) As k increases, the gap between methods narrows, since selecting more candidates reduces the sensitivity to ranking quality at the top. This pattern is consistent across all benchmarks.



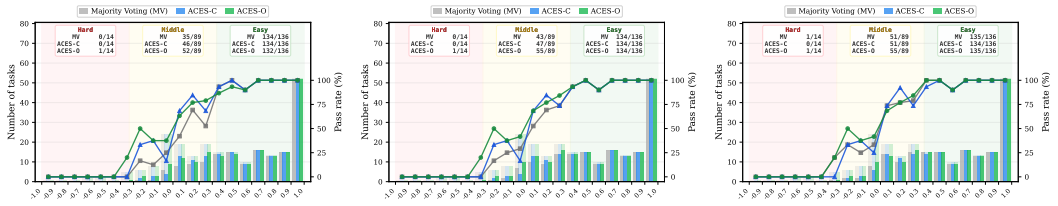
(a) Pass@1 (b) Pass@2 (c) Pass@5

Figure 6: Assumption 4 analysis on HumanEval (tasks binned by $\bar{\delta}$). ACES-O leads in the Middle region across all k; the gap narrows as k increases.



(a) Pass@1 (b) Pass@2 (c) Pass@5

Figure 7: Assumption 4 analysis on HumanEval$^+$ (tasks binned by $\bar{\delta}$). ACES-O leads in the Middle region across all k; the gap narrows as k increases.



(a) Pass@1 (b) Pass@2 (c) Pass@5

Figure 8: Assumption 4 analysis on MBPP (tasks binned by $\bar{\delta}$). ACES-O leads in the Middle region across all k; the gap narrows as k increases.

C.5 Selection vs. Weighting

ACES-C combines *test selection* through the $\max(0, \cdot)$ threshold with *non-uniform weighting* of the retained tests. To disentangle these effects, we analyze ideal filtering by true test quality and then compare empirical filtering with weighting.

Theoretical motivation. Under the model and Assumption 4, Theorem 6 gives

$$\operatorname{sign}(\mathbb{E}[q_j]) = \operatorname{sign}(\delta_j).$$

If $\bar{\delta} > 0$ and one retains exactly the m' tests with $\delta_j > 0$, uniform weights on them cannot decrease the signal-to-noise ratio. Let $R(w_{\text{unif}}) = m\bar{\delta}^2$ denote the SNR of Majority Voting (Section 2.2). Uniform weighting over only the m' tests with $\delta_j > 0$ gives:

$$\frac{R(w_{\text{filter}})}{R(w_{\text{unif}})} = \underbrace{\left(\frac{\sum_{j:\,\delta_j>0}\delta_j}{\sum_{j=1}^{m}\delta_j}\right)^2}_{\geq 1} \cdot \underbrace{\frac{m}{m'}}_{\geq 1} \geq 1,$$

Retaining only positive terms preserves or increases the total signal, and $m/m' \geq 1$. By Theorem 2, a higher $R(w)$ yields a tighter Pass@k bound.

The expected-score identity motivates the empirical threshold LOO-AUC$_j > 1/2$. For $m' > 0$, define ACES-C Filter as:

$$w_j^{\text{filter}} = \begin{cases} 1/m' & \text{if LOO-AUC}_j(w_{\text{unif}}) > 1/2, \\ 0 & \text{otherwise,} \end{cases} \qquad m' = |\{j : \text{LOO-AUC}_j(w_{\text{unif}}) > 1/2\}|.$$

Experimental design. We compare three methods: (1) Majority Voting (uniform weights on all tests), (2) ACES-C Filter (discard tests with LOO-AUC$_j \leq 1/2$, uniform weights on the rest), and (3) ACES-C (LOO-AUC excess weighting, Eq. (8)).

Table 8: Selection vs. weighting (Pass@1). Subscripts denote the change from GPT-3.5-Turbo direct inference. ACES-C Filter isolates the contribution of test selection; ACES-C adds non-uniform weighting on top.

Method	HumanEval	HumanEval$^+$	MBPP
GPT-3.5-Turbo	68.38	58.75	66.80
Majority Voting	$80.49_{+12.1}$	$69.51_{+10.8}$	$68.62_{+1.8}$
ACES-C Filter	$81.10_{+12.7}$	$69.51_{+10.8}$	$69.32_{+2.5}$
ACES-C	$82.93_{+14.6}$	$71.34_{+12.6}$	$71.19_{+4.4}$

Results. Empirically, ACES-C Filter matches or improves over Majority Voting on all three benchmarks (+0.61 on HumanEval, +0.00 on HumanEval$^+$, +0.70 on MBPP). However, the gains are modest because most misleading tests have $|\delta_j|$ close to zero (Figure 3), so removing them changes $R(w)$ only slightly. The additional gain from non-uniform weighting (ACES-C vs. ACES-C Filter) is substantially larger (+1.83 on HumanEval, +1.83 on HumanEval$^+$, +1.87 on MBPP), consistently accounting for over 70% of the total improvement. This demonstrates that the primary value of LOO-AUC lies not in binary test filtering, but in the continuous weight magnitudes that amplify informative tests proportionally to their discriminative power.

C.6 Sensitivity to Number of Tests

Figure 9 shows Pass@1 as a function of the number of available tests m', where tests are randomly subsampled from the full test suite (averaged over 5 trials). All methods use the same configuration as in the main experiments (Section 4.1).

All three methods improve consistently with m', but their scaling behaviors differ substantially.

MV plateaus around $m' = 50$–100. Since uniform weighting treats informative and misleading tests equally, adding more tests of mixed quality provides diminishing marginal benefit once the pool is large enough. Both ACES variants continue improving beyond this plateau: more tests provide better LOO-AUC estimates, enabling finer-grained weight differentiation between informative and misleading tests. At $m' = 100$, ACES-C already outperforms MV at full $m' = 500$ on all benchmarks, showing that intelligent weighting is more valuable than simply collecting more tests.

The gap between ACES-O and ACES-C generally widens with m': at small m' the two variants perform similarly, but with more tests ACES-O's optimization can exploit the richer signal to further refine the weights. Practically, even a modest test budget ($m' \approx 100$) suffices for the LOO-AUC weighting to capture most of the available signal.

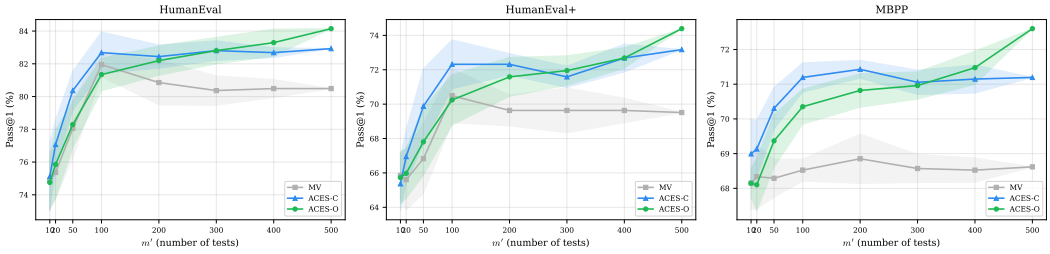


Figure 9: Sensitivity to the number of tests m'. Tests are randomly subsampled; results averaged over 5 trials (shaded: $\pm$ std). MV plateaus around m'=50–100; both ACES variants continue improving and surpass MV at m'=500 with only m'=100 tests.

C.7 Sensitivity to Number of Candidate Codes

Figure 10 shows Pass@1 as a function of the number of candidate codes n', where codes are stratified-subsampled to preserve the correct/incorrect ratio π (averaged over 10 trials). All m tests are retained; all methods use the same configuration as in the main experiments.

All three methods are stable with respect to n' (standard deviations $\leq 1.6\%$ throughout), but their scaling behaviors differ.

MV is approximately flat across all n', and on MBPP it *decreases* from 70.8% ($n' = 20$) to 68.6% ($n' = 200$): with more candidates (most of which are incorrect), there are more incorrect codes that may outscore the best correct one under equal weighting. ACES-C maintains performance close to its full-data result even at $n' = 20$. This is because ACES-C computes weights in closed form from the pass matrix; even a small candidate pool suffices to produce meaningful LOO-AUC estimates. ACES-O benefits the most from additional candidates. At small n', the optimization has too few pairwise comparisons between passing and failing codes to learn reliable weights; as n' grows, the gradient signal becomes richer and ACES-O achieves the best performance at $n' = 200$ on all benchmarks.

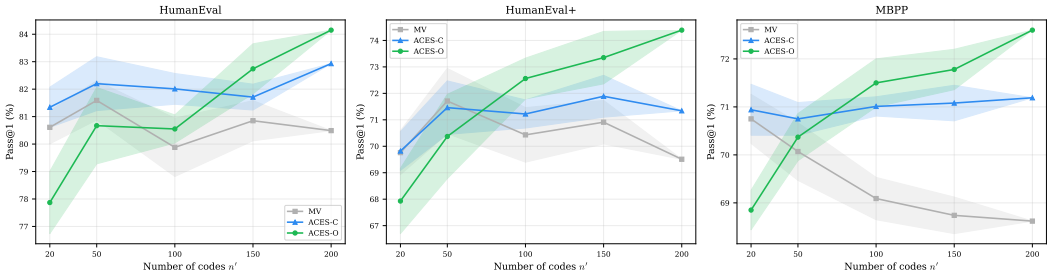


Figure 10: Sensitivity to the number of candidate codes n'. Codes are stratified-subsampled to preserve the correct/incorrect ratio; results averaged over 10 trials (shaded: $\pm$ std). MV is flat or declining; ACES-C is robust even at small n'; ACES-O improves steadily and leads at n'=200.

Summary of data sensitivity (Appendices C.6–C.7). The two sensitivity analyses reveal complementary strengths of the three methods. MV plateaus early with m' and can even degrade with larger n', since uniform weighting cannot exploit additional tests or candidates. ACES-C is robust to both m' and n', requiring neither many tests nor many candidates to achieve strong performance; it is the safest choice under limited data budgets. ACES-O achieves the best performance when both m' and n' are sufficiently large, as the optimization benefits from richer pairwise signal; in the standard setting (n=200, $m \approx 500$), it consistently leads on all benchmarks.

C.8 Effect of Pre-Filtering Cutoff

Figure 11 shows the effect of the pre-filtering cutoff K on both ACES variants. For each K, we select the top-K candidates by majority vote, run the reranking method on the resulting $K \times m$ submatrix, and score all $n = 200$ codes with the learned weights.

The two variants exhibit complementary trends, each excelling in a different regime. ACES-C generally improves as K increases, reaching its best performance at $K = 200$ (no pre-filtering) on all benchmarks. This is consistent with the findings in Appendix C.7: more codes provide better LOO-AUC estimates for the closed-form weighting.

ACES-O achieves its best results at small to moderate K (8–32), where the pre-filtered candidate pool concentrates on high-quality codes and provides a cleaner optimization landscape. Performance is strong across a wide range of K values and declines gradually for larger K, since including many low-quality candidates introduces noise into the pairwise comparisons.

These complementary profiles are consistent with the design of the two variants: ACES-C operates on all n candidates to maximize its closed-form estimates, while ACES-O benefits from a focused candidate pool for optimization.

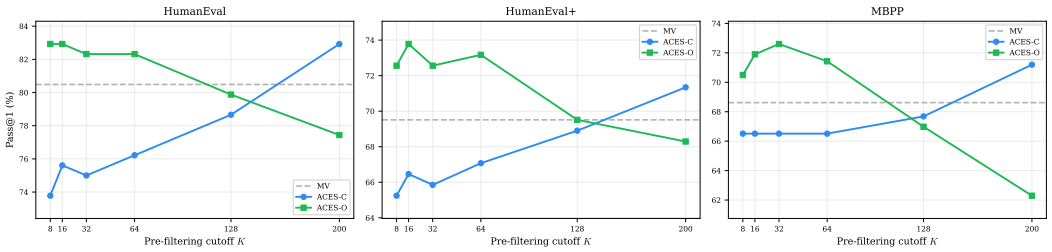


Figure 11: Sensitivity to the pre-filtering cutoff K. ACES-C (blue) benefits from larger candidate pools; ACES-O (green) benefits from focused pre-filtering. Dashed line: MV baseline (independent of K).

C.9 ACES-O Convergence

Figure 12 shows the convergence behavior of ACES-O over $T = 300$ iterations, averaged across all tasks per benchmark. The top row plots the surrogate objective $J(w) = \sum_j w_j \left(\widehat{\text{LOO-AUC}}_j(w) - \frac{1}{2}\right)$ (Eq. 9), where $\widehat{\text{LOO-AUC}}_j$ is the logistic surrogate (Eq. 12). The bottom row plots the corresponding Pass@1 (smoothed with a 15-step moving average; raw values shown in light green).

The surrogate objective $J(w)$ increases smoothly and monotonically on all three benchmarks, with the steepest gains in the first 50 iterations and near-complete convergence by iteration 80–100. Pass@1 reflects this stability: once the surrogate converges, the discrete Pass@1 metric remains essentially flat, fluctuating by at most 0.6 percentage points between $T = 50$ and $T = 300$ on all benchmarks. ACES-O surpasses both MV and ACES-C early in the optimization and maintains this advantage throughout, confirming that the objective $J(w)$ is well-aligned with downstream Pass@k performance. The results are robust to the choice of T: any value in the range $[50, 300]$ yields comparable performance.

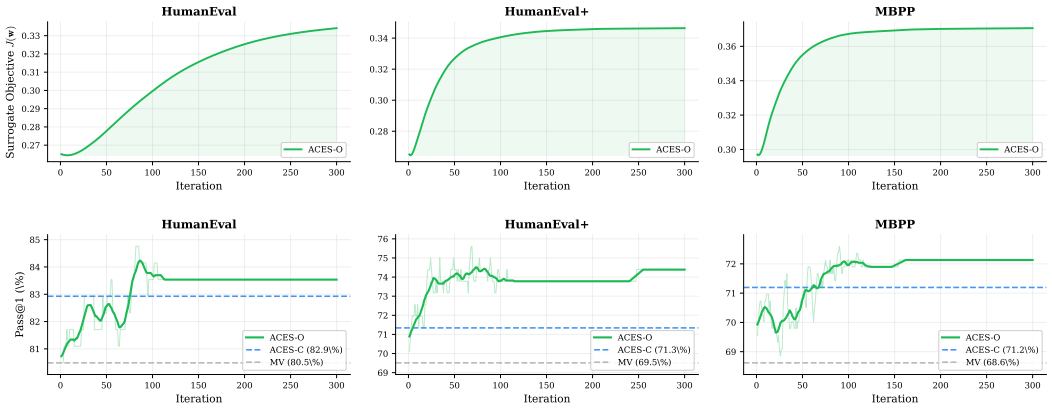


Figure 12: ACES-O convergence over 300 iterations, averaged across all tasks. **Top:** surrogate objective $J(\mathbf{w})$ converges by iteration 80–100. **Bottom:** Pass@1 (smoothed; raw in light green) surpasses both MV (dashed gray) and ACES-C (dashed blue) and remains stable for $T \in [50, 300]$.

C.10 ACES-O Hyperparameter Sensitivity

Figure 13 shows the effect of varying the two key ACES-O hyperparameters independently: the logistic surrogate sharpness γ (top row) and the learning rate η (bottom row). When sweeping one parameter, the other is held at its default value.

Both hyperparameters exhibit broad stable regions. For γ, a sharper logistic surrogate better approximates the indicator function, and all tested values in $[5, 50]$ consistently outperform MV on every benchmark. For η, the stable region is even wider: $\eta \in [0.005, 0.5]$ consistently outperforms MV

across all three benchmarks. The values used in the main experiments fall well within these stable regions.

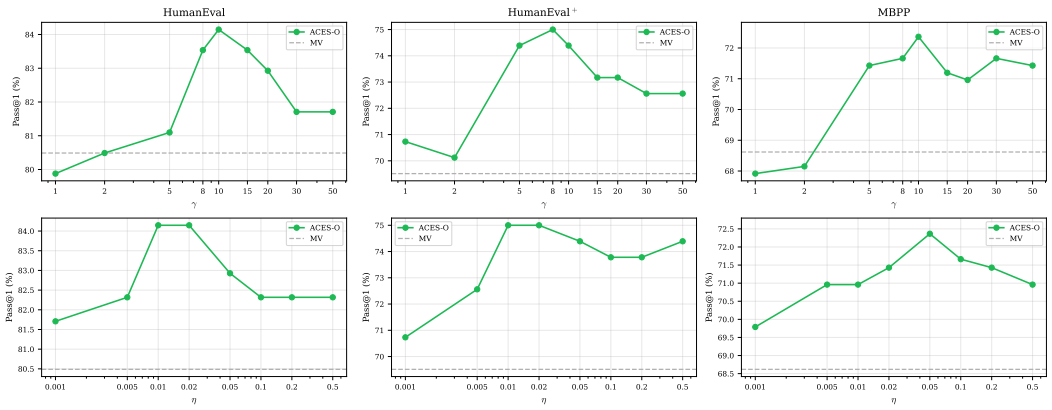


Figure 13: ACES-O hyperparameter sensitivity. **Top:** Pass@1 vs. surrogate sharpness γ (other parameters at defaults). **Bottom:** Pass@1 vs. learning rate η. Dashed gray line: MV baseline. For the tested values $\gamma \in [5, 50]$ and $\eta \in [0.005, 0.5]$, ACES-O consistently outperforms MV on all benchmarks.

C.11 Computational Cost

Figure 14 compares the reranking time per task for five methods ($n = 200$, $m = 500$), measured on a single CPU core of an Apple M3 MacBook Air (16 GB RAM) and averaged over all tasks. We time only the reranking step; shared upstream costs (code generation, test generation, code execution) are excluded. To ensure a fair comparison, all execution outputs are pre-hashed before timing, so that string comparisons in output-level methods (MBR-exec) reduce to integer comparisons.

All methods finish within one second per task. MV, CodeT, ACES-C, and ACES-O operate solely on the binary pass matrix; among these, ACES-C (9 ms) adds negligible overhead over MV (4 ms) thanks to its closed-form computation. ACES-O requires $\sim$0.85 s for iterative optimization, comparable to MBR-exec ($\sim$0.38 s), which incurs $O(n^2 m)$ pairwise output comparisons. In practice, the reranking cost for all methods is negligible relative to upstream code execution, which dominates wall-clock time. Note that all methods compared above operate solely on execution results (the binary pass matrix or output strings). Methods using static analysis or additional LLM calls involve extra processing stages and are not included in this execution-only timing comparison.

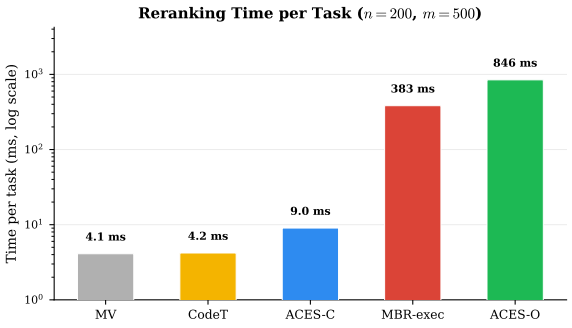


Figure 14: Reranking time per task (log scale, n=200, m=500, single CPU core). Shared upstream costs are excluded; execution outputs are pre-hashed for fair comparison. All methods finish within one second.

C.12 Pairwise Vote Statistics

To complement the assumption analysis, we directly measure the empirical distribution of the three pairwise vote types defined in Table 1. For each non-trivial task, we enumerate all (c^+, c^-, t_j) triples and classify each vote $h_j = B_{c^+,j} - B_{c^-,j}$ as informative $(+1)$, uninformative (0), or misleading (-1). Table 9 reports the aggregate proportions.

Table 9: Pairwise vote distribution across benchmarks. Informative votes consistently dominate misleading ones, supporting a positive aggregate ranking signal.

Benchmark	Informative (%)	Uninformative (%)	Misleading (%)	Inf/Mis ratio
HumanEval	22.6	72.2	5.3	4.3×
HumanEval$^+$	20.7	74.9	4.4	4.7×
MBPP	30.7	59.1	10.2	3.0×

Misleading votes are rare across all benchmarks ($\leq 10.2\%$), confirming that the majority of pairwise test–code interactions provide either correct or neutral ranking signal. Uninformative votes dominate (59–75%): even after removing constant-column tests, most non-constant tests agree on any given (c^+, c^-) pair, since a test that distinguishes some pairs may still assign the same outcome to others. This highlights the value of concentrating weight on the subset of tests that actively discriminate. MBPP has roughly twice the misleading rate of HumanEval (10.2% vs. 5.3%), consistent with its lower assumption satisfaction rate (71.1% vs. 83.1%; Table 7). The informative-to-misleading ratio remains $\geq 3\times$ on all benchmarks, providing further evidence of positive aggregate test signal.

C.13 Generation Prompts

HumanEval, HumanEval$^+$, and MBPP. We use the prompts from the MPSC protocol (Huang et al., 2024), reproduced below:

Prompt for generating candidate solutions (HumanEval / HumanEval$^+$ / MBPP)

```
I want you to act like a Python programmer. I will give you the
declaration of a function and comments about its property. You need to
 implement the body of the function in the code block. Do not modify
any code I provide. Do not provide any explanations.

Here is the question.
```Python
{Docstring}
```
```

Prompt for generating test cases (HumanEval / HumanEval$^+$ / MBPP)

```
```Python
# Given a docstring, continue to write the
# following code with 10 valid assertion
# statements to check the correctness of the
# function. Provide diverse test cases.
{Docstring}
    pass

# check the correctness of with 10 different
# valid assertion statements in the form of
# "assert {entry point}(...) == ..."
assert
```

LeetCodeDataset (Appendix C.2).
```

Prompt for generating candidate solutions (LeetCodeDataset)

```
You are an expert Python programmer. You will be given a question (
problem specification) and will generate a correct Python program that
 matches the specification and passes all tests.

### Question:
{question}

### Format: You will use the following starter code to write the
solution to the problem and enclose your code within delimiters.
```python
{starter_code}
```

Answer: (use the provided format with backticks)
```

Prompt for generating test cases (LeetCodeDataset)

```
You are an expert Python programmer. You will be given a question (
problem specification) and starter code, and you will generate diverse
 and semantically correct Python assertion statements to test the
target function.

### Question:
{question}

### Starter Code:
```python
{starter_code}
```

Target Call:
{entry_point}

Format: Output exactly 10 raw Python assertion statements. Each
line must start with `assert `. Use the exact target call that is
specified. Do not output markdown, code fences, comments, explanations
, helper functions, or any text other than the assertions.

Answer:
```