BEYOND EXECUTION AGREEMENT WITH FEW TESTS: RELATIONAL TRANSMISSION FOR CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have become increasingly capable code generators, yet reliable assessment of generated code remains essential to improving inference and learning. Many existing approaches draw on model likelihood for independent assessment and execution agreement for relational assessment on available tests. However, model likelihood can favor functionally incorrect code, while execution agreement ties correct and incorrect candidates with identical test results. We therefore seek finer relational evidence in code differences to move beyond execution agreement with few tests. To this end, we propose Trace-Conditioned Relational Transmission (TCRT). For each candidate pair, TCRT uses each response's reasoning trace or other non-code text in turn to score changed tokens on both sides of the code diff. For each response's context, the mean token log probability defines a directed local log potential. These potentials, with unary model likelihood and execution agreement, define a relational kernel. TCRT applies finite-step relational transmission through this kernel to redistribute evidence and rank the candidates. Theoretically, our unified analysis explains why execution agreement can leave likelihood-based misrankings unresolved. Our finite-step bound guarantees pairwise ranking recovery when contexts favoring the correct candidate receive sufficient weight. Empirically, TCRT consistently outperforms strong baselines across LLMs and code-generation benchmarks, with particularly strong gains when few tests are available.

1 INTRODUCTION

Large language models (LLMs) have become increasingly capable code generators (Chen et al., 2021; Li et al., 2022). Yet generation alone does not establish whether the resulting code is correct, making reliable code assessment central to progress in both inference and learning. Such assessment enables test-time computation to improve outputs by selecting better solutions from additional samples (Wu et al., 2025; Snell et al., 2025; Li et al., 2025), supports high-quality data selection for supervised fine-tuning, and provides reward signals for reinforcement learning.

Many code assessment methods use model likelihood for independent scoring or execution agreement for relational scoring. Perplexity (PPL) summarizes code-token probabilities as a proxy for model uncertainty, with lower values indicating greater confidence in a candidate. Self-consistency (SC) instead pools evidence from agreeing candidates (Wang et al., 2023). For code, matching outputs on the available inputs define execution-agreement classes (Shi et al., 2022; To et al., 2024). Some methods combine these two principles. For example, the consistency component of Reasoning-Pruning Perplexity Consistency (RPC) aggregates model-probability mass within execution-agreement classes (Zhou et al., 2026).

However, these approaches can still fail to rank correct code above incorrect code. Likelihood-based scoring can favor incorrect code because model confidence need not align with code correctness. Execution agreement provides relational evidence, but its resolution depends on test-suite adequacy. When the available tests fail to expose a behavioral difference, correct and incorrect code can enter the same execution-agreement class and receive identical agreement-based support. With two available tests, 28.30% of execution-agreement classes on HumanEvalPlus and 19.74% on MBPPPlus contain both correct and incorrect code (Fig. 1). Likelihood weighting changes the support assigned to each class, but leaves these agreement-based ties unresolved.

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

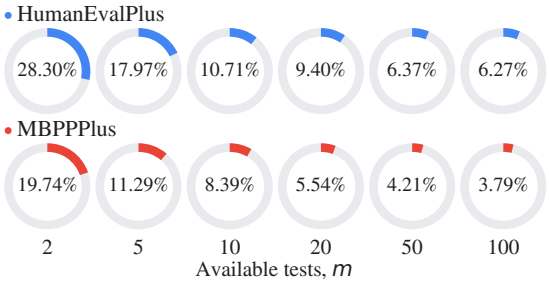

Figure 1: Share of execution-agreement classes containing both correct and incorrect code (colored).

To move beyond execution agreement with few tests, we seek finer relational evidence within the generated responses. Each response contains candidate code and accompanying non-code text, such as a reasoning trace. Together with the problem prompt, this text forms a context specific to that candidate. A code diff identifies where two candidates differ, even when their observed behavior is identical. We assess the code fragments on both sides of the diff for compatibility with each response's context in turn. Because the assessment depends on which response provides the context, the resulting relation is directed. These relations let each candidate receive evidence from multiple contexts, providing a basis for ranking candidates that execution agreement leaves tied.

Building on this idea, we propose Trace-Conditioned Relational Transmission (TCRT). For each candidate pair, TCRT uses each candidate's context in turn to score changed tokens on both sides of the code diff. For each context, the mean token log probability defines a directed local log potential. These potentials, together with unary model likelihood and execution agreement, define a relational kernel that determines how each context's evidence is distributed across candidates. Starting from initial candidate weights, TCRT uses the kernel to aggregate evidence from all contexts into candidate scores. Finite-step relational transmission repeats this aggregation, using each candidate's score as the weight of its context in the next step. The final scores rank the candidates, yielding a set-level assessment that retains execution agreement while using evidence beyond observed executions.

Theoretically, we unify PPL, SC, their likelihood-weighted combination, and TCRT by separating source weights from candidate relations. This framework shows why reweighting execution agreement cannot resolve likelihood-based misrankings within an execution-agreement class. For TCRT, we derive a finite-step ranking bound that guarantees reversal of these misrankings when contexts favoring the correct candidate receive sufficient weight. Recovery remains possible even when these contexts favor the correct candidate less strongly than others favor the incorrect one. Empirically, TCRT consistently outperforms strong baselines across LLMs and code-generation benchmarks. Its average advantage is larger when few tests are available. Our main contributions are as follows:

(1) Method: code assessment beyond execution agreement. We propose TCRT for code assessment with few tests, using relational evidence from generated contexts and code differences to distinguish candidates that execution agreement leaves tied.

(2) Theory: unified analysis and finite-step ranking recovery. We develop a unified scoring framework that characterizes the resolution limit of execution agreement. We derive a finite-step ranking bound for TCRT, establishing conditions for reversing likelihood-based pairwise misrankings that execution agreement leaves unresolved.

(3) Experiments: consistent gains with few tests. Across LLMs and code-generation benchmarks, TCRT improves ranked Pass@1 over strong baselines, averaging larger gains with few tests.

2 Preliminaries and Theoretical Motivation

Given a code-generation problem x, an LLM samples K candidate responses $\{(r_i, c_i)\}_{i=1}^{K}$, where c_i is the extracted code and r_i the accompanying generated text, often including a reasoning trace. The scoring methods considered here assign each candidate c_i a scalar score s_i based on model likelihood, execution agreement, or both. We first present these scoring principles in a unified view, then analyze the resolution limits of execution agreement under inadequate tests.

2.1 Preliminaries: Existing Scoring Principles

Unary model-likelihood evidence. Perplexity-based scoring (PPL) evaluates each candidate using its code-token likelihood under the LLM p_θ. Let h_i be the serialized context containing x, r_i, and

the code opening delimiter; let $|c_i|$ and $p_\theta(c_i \mid h_i)$ denote the code-token count and autoregressive code likelihood, respectively. Let $\mathbf{q} \in \mathbb{R}^{1\times K}$ collect the mean code-token log likelihoods and $\mathbf{s}^{\mathrm{PPL}} \in \mathbb{R}^{1\times K}$ the corresponding normalized inverse-PPL scores:

$$q_i := \frac{1}{|c_i|}\log p_\theta(c_i \mid h_i), \qquad \mathrm{PPL}_i := \exp(-q_i), \qquad s_i^{\mathrm{PPL}} := \frac{\exp(q_i)}{\sum_{j=1}^{K}\exp(q_j)}. \tag{1}$$

Normalization preserves the inverse-PPL ranking. However, code-token likelihood need not track functional correctness, so PPL may favor an incorrect candidate.

Execution-agreement evidence. For code generation, we instantiate self-consistency (SC) (Wang et al., 2023) through execution agreement on the available test case inputs. Let Exec return the ordered vector of outputs and failure states on these inputs. The execution signature of c_i is

$$b_i = \mathrm{Exec}(c_i;\ \text{available test case inputs}). \tag{2}$$

With $\mathbb{I}\{\cdot\}$ denoting the indicator function and $\mathbf{1} \in \mathbb{R}^{1\times K}$ the all-ones vector, define $A \in \{0,1\}^{K\times K}$ and $\mathbf{s}^{\mathrm{SC}} \in \mathbb{R}^{1\times K}$ by

$$A = [\mathbb{I}\{b_i = b_j\}]_{i,j=1}^{K}, \qquad \mathbf{s}^{\mathrm{SC}} = \mathbf{1}A. \tag{3}$$

Thus s_i^{SC} counts sampled candidates sharing b_i, which form an execution-agreement class. If the tests leave correct and incorrect candidates in the same class, SC assigns them equal scores.

A unified view. We express these scoring principles through source weights $\mathbf{w} \in \mathbb{R}^{1\times K}_{\geq 0}$ and a relation matrix $R \in \mathbb{R}^{K\times K}$, where R_{ij} specifies how source candidate i contributes to target candidate j. RPC's Perplexity Consistency (PC) component sums model probabilities over distinct responses within consistency classes, defined by execution agreement in its code experiments (Zhou et al., 2026). Let $\mathbf{w}^{\mathrm{PC}}$ denote these nonnegative source weights, counting each distinct response once. With I_K denoting the identity matrix, the shared form and method-specific choices of $(\mathbf{w}, R)$ are:

$$\mathbf{s}(\mathbf{w}, R) = \mathbf{w}R \in \mathbb{R}^{1\times K}, \quad \mathrm{PPL}:\ (\mathbf{s}^{\mathrm{PPL}}, I_K), \quad \mathrm{SC}:\ (\mathbf{1}, A), \quad \mathrm{PC}:\ (\mathbf{w}^{\mathrm{PC}}, A), \tag{4}$$

Here $\mathbf{w}$ weights source evidence, and R assigns it to targets. This distinction matters when PPL ranks an incorrect candidate above a correct one with the same execution signature.

2.2 Theoretical Motivation: Limits of Execution Agreement

In the unified view, execution-agreement scoring uses a fixed relation $R = A$, while the source weights $\mathbf{w}$ can vary. We therefore characterize when a fixed relation R forces distinct candidates c_a and c_b to receive equal scores for all source weights.

Corollary 2.1 (Pairwise resolution criterion)**.** *Let $R_{:a}$ and $R_{:b}$ be the corresponding columns of a fixed relation R. The score difference in Eq.* (4) *is*

$$s_a(\mathbf{w}, R) - s_b(\mathbf{w}, R) = \mathbf{w}\,(R_{:a} - R_{:b})\,. \tag{5}$$

The pair receives equal scores for every $\mathbf{w} \in \mathbb{R}^{1\times K}_{\geq 0}$ if and only if $R_{:a} = R_{:b}$.

For SC and PC aggregation, $R = A$: when $b_a = b_b$, every source supports both targets equally, yielding identical columns $A_{:a} = A_{:b}$. Changing source weights can alter class scores but cannot break within-class ties. PPL may break these ties: $R = I_K$ reduces Eq. (5) to $s_a^{\mathrm{PPL}} - s_b^{\mathrm{PPL}}$, but this ordering reflects only the pair's code likelihoods, not necessarily functional correctness.

These limitations matter when few available tests leave a correct and an incorrect candidate with the same execution signature. SC and PC aggregation assign equal scores to this pair, while PPL may rank the incorrect candidate higher. This motivates finer relational evidence from the generated responses, while retaining execution agreement as a coarse behavioral signal.

3 Methodology: TCRT

To distinguish candidates that execution agreement leaves tied, TCRT introduces finer relational evidence from generated contexts and code differences. It first constructs trace-conditioned local log potentials, then combines them with unary model likelihood and execution agreement into a relational kernel. Finite-step relational transmission through this kernel produces final candidate scores. Figure 2 illustrates the framework, and Sec. 3.4 establishes conditions for pairwise ranking recovery when PPL misranks candidates that execution agreement leaves tied.

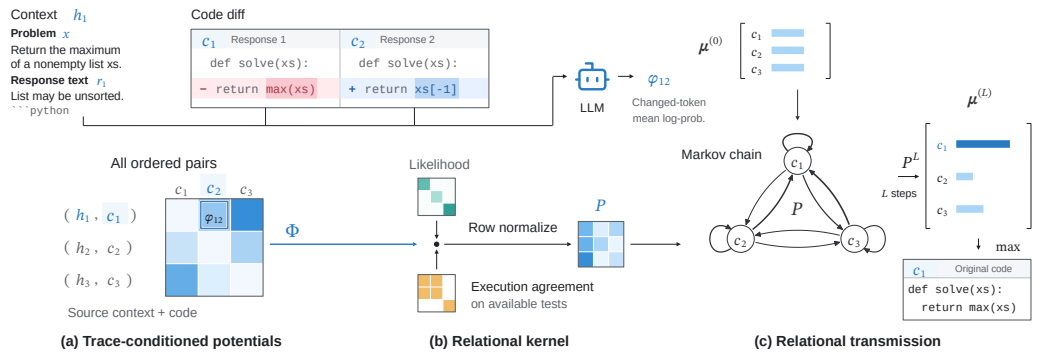

Figure 2: Overview of TCRT. (a) Trace-conditioned local log potentials Φ, with ϕ_{12} shown above. (b) Relational kernel P incorporating unary model likelihood and execution agreement. (c) Finite-step relational transmission and candidate selection. Bars show illustrative candidate weights.

3.1 Trace-Conditioned Local Potentials

To obtain finer relational evidence, TCRT uses each context h_i to assess both its own code c_i and other candidates, including those in the same execution-agreement class. For each ordered pair (i, j), a line-level code diff identifies the changed regions in c_i and c_j. This focuses likelihood scoring on local implementation differences rather than on the entire code.

TCRT scores tokens in the diff-identified code spans of c_i and c_j under source context h_i. For this pair, we collect the scored token occurrences in a list Δ, with each entry (c, ℓ) identifying the code $c \in \{c_i, c_j\}$ and token position ℓ. Although scoring is restricted to the diff spans, each token c_ℓ is evaluated with its full code prefix $c_{<\ell}$, including tokens outside those spans. For $|\Delta| > 0$, averaging these token log probabilities gives the directed local log potential

$$\phi_{ij} := \frac{1}{|\Delta|} \sum_{(c,\ell)\in\Delta} \log p_\theta(c_\ell \mid h_i, c_{<\ell}). \tag{6}$$

Thus ϕ_{ij} measures local compatibility with h_i at the implementation differences between the two candidates. For identical code, we set $\phi_{ij} = 0$.

Collecting the potentials over all ordered pairs gives the local log-potential matrix $\Phi = [\phi_{ij}] \in \mathbb{R}_{\le 0}^{K\times K}$, with sources as rows and targets as columns. Reversing a pair changes the conditioning context from h_i to h_j, so Φ need not be symmetric. Each column gathers local compatibility evidence about one target from multiple source contexts. Even when two targets share an execution signature, their code differences relative to the same source can yield different local potentials, allowing distinct columns in Φ despite identical columns in A.

3.2 Relational Kernel

TCRT combines the finer local evidence in Φ with execution agreement in A. Self-comparisons have no code diff, so the mean code-token log likelihoods $\mathbf{q}$ from Eq. (1) provide unary evidence for each source's own candidate. Together, these signals define the logit matrix Z:

$$Z := \mathrm{Diag}(\mathbf{q}) + (\mathbf{1}_{K\times K} - \beta A) \odot \Phi. \tag{7}$$

Here $\mathbf{1}_{K\times K}$ is the all-ones matrix, and $\odot$ denotes elementwise multiplication. The agreement coefficient $0 < \beta < 1$ controls how strongly TCRT favors relations between candidates with matching behavior on the available tests. Agreement modulates rather than replaces local potentials, so a source can still assign different support to candidates with identical execution signatures.

To specify how each source distributes support across targets, TCRT normalizes each row of Z. Writing this row-wise operation as $\mathcal{N}_{\mathrm{row}}$, we obtain the relational kernel

$$P := \mathcal{N}_{\mathrm{row}}(Z) \in [0, 1]^{K\times K}, \qquad P\mathbf{1}^\mathsf{T} = \mathbf{1}^\mathsf{T}. \tag{8}$$

Here P_{ij} is the fraction of source i's weight assigned to target j, not a correctness probability. For example, row-wise softmax gives $P_{ij} = \exp(Z_{ij})/\sum_k \exp(Z_{ik})$. Since each row sums to one, the source weights in Eq. (4) determine each source's total contribution, while P determines its distribution across targets.

3.3 Finite-Step Relational Transmission

With P as the relation R in Eq. (4), a single aggregation yields candidate scores $\mathbf{w}P$ from source weights $\mathbf{w}$. TCRT repeats this aggregation to let the support received by each candidate inform the influence of its associated context. Let $\boldsymbol{\mu}^{(t)} \in [0,1]^{1\times K}$ collect the candidate weights after t transmission steps, with unit sum. Starting from initial weights $\boldsymbol{\mu}^{(0)}$ (Appendix B.2), TCRT applies the fixed kernel P for an integer number of steps $L \geq 1$, giving the updates and final scores

$$\begin{aligned} &\boldsymbol{\mu}^{(t+1)} = \boldsymbol{\mu}^{(t)}P, \qquad t = 0, \ldots, L-1, \\ &\mathbf{s}^{\mathrm{TCRT}} := \boldsymbol{\mu}^{(L)} = \boldsymbol{\mu}^{(0)}P^L. \end{aligned} \tag{9}$$

In this recurrence, the score received by c_i becomes the next-step weight of its context h_i, giving contexts associated with more strongly supported candidates greater influence. Repeated aggregation therefore transmits support through intermediate candidates, yielding the relation $R = P^L$ with source weights $\mathbf{w} = \boldsymbol{\mu}^{(0)}$ in Eq. (4). This finite-state Markov process (Levin & Peres, 2026) redistributes support over the fixed candidate set; after L steps, TCRT returns the highest-scoring original code. The resulting scores provide a set-level code assessment that combines execution agreement with trace-conditioned evidence from code differences.

3.4 Theoretical Analysis: Pairwise Ranking Recovery

We establish sufficient conditions under which TCRT correctly ranks a candidate pair that execution agreement leaves tied. As in Sec. 2.2, consider a correct candidate c_a and an incorrect candidate c_b with identical execution signatures $b_a = b_b$ on the available tests. PPL can distinguish the pair through code likelihood, but this ordering need not reflect correctness. To analyze how TCRT ranks this pair, define the source-preference vector $\mathbf{d} := P_{:a} - P_{:b} \in \mathbb{R}^{K\times 1}$. Here $d_i > 0$ means that source i favors the correct candidate c_a over c_b. The last transmission step in Eq. (9) aggregates these preferences with source weights $\boldsymbol{\mu}^{(L-1)}$, giving the final score difference:

$$\mu_a^{(L)} - \mu_b^{(L)} = \boldsymbol{\mu}^{(L-1)}\mathbf{d}. \tag{10}$$

The final ordering thus depends on source preferences and the weights used in transmission.

Source informativeness. Trace-conditioned local evidence can favor c_a even when execution agreement leaves the pair tied. Since c_a and c_b have identical execution signatures, execution agreement scales both local potentials by the same positive factor within each source row. For a source $i \notin \{a, b\}$, neither target is its own candidate, so neither receives unary self evidence in Eq. (7). Row-wise softmax therefore preserves this local preference: $\phi_{ia} > \phi_{ib}$ implies $d_i > 0$. Let $\mathcal{S} \subseteq \{1, \ldots, K\} \setminus \{a, b\}$ be a nonempty set of informative sources used only for this analysis. Their own code need not be correct. To bound the final score difference, assume each source in $\mathcal{S}$ assigns at least $\gamma_+ > 0$ more support to c_a than to c_b. For each source outside $\mathcal{S}$, any excess support for c_b over c_a is at most $\gamma_- \geq 0$:

$$d_i \geq \gamma_+ \quad (i \in \mathcal{S}), \qquad d_i \geq -\gamma_- \quad (i \notin \mathcal{S}). \tag{11}$$

These bounds certify correct ranking if $\mathcal{S}$ has enough weight to outweigh opposing preferences.

Support alignment. Support received by candidates in $\mathcal{S}$ becomes the next-step weight of their informative contexts. We therefore bound how much support the set receives and retains. Let $P_i(\mathcal{S}) := \sum_{j\in\mathcal{S}} P_{ij}$ be the fraction of source i's weight sent to $\mathcal{S}$. Assume each source outside $\mathcal{S}$ sends at least a fraction $\eta > 0$ of its weight into the set, while each source inside sends at most a fraction $\varepsilon > 0$ outside, with $\eta + \varepsilon \leq 1$:

$$P_i(\mathcal{S}) \geq \begin{cases} 1-\varepsilon, & i \in \mathcal{S}, \\ \eta, & i \notin \mathcal{S}. \end{cases} \tag{12}$$

Since both targets lie outside $\mathcal{S}$, these constraints also imply $\gamma_+ \leq \varepsilon$ and allow $0 \leq \gamma_- \leq 1 - \eta$ without loss of generality. The constraints in Eq. (12) give a lower bound on the total weight of $\mathcal{S}$ after each transmission step. Together with the support differences in Eq. (11), this bound gives sufficient conditions for c_a to rank above c_b, even if other sources favor c_b.

Theorem 3.1 (Finite-step ranking recovery)**.** *Let P be any row-stochastic kernel satisfying Eqs.* (11) *and* (12)*. Define the decay factor in the bound as $\lambda := 1 - \eta - \varepsilon \in [0, 1)$, with $\lambda^0 = 1$. For any initial distribution $\boldsymbol{\mu}^{(0)}$, let $\mu^{(0)}(\mathcal{S}) := \sum_{i \in \mathcal{S}} \mu_i^{(0)}$ denote its total weight on $\mathcal{S}$. For every integer $L \geq 1$, the final score difference satisfies*

$$\mu_a^{(L)} - \mu_b^{(L)} \geq \frac{\eta\gamma_+ - \varepsilon\gamma_-}{\eta + \varepsilon} + (\gamma_+ + \gamma_-)\lambda^{L-1} \left(\mu^{(0)}(\mathcal{S}) - \frac{\eta}{\eta + \varepsilon} \right). \tag{13}$$

In particular, $\mu_a^{(L)} > \mu_b^{(L)}$ for all sufficiently large finite L if

$$\eta\gamma_+ > \varepsilon\gamma_-. \tag{14}$$

Under Eq. (14), informative contexts eventually carry enough total weight for correct pairwise ranking, even when $\gamma_+ \leq \gamma_-$. At a chosen depth L, a positive lower bound in Eq. (13) certifies the correct ordering of the pair, even if PPL ranks the incorrect candidate higher. Appendix A provides the proofs, including a kernel construction that attains the lower bound.

4 Experiments

We evaluate TCRT across four code-generation benchmarks and four LLM backbones, focusing on code selection with few available tests. Beyond baseline comparisons, we examine how performance depends on candidate and test budgets and whether focusing on code differences improves selection over full-code scoring. Additional experiments are provided in Appendix D.

4.1 Experimental Setup

Benchmarks and LLM backbones. We evaluate on HumanEval+ (Liu et al., 2023), MBPP+ (Liu et al., 2023), LiveCodeBench (Jain et al., 2025), and CodeContests+ (Wang et al., 2025), covering both function-level synthesis and contest-style program generation. Our backbones are DeepSeek-Coder-6.7B-Instruct (Guo et al., 2024), Qwen2.5-Coder-7B-Instruct (Hui et al., 2024), Qwen3-8B (Yang et al., 2025), and Qwen3.5-9B (Qwen Team, 2026).

Baselines. We compare against PPL, which scores candidates independently by mean code-token log likelihood, and an execution-based instantiation of SC, which aggregates support from candidates with matching execution signatures (Wang et al., 2023). RPC combines reasoning pruning with likelihood-weighted consistency (Zhou et al., 2026). Together, these baselines assess whether TCRT's finer candidate relations improve selection beyond likelihood, execution agreement, or their combination with few tests.

Evaluation protocol and implementation details. All methods rank the same candidate pools at matched test budgets. For the main comparison, we use $K = 128$ candidates per problem on HumanEval+ and $K = 32$ on the other benchmarks. The available test budget is $m \in \{2, 5, 100\}$ inputs per problem. This comparison excludes problems whose candidates are all correct or all incorrect, since correctness is unaffected by selection. We report Pass@1 as the percentage of evaluated problems whose top-ranked candidate is correct. We repeat each experiment 20 times with different random seeds and report the mean. Further details appear in Appendices B and C.

4.2 Main Results

Table 1 reports Pass@1 across the four backbones and benchmarks at matched test budgets. All macro-averages below weight model and benchmark settings equally.

1. Better assessment of the same candidate pools. TCRT outperforms all baselines across the evaluated backbones, benchmarks, and test budgets. The gain over the strongest baseline in each setting averages 5.24 percentage points. With candidate pools and test budgets fixed across methods, these gains reflect better selection of correct candidates rather than additional generation.

Table 1: Pass@1 (%) across four benchmarks and four LLM backbones with $m \in \{2, 5, 100\}$ tests.

	HumanEval+			MBPP+			LiveCodeBench			CodeContests+			Avg.		
Method	**2**	**5**	**100**	**2**	**5**	**100**	**2**	**5**	**100**	**2**	**5**	**100**	**2**	**5**	**100**
DeepSeek-Coder-6.7B-Instruct															
PPL	64.6	64.6	64.6	55.0	55.0	55.0	44.2	44.2	44.2	38.7	38.7	38.7	50.7	50.7	50.7
SC	76.2	77.6	80.3	65.9	65.9	69.4	53.9	53.2	55.1	45.1	45.8	49.3	60.3	60.6	63.5
RPC	76.2	77.6	80.3	65.5	65.9	69.4	54.5	53.2	55.1	45.1	46.5	49.3	60.3	60.8	63.5
TCRT	**84.4**	**83.0**	**85.0**	**73.4**	**73.4**	**73.4**	**60.3**	**59.0**	**59.0**	**55.6**	**55.6**	**53.5**	**68.4**	**67.7**	**67.7**
Δ	+8.2	+5.4	+4.8	+7.4	+7.4	+3.9	+5.8	+5.8	+3.8	+10.6	+9.2	+4.2	+8.0	+6.9	+4.2
Qwen2.5-Coder-7B-Instruct															
PPL	77.5	77.5	77.5	67.2	67.2	67.2	50.5	50.5	50.5	48.4	48.4	48.4	60.9	60.9	60.9
SC	82.4	81.4	83.3	68.7	71.7	72.7	62.0	61.0	65.5	51.6	54.8	54.2	66.2	67.2	68.9
RPC	82.4	81.4	83.3	68.7	71.7	72.7	61.0	61.0	65.5	51.0	54.8	52.9	65.8	67.2	68.6
TCRT	**84.3**	**86.3**	**85.3**	**76.8**	**77.3**	**78.3**	**70.0**	**68.0**	**70.0**	**60.0**	**61.3**	**61.9**	**72.8**	**73.2**	**73.9**
Δ	+2.0	+4.9	+2.0	+8.1	+5.6	+5.6	+8.0	+7.0	+4.5	+8.4	+6.5	+7.8	+6.6	+6.0	+4.9
Qwen3-8B															
PPL	72.6	72.6	72.6	62.0	62.0	62.0	47.9	47.9	47.9	45.7	45.7	45.7	57.1	57.1	57.1
SC	75.0	78.6	78.6	66.5	67.7	68.4	51.0	55.2	56.3	54.9	57.4	60.5	61.9	64.7	65.9
RPC	75.0	78.6	78.6	66.5	67.7	68.4	51.6	55.2	55.2	54.3	57.4	60.5	61.8	64.7	65.7
TCRT	**80.4**	**81.1**	**81.0**	**70.9**	**72.2**	**70.9**	**55.7**	**58.3**	**57.8**	**60.5**	**61.7**	**61.7**	**66.9**	**68.3**	**67.8**
Δ	+5.4	+2.5	+2.4	+4.4	+4.4	+2.5	+4.2	+3.1	+1.6	+5.6	+4.3	+1.2	+4.9	+3.6	+1.9
Qwen3.5-9B															
PPL	72.8	72.8	72.8	69.5	69.5	69.5	61.6	61.6	61.6	53.6	53.6	53.6	64.4	64.4	64.4
SC	81.5	80.4	82.6	72.5	71.9	71.9	64.3	64.3	67.4	63.7	68.7	70.5	70.5	71.3	73.1
RPC	81.5	80.4	82.6	72.5	71.9	71.9	64.3	64.0	67.0	63.7	68.7	70.5	70.5	71.2	73.0
TCRT	**91.3**	**92.4**	**93.5**	**74.9**	**75.5**	**74.3**	**69.4**	**70.1**	**71.4**	**68.4**	**69.4**	**72.7**	**76.0**	**76.8**	**78.0**
Δ	+9.8	+12.0	+10.9	+2.4	+3.6	+2.4	+5.1	+5.8	+4.1	+4.7	+0.7	+2.2	+5.5	+5.5	+4.9

Notes. Avg. is the macro-average over four benchmarks. Bold marks the best overall result; underlining marks the strongest baseline. Green Δ is the absolute percentage-point gain over the strongest baseline; Δ Avg. averages the four gains. Gains are computed before rounding.

2. Better assessment can compensate for fewer tests. With just 2 tests, TCRT outperforms the strongest baseline using 100 tests in most settings, with a macro-averaged advantage of 3.14 percentage points. At matched budgets, its average gain over the strongest baseline is also larger with fewer tests: 6.24 points at $m = 2$ versus 3.98 at $m = 100$. The cross-budget comparison therefore supports better candidate assessment as a way to compensate for limited test availability.

3. Gains persist at the largest test budget. At $m = 100$, TCRT retains a 3.98-point average advantage over the strongest baseline, with positive gains in every model and benchmark setting. More tests need not fully distinguish correct from incorrect candidates. Distinct inputs may exercise similar behaviors and leave execution-agreement ties unresolved. Under such conditions, TCRT's trace-conditioned relational evidence can support selection beyond execution agreement, offering a plausible explanation for the persistent gains.

4.3 Analysis and Ablation Studies

The main results establish TCRT's selection gains at matched candidate and test budgets. We next examine how performance changes as these budgets vary individually and jointly, and assess the value of focusing likelihood scoring on code differences rather than the full code of each candidate.

1. Relational assessment complements additional testing. **(i)** On HumanEval+ (Fig. 3), SC and RPC plateau below TCRT, showing that finer candidate assessment remains valuable after gains from additional tests largely saturate. PPL remains flat because it does not use test execution. **(ii)** With one or two tests, TCRT stays near its higher plateau, while SC and RPC lose more relative

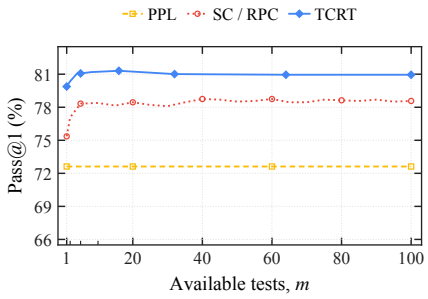

Figure 3: Test-budget scaling on HumanEval+. SC and RPC overlap.

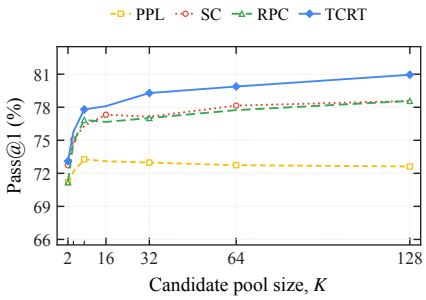

Figure 4: Candidate-pool scaling on HumanEval+ with Qwen3-8B at $m = 100$.

to theirs. This supports candidate relations as complementary evidence for maintaining selection quality without extensive testing. **(iii)** SC and RPC gain mainly from the first few tests, with little improvement thereafter. This is consistent with added tests exercising similar behaviors without resolving correctness differences. Test quantity need not ensure test-suite adequacy; further gains then require evidence that distinguishes unresolved candidates, not simply more inputs.

2. Finer assessment improves candidate-pool scaling. **(i)** In the Qwen3-8B study on HumanEval+ (Fig. 4), PPL levels off after its initial gains, whereas SC, RPC, and TCRT continue to improve overall. These contrasting trajectories show that the benefit of additional sampling depends on how candidates are assessed, not pool size alone. **(ii)** At larger K, TCRT separates from SC and RPC after tracking them closely with smaller pools. This supports finer candidate relations as a way to extract more value from additional samples. The pattern is consistent with TCRT's use of candidates both as selectable solutions and as contexts for assessing one another.

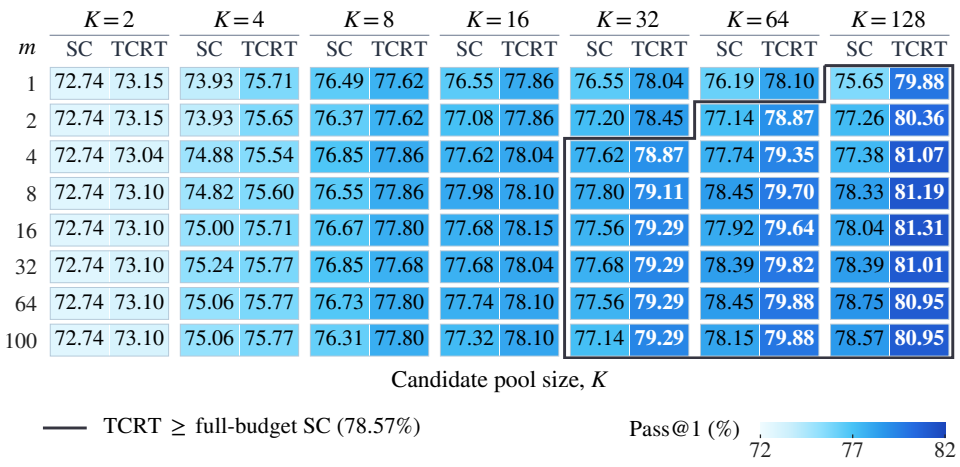

Figure 5: Pass@1 (%) across K and m: SC (left) and TCRT (right) share a color scale. The stair-step boundary encloses TCRT settings matching or exceeding SC's largest-budget result, 78.57% at $(K, m) = (128, 100)$.

3. Strong performance with substantially fewer candidates and tests. **(i)** Even with very few tests, TCRT achieves substantial gains from candidate expansion (Fig. 5). With only one test, increasing K from 2 to 128 yields 6.73 percentage points for TCRT versus 2.91 for SC. **(ii)** TCRT can outperform SC with sharply reduced candidate and test budgets. The stair-step boundary encloses settings where TCRT matches or exceeds SC's largest-budget result, 78.57% at $(K, m) = (128, 100)$. At $(32, 4)$, TCRT achieves 78.87% with 75% fewer candidates and 96% fewer tests than SC's reference configuration.

4. Ablation: focusing relational evidence on code differences. TCRT focuses trace-conditioned scoring on code differences to emphasize the implementation choices that distinguish candidates. To evaluate this design, we compare TCRT with a retuned full-code variant (Table 2). Changed-span scoring improves Pass@1 by 1.19 to 1.85 percentage points on three benchmarks and matches full-code scoring on LiveCodeBench. These results support difference-focused local potentials as a more effective basis for candidate assessment than full-code scoring.

Table 2: Pass@1 (%) in the locality ablation with changed-span and retuned full-code scoring.

Benchmark	TCRT	Full-code
HumanEval+	**82.14**	80.95
MBPP+	**70.89**	69.62
LiveCodeBench	**57.81**	**57.81**
CodeContests+	**61.73**	59.88

5 Related Work

Code assessment from likelihood and execution. The benefit of additional sampling depends on candidate assessment and aggregation (Farinhas et al., 2024; Zhou et al., 2025). Model-based evidence includes code-generation and reverse instruction likelihoods (Zhang et al., 2023), while LEVER combines generation probability with learned execution-conditioned verification before aggregating candidates with matching results (Ni et al., 2023). Consistency-based aggregation pools reasoning paths by final answer in self-consistency (Wang et al., 2023); for code, MBR-EXEC uses agreement of execution results across the available inputs as its minimum Bayes risk (MBR) criterion, with likelihood-based tie-breaking (Shi et al., 2022). RPC combines model probability with execution-defined consistency classes in its code experiments (Zhou et al., 2026). However, when inadequate tests give correct and incorrect code identical execution signatures, their agreement-based support remains tied even after reweighting the contributing candidates.

Strengthening execution evidence. Expanded evaluation suites expose errors missed by original tests (Liu et al., 2023; Stroebl et al., 2026), while TC-Bench evaluates tests against diverse failure patterns in a compact set of incorrect programs (Luo et al., 2026). Test construction and aggregation strengthen this evidence. CodeT groups generated programs that pass the same subset of generated input-output tests and scores each group by its size times its passed-test count (Chen et al., 2023); S* and ExPairT-LLM adaptively obtain distinguishing inputs for candidate selection (Li et al., 2025; Yuviler & Drachsler-Cohen, 2026). CURE and UTRL learn test generators by interacting with candidate code (Wang et al., 2026; Lee et al., 2026), while ACES uses leave-one-out AUC consistency with rankings from the remaining tests to determine test weights (Sun et al., 2026). These works assess or strengthen test evidence; generated responses offer complementary evidence.

Relational assessment across candidates. Relational assessment depends on both the evidence defining candidate relations and the aggregation rule. In text generation, MBR variants incorporate model probabilities into expected-utility estimation or adjust utilities to reflect latent output structure (Jinnai et al., 2024; Eikema et al., 2025). Similarity-based aggregation uses token overlap for lightweight reranking, or internal-activation similarity within answer groups to augment answer frequency (Jain et al., 2024; Jiang et al., 2026). Execution-based relations can also be graded: SRank aggregates validation scores across execution-defined clusters using the fraction of matching outputs on the test inputs (To et al., 2024), while Liu et al. (2026) weight pairwise execution consistency with static code-quality scores. Without adding tests, TCRT retains execution agreement and augments it with directed trace-conditioned local potentials over both sides of code differences. Finite-step relational transmission uses each candidate's received support to reweight its context for later steps.

6 Conclusion

We presented TCRT to assess code beyond execution agreement with few tests. It augments execution agreement with local log potentials from code diffs under candidate-specific contexts and aggregates evidence through finite-step relational transmission. Our unified analysis characterizes execution agreement's resolution limit and gives sufficient conditions for finite-step recovery from likelihood-based pairwise misrankings. Across LLMs and code-generation benchmarks, TCRT consistently outperforms strong baselines and achieves larger average gains with few tests. Results also show that finer candidate assessment can compensate for fewer tests. Future work can study how relational evidence supports training code-generation models via data curation and reward design.

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

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *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.

Bryan Eikema, Anna Rutkiewicz, and Mario Giulianelli. Structure-conditional minimum bayes risk decoding. In *Proceedings of the 2025 conference on empirical methods in natural language processing*, pp. 31694–31711, 2025.

António Farinhas, Haau-Sing Li, and André F Martins. Reranking laws for language generation: A communication-theoretic perspective. *Advances in Neural Information Processing Systems*, 37: 111074–111105, 2024.

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.

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.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Siddhartha Jain, Xiaofei Ma, Anoop Deoras, and Bing Xiang. Lightweight reranking for language model generations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6960–6984, 2024.

Junqi Jiang, Tom Bewley, Salim I Amoukou, Francesco Leofante, Antonio Rago, Saumitra Mishra, and Francesca Toni. Representation consistency for accurate and coherent llm answer aggregation. *Advances in Neural Information Processing Systems*, 38:51560–51591, 2026.

Yuu Jinnai, Tetsuro Morimura, Ukyo Honda, Kaito Ariu, and Kenshi Abe. Model-based minimum Bayes risk decoding for text generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 22326–22347, 2024.

Dongjun Lee, Changho Hwang, and Kimin Lee. Learning to generate unit test via adversarial reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 148091–148111, 2026.

David A Levin and Yuval Peres. *Markov chains and mixing times*. American Mathematical Society, 2026.

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*, 2025.

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.

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. *Advances in neural information processing systems*, 36:21558–21572, 2023.

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*, volume 40, pp. 32096–32104, 2026.

Xianzhen Luo, Jinyang Huang, Wenzhen Zheng, Qingfu Zhu, Mingzheng Xu, Yiheng Xu, YuanTao Fan, and Wanxiang Che. How many code and test cases are enough? evaluating test cases generation from a binary-matrix perspective. In *International Conference on Learning Representations*, volume 2026, pp. 38008–38050, 2026.

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*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26106–26128, 2023.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

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, 2022.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling. In *International Conference on Learning Representations*, volume 2026, pp. 92601–92633, 2026.

Hui Sun, Yun-Ji Zhang, Zheng Xie, Ren-Biao Liu, Yali Du, Xin-Ye Li, and Ming Li. Aces: Who tests the tests? leave-one-out auc consistency for code generation. *arXiv preprint arXiv:2604.03922*, 2026.

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

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

Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. Co-evolving llm coder and unit tester via reinforcement learning. *Advances in Neural Information Processing Systems*, 38:143630–143664, 2026.

Zihan Wang, Siyao Liu, Yang Sun, Ming Ding, and Hongyan Li. CodeContests+: High-quality test case generation for competitive programming. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025.

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 *International Conference on Learning Representations*, volume 2025, pp. 55824–55845, 2025.

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

Tom Yuviler and Dana Drachsler-Cohen. Expairt-llm: Exact learning for llm code selection by pairwise queries. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 34557–34565, 2026.

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*, volume 202 of *Proceedings of Machine Learning Research*, pp. 41832–41846, 2023.

Yilun Zhou, Austin Xu, Peifeng Wang, Caiming Xiong, and Shafiq Joty. Evaluating judges as evaluators: The JETTS benchmark of LLM-as-judges as test-time scaling evaluators. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 79436–79471, 2025.

Zhi Zhou, Tan Yuhao, Zenan Li, Yuan Yao, Lan-Zhe Guo, Yu-Feng Li, and Xiaoxing Ma. A theoretical study on bridging internal probability and self-consistency for llm reasoning. *Advances in Neural Information Processing Systems*, 38:87380–87413, 2026.

Appendix

A Proofs and Supporting Analysis **14**
A.1 Pairwise Resolution . . . 14
A.2 From Local Potentials to Source Preferences . . . 14
A.3 Finite-Step Ranking Recovery . . . 14
A.4 Tightness of the Recovery Bound . . . 15

B Method Details **16**
B.1 Local-Potential Computation . . . 16
B.2 Transmission and Selection . . . 16

C Experimental Protocol **17**
C.1 Evaluation Population and Correctness . . . 17
C.2 Budgets and Repeated Sampling . . . 17
C.3 Scoring and Comparison Settings . . . 18

D Additional Experiments **19**
D.1 Effect of Transmission Depth . . . 19
D.2 Sensitivity to the Agreement Coefficient . . . 19
D.3 Correctness Differences Within Execution Classes . . . 20

A Proofs and Supporting Analysis

This section develops the analysis supporting the resolution limit of execution agreement and the ranking-recovery guarantee for TCRT. We first prove Corollary 2.1, then connect local potentials to source preferences, prove Theorem 3.1, and construct a kernel that attains its bound.

A.1 Pairwise Resolution

Proof of Corollary 2.1. For any two target columns, Eq. (4) gives $s_a(\mathbf{w}, R) - s_b(\mathbf{w}, R) = \mathbf{w}(R_{:a} - R_{:b})$. Equal columns therefore tie for every nonnegative source vector. Conversely, if the columns differ in row i, choosing the nonnegative source vector with $w_i = 1$ and $w_j = 0$ for $j \neq i$ produces a nonzero margin. □

A.2 From Local Potentials to Source Preferences

Consider the softmax normalization in Sec. 3.2. For two targets with $b_a = b_b$ and a source $i \notin \{a, b\}$, execution agreement gives $A_{ia} = A_{ib}$, while the unary diagonal term affects neither target. Define $\chi_i := 1 - \beta A_{ia} > 0$. The common row denominator cancels in the ratio, giving

$$\log \frac{P_{ia}}{P_{ib}} = Z_{ia} - Z_{ib} = \chi_i(\phi_{ia} - \phi_{ib}). \tag{15}$$

Consequently, $d_i = P_{ia} - P_{ib}$ has the same sign as $\phi_{ia} - \phi_{ib}$: a local preference survives normalization even when execution agreement cannot distinguish the targets. This establishes the softmax preference claim in Sec. 3.4. The quantitative informativeness and support-alignment conditions in Eqs. (11) and (12) are the separate assumptions used below.

A.3 Finite-Step Ranking Recovery

Proof of Theorem 3.1. Let u_t denote the total weight on the informative-source set $\mathcal{S}$ after t steps:

$$u_t := \mu^{(t)}(\mathcal{S}) := \sum_{i \in \mathcal{S}} \mu_i^{(t)}. \tag{16}$$

Nonnegativity and row stochasticity preserve the unit sum of $\boldsymbol{\mu}^{(t)}$. The support-alignment condition in Eq. (12) gives

$$\begin{aligned} u_{t+1} &= \sum_{i=1}^{K} \mu_i^{(t)} P_i(\mathcal{S}) \\ &\geq (1-\varepsilon)u_t + \eta(1-u_t) = \eta + \lambda u_t, \qquad \lambda := 1 - \eta - \varepsilon. \end{aligned} \tag{17}$$

Define $u_\star := \eta/(\eta + \varepsilon)$, the fixed point of the scalar lower-bound recurrence. Subtracting $u_\star = \eta + \lambda u_\star$ from Eq. (17) gives $u_{t+1} - u_\star \geq \lambda(u_t - u_\star)$. Since $0 \leq \lambda < 1$, induction yields

$$u_t \geq u_\star + \lambda^t(u_0 - u_\star). \tag{18}$$

The convention $\lambda^0 = 1$ includes the initial step; when $\lambda = 0$, the bound is $u_\star = \eta$ for every $t \geq 1$.

For the fixed pair (a, b), let $\mathbf{d} = P_{:a} - P_{:b}$, as in Sec. 3.4. For any integer $L \geq 1$, the final pairwise margin is

$$\begin{aligned} \mu_a^{(L)} - \mu_b^{(L)} &= \boldsymbol{\mu}^{(L-1)}\mathbf{d} \\ &\geq \gamma_+ u_{L-1} - \gamma_-(1 - u_{L-1}) \\ &= (\gamma_+ + \gamma_-)u_{L-1} - \gamma_-, \end{aligned} \tag{19}$$

where the inequality uses Eq. (11). Substituting Eq. (18) into Eq. (19) gives Eq. (13). As L increases, this lower bound tends to $(\eta\gamma_+ - \varepsilon\gamma_-)/(\eta + \varepsilon)$. Under Eq. (14), this limit is strictly positive and $\lambda^{L-1} \to 0$, so the margin is positive for all sufficiently large finite L. □

Sufficient transmission depth. The bound also identifies a sufficient depth for a fixed initialization. Let $u_{\mathrm{req}} := \gamma_-/(\gamma_+ + \gamma_-)$ be the source mass at which the lower bound in Eq. (19) equals zero. The lower bound in Eq. (13) is positive at depth L if and only if

$$u_\star + \lambda^{L-1}(u_0 - u_\star) > u_{\mathrm{req}}. \tag{20}$$

If $u_0 > u_{\mathrm{req}}$, depth $L = 1$ is already certified. If $\lambda = 0$ and Eq. (14) holds, then $u_\star = \eta > u_{\mathrm{req}}$, so every $L \geq 2$ is certified. For $0 < \lambda < 1$, if $u_0 \leq u_{\mathrm{req}}$ and Eq. (14) holds, then $u_\star > u_{\mathrm{req}}$, and any integer L satisfying

$$\lambda^{L-1} < \frac{u_\star - u_{\mathrm{req}}}{u_\star - u_0}, \qquad \text{equivalently} \qquad L > 1 + \frac{\log[(u_\star - u_{\mathrm{req}})/(u_\star - u_0)]}{\log \lambda}, \tag{21}$$

has a positive certified margin. This is a certificate for the pair's ordering at a specified depth, not a requirement to reach a stationary distribution. The actual score difference need not increase at every step.

A.4 Tightness of the Recovery Bound

The bound is sharp over row-stochastic kernels satisfying the assumptions of Theorem 3.1. First, these assumptions constrain the possible preference margins because both targets lie outside $\mathcal{S}$. For any $i \in \mathcal{S}$, Eq. (12) leaves at most ε mass outside $\mathcal{S}$, so

$$\gamma_+ \leq d_i = P_{ia} - P_{ib} \leq P_{ia} \leq \varepsilon. \tag{22}$$

Likewise, for $i \notin \mathcal{S}$, support alignment gives $d_i \geq -P_{ib} \geq -(1 - \eta)$. Thus γ_- can be tightened to $\min\{\gamma_-, 1 - \eta\}$ without any additional assumption.

To attain the bound, let $\boldsymbol{\nu} \in [0, 1]^{1\times K}$ be any probability vector supported on the nonempty set $\mathcal{S}$. For $0 < \gamma_+ \leq \varepsilon$ and $0 \leq \gamma_- \leq 1 - \eta$, define a row-stochastic kernel as follows. For every $i \in \mathcal{S}$, set

$$P_{ij} = (1 - \varepsilon)\nu_j \quad (j \in \mathcal{S}), \qquad P_{ia} = \frac{\varepsilon + \gamma_+}{2}, \qquad P_{ib} = \frac{\varepsilon - \gamma_+}{2}. \tag{23}$$

For every $i \notin \mathcal{S}$, set

$$P_{ij} = \eta\nu_j \quad (j \in \mathcal{S}), \qquad P_{ia} = \frac{1 - \eta - \gamma_-}{2}, \qquad P_{ib} = \frac{1 - \eta + \gamma_-}{2}. \tag{24}$$

All unspecified entries are zero. Every row in Eqs. (23) and (24) is nonnegative and sums to one. Moreover, $P_i(\mathcal{S})$ equals $1 - \varepsilon$ on $\mathcal{S}$ and η outside $\mathcal{S}$, while d_i equals γ_+ on $\mathcal{S}$ and $-\gamma_-$ outside $\mathcal{S}$. Consequently, both inequalities in the proof of Theorem 3.1 hold with equality for every initialization and every depth $L \geq 1$. Hence no uniformly larger lower bound follows from $(\gamma_+, \gamma_-, \eta, \varepsilon, u_0)$ alone for this class of kernels.

B METHOD DETAILS

This section details how TCRT computes local evidence and converts it into candidate scores. We first describe context construction, token alignment, and pooling, then specify transmission, final selection, and computational cost.

B.1 LOCAL-POTENTIAL COMPUTATION

Conditioning and rescoring. For source response i, we form h_i by concatenating the serialized chat prompt, the accompanying text r_i with boundary whitespace removed, and the code opening delimiter. Each candidate code c_j is teacher-forced independently after h_i using its own full autoregressive prefix. The self record (i, i) and cross record (i, j) therefore evaluate both sides of a comparison under the same source context. The same serialization and code delimiter are used for scoring and token-offset alignment.

Changed-token alignment and pooling. We compute a line-level diff using Python's sequence matcher with `autojunk=False`. The resulting changed character spans are aligned with all overlapping tokens using their character offsets. Deleted and replaced source spans supply self log likelihoods; inserted and replaced target spans supply source-conditioned target log likelihoods. Tokens are collected span by span: each overlap with a recorded log probability contributes one occurrence to the list Δ in Eq. (6) and to its averaging denominator. We pool the source and target occurrences in a single token-weighted mean, rather than averaging two side-level means. A one-sided change uses the changed side alone. Identical code gives $\phi_{ij} = 0$, while the self logit is $Z_{ii} = q_i$ as defined in Eq. (7).

B.2 TRANSMISSION AND SELECTION

For agreement-based initialization, let $G_i := \{j : b_j = b_i\}$ be candidate i's execution-agreement class. The initial weight $\mu_i^{(0)} = |G_i|^{-1} \sum_{j \in G_i} s_j^{\mathrm{PPL}}$ distributes each class's total normalized likelihood weight uniformly among its members.

For a fixed pool and scoring configuration, TCRT constructs P once and reuses it at every step of Eq. (9). Successive vector-matrix products compute the final weights without explicitly forming P^L. Setting $L = 0$ retains the initial distribution and defines the no-transmission reference in Appendix D.1.

Selection. We return the original candidate with maximal $\mu_i^{(L)}$, resolving ties by larger q_i and then by the smaller stored candidate identifier.

Computation. For a fixed candidate pool, the self and cross likelihood records can be cached for the K^2 ordered source-target pairs. Once the local potentials are available, the dense kernel requires $O(K^2)$ storage and L vector-matrix updates require $O(LK^2)$ arithmetic operations. These updates use the fixed kernel and do not generate or execute additional code.

C Experimental Protocol

This section specifies the evaluation protocol underlying the comparisons in Sec. 4. We first define the evaluated populations and correctness labels, then describe budgets and repeated sampling, and finally give the baseline, tuning, and ablation settings.

C.1 Evaluation Population and Correctness

The main comparison isolates problems for which candidate selection can affect correctness: each evaluated pool contains at least one correct and one incorrect candidate. An all-correct pool is solved by every selector, while an all-incorrect pool is solved by none. Table 3 reports the resulting problem counts for each backbone and benchmark. Within a setting, all methods rank the same candidate pools and are evaluated on the same problems. Pass@1 is the percentage of these evaluated problems whose top-ranked candidate is correct.

Table 3: Number of evaluated problems in the main selection comparison. Each problem has both correct and incorrect candidates in its pool.

Backbone	HumanEval+	MBPP+	LiveCodeBench	CodeContests+
DeepSeek-Coder-6.7B-Instruct	147	229	156	142
Qwen2.5-Coder-7B-Instruct	102	198	200	155
Qwen3-8B	84	158	192	162
Qwen3.5-9B	92	167	294	278

As in Sec. 4.1, each selection experiment is repeated 20 times with different random seeds, and the reported Pass@1 is the mean. The Avg. columns in Table 1 weight the four benchmarks equally within each backbone. Cross-backbone summaries weight all 16 backbone-benchmark settings equally. For a gain over the strongest baseline, we first subtract the largest baseline score in each setting and then average these differences. Calculations use the reported results before the additional rounding applied in the main table.

Execution signatures use pre-generated CodeT-style test-input pools shared across methods, with up to 100 available inputs per problem. Correctness labels instead come from the official benchmark evaluators: the augmented test suites for HumanEval+ and MBPP+, and the official held-out evaluation procedures for LiveCodeBench and CodeContests+. Outputs on the available tests determine execution-agreement classes, not correctness labels.

C.2 Budgets and Repeated Sampling

For the main comparison, we sample $K = 128$ candidate programs per HumanEval+ problem and $K = 32$ per problem in the other three benchmarks. We compare the methods at matched test budgets $m \in \{2, 5, 100\}$. Here m counts the available inputs used to form execution signatures in Eq. (2). PPL does not use test execution and is therefore independent of m.

The HumanEval+ test-budget study in Fig. 3 fixes $K = 128$ and varies the number of available tests from 1 to 100. The candidate-pool study in Fig. 4 reports $K \in \{2, 4, 8, 16, 32, 64, 128\}$ at $m = 100$. The joint-budget study in Fig. 5 reports the same candidate budgets and $m \in \{1, 2, 4, 8, 16, 32, 64, 100\}$. The HumanEval+ test-budget, candidate-pool, and transmission-depth studies each evaluate 84 problems.

Within each budget study, the evaluated problems, original candidate pools, and correctness labels remain fixed across runs and budgets. In candidate-budget experiments, we shuffle each problem's candidate pool once per seed and use its first K entries, making candidate subsets nested across budgets within a run. In test-budget experiments, we separately sample m inputs without replacement from the shared 100-input pool for each seed, problem, and budget. Test subsets at different budgets need not be nested. We recompute execution signatures and relational kernels for each sampled test set while reusing the cached pairwise code scores.

C.3 Scoring and Comparison Settings

PPL selects the candidate with the largest unary score q_i. SC ranks execution-agreement classes by size. For RPC, we use the full implementation, including its Reasoning Pruning and Perplexity Consistency components (Zhou et al., 2026). All methods start from the same candidate pools; SC, RPC, and TCRT use execution agreement on the same available inputs.

For each backbone and benchmark, we selected the scoring configuration by finite-grid search using correctness labels from the same mixed subset used for reporting results. The selected configuration is reused across candidate and test budgets without per-budget retuning. Each sensitivity sweep varies its named parameter while keeping the remaining settings fixed within that sweep.

Full-code comparison. Table 2 compares TCRT's two-sided changed-span potential with a target-only full-code alternative. For each ordered pair, the alternative averages the recorded cross-candidate log probabilities over all target-code tokens, whereas TCRT pools changed-token occurrences from both the source and target. The Qwen3-8B comparison shares candidate pools, execution-agreement classes, budgets, and correctness labels between the two scoring rules. The full-code configuration is tuned independently over the same grid rather than copied from TCRT.

D ADDITIONAL EXPERIMENTS

This section extends the analysis of relational transmission and test-suite adequacy. We examine sensitivity to transmission depth and the agreement coefficient, then report within-class correctness differences on LiveCodeBench and CodeContests+.

D.1 EFFECT OF TRANSMISSION DEPTH

We vary the depth L in Eq. (9) to assess repeated relational aggregation. Using Qwen3-8B on the 84-problem HumanEval+ selection subset with $K = 128$ candidates and $m = 100$ available tests per problem, we evaluate every integer L from 0 to 32. The sweep uses entmax-1.5 normalization on the raw relational logits and agreement coefficient $\beta = 0.6$; all other settings remain fixed across depths. At $L = 0$, scores retain their initialization; $L = 1$ performs one aggregation.

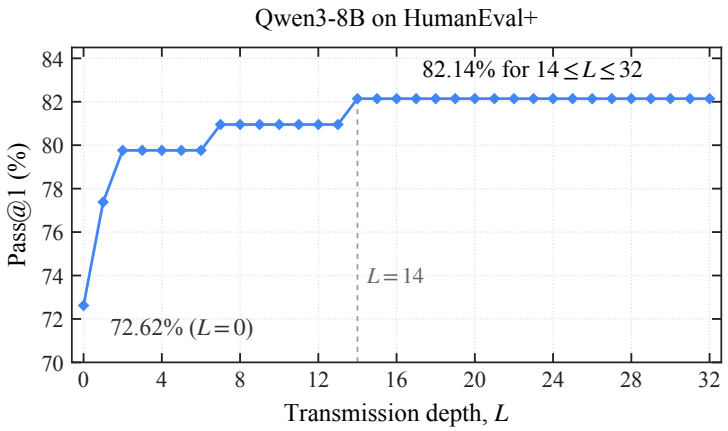

Figure 6: Pass@1 (%) of TCRT at different transmission depths on HumanEval+ with Qwen3-8B, $K = 128$, and $m = 100$. Each marker represents an evaluated integer depth; the vertical guide marks $L = 14$, the start of the observed accuracy plateau.

Pass@1 rises from 72.62% at $L = 0$ to 77.38% at $L = 1$ and reaches 82.14% at $L = 14$, gaining 9.52 percentage points over no transmission and 4.76 over a single step (Fig. 6). Pass@1 remains at 82.14% at every evaluated depth from 14 through 32, supporting $L = 20$ as a practical finite-depth choice in this setting.

D.2 SENSITIVITY TO THE AGREEMENT COEFFICIENT

Using Qwen3-8B on the 84-problem HumanEval+ selection subset with $K = 128$ and $m = 100$, we vary the agreement coefficient $\beta \in \{0, 0.3, 0.6, 0.8, 0.9, 0.95\}$ in Eq. (7). All other settings remain fixed throughout this sweep. The boundary case $\beta = 0$ removes the agreement adjustment from the cross-candidate logits and serves as an unmodulated reference. PPL and SC use the same candidate pools and provide fixed references because neither depends on β.

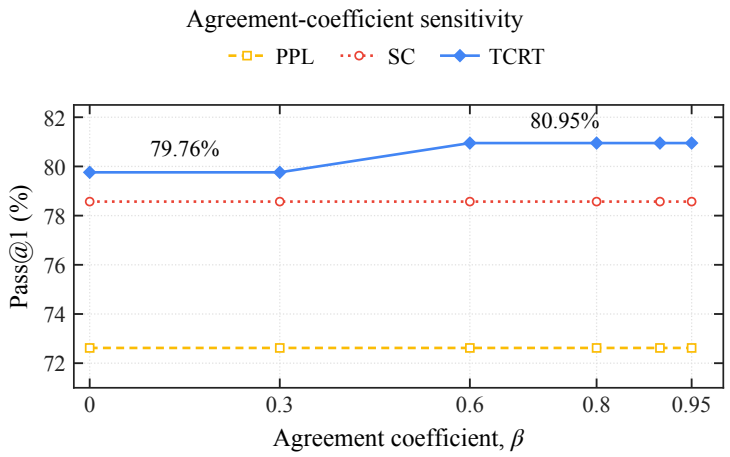

Figure 7: Pass@1 (%) at different agreement coefficients on HumanEval+ with Qwen3-8B, $K = 128$, and $m = 100$. Markers show the evaluated coefficients; PPL and SC provide fixed references. At $\beta = 0$, the relational logits have no agreement adjustment.

Pass@1 reaches 80.95% at each of $\beta \in \{0.6, 0.8, 0.9, 0.95\}$, compared with 79.76% at $\beta = 0$ and 0.3 (Fig. 7). TCRT exceeds PPL (72.62%) and SC (78.57%) at every evaluated coefficient. Together with the depth study, these results motivate $L = 20$ and $\beta = 0.6$ as a practical default for this setting.

D.3 Correctness Differences Within Execution Classes

We extend the execution-class analysis in Fig. 1 to LiveCodeBench and CodeContests+ using Qwen3-8B with $K = 32$ candidates per problem. The evaluated populations contain 192 and 162 problems, respectively, as listed in Table 3. A mixed class contains at least one correct and one incorrect candidate. For each $m \in \{2, 5, 10, 20, 50, 100\}$, we report $100N_{\text{mixed}}/N_{\text{all}}$, where N_{mixed} and N_{all} are the aggregate counts of mixed classes and all observed execution-agreement classes, respectively.

With 2 available tests, 16.50% and 9.46% of their execution-agreement classes, respectively, contain both correct and incorrect code (Fig. 8). At 100 tests, the corresponding shares are 2.37% and 0.68%, respectively. The remaining within-class correctness differences motivate assessing candidates beyond execution agreement, even at the larger test budget.

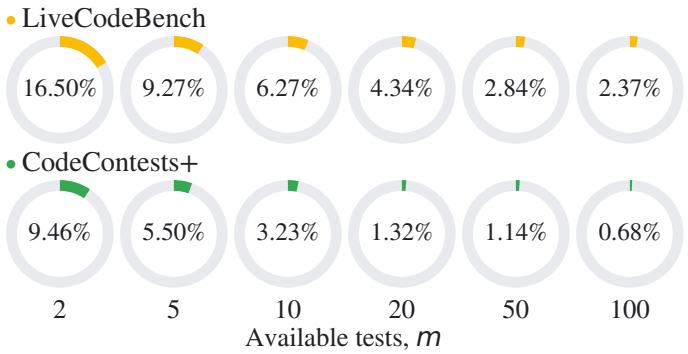

Figure 8: Share of execution-agreement classes containing both correct and incorrect code (colored) on LiveCodeBench and CodeContests+, using Qwen3-8B with $K = 32$.