CRITERALIGN: CRITERION-CENTRIC RATIONALE ALIGNMENT FOR CODE PREFERENCE JUDGING

Anonymous authors
Paper under double-blind review

ABSTRACT

Pairwise human preference prediction is central to evaluating code-generation systems, where quality often depends on task-specific trade-offs beyond functional correctness. While rubric-based LLM judges improve interpretability by decomposing evaluation into explicit criteria, most existing pipelines remain pointwise: they score each response independently and derive preferences by comparing aggregated scores. We show that this design is poorly matched to pairwise code preference prediction and can underperform a strong monolithic judge. We propose CRITERALIGN, a criterion-centric framework that adapts rubric-based judging to pairwise preference evaluation through direct criterion-level pairwise judgments, tie-driven criterion refinement, swap-consistency filtering, and final pairwise synthesis. We further introduce Human-Preference-Aligned Guidance (HPAG), synthesized offline from training examples by extracting recurring rationale gaps between human preferences and monolithic judge predictions, and injected into the criterion generator, criterion judge, and final judge. On BigCodeReward, CRITERALIGN improves a Qwen2.5-VL-32B monolithic judge from 60.4% to 66.3% accuracy, with ablations confirming the contributions of pairwise criterion design and HPAG.

1 INTRODUCTION

Large language models are increasingly used to generate code, user interfaces, visual programs, and interactive applications (Chen et al., 2021; Jimenez et al., 2024; Li et al., 2025; Zhuo et al., 2025). As these systems become more capable, evaluating their outputs becomes increasingly difficult (Minaee et al., 2024). For many coding tasks, the quality often depends on task-specific trade-offs beyond functional correctness that is necessary but not sufficient (Miah & Zhu, 2024). Thus, code evaluation is often a preference problem rather than a single correctness check: given the same instruction and two candidate solutions, the goal is to determine which response better satisfies human expectations (Jiang et al., 2025).

Since human evaluation is expensive, recent work has turned to LLMs as automatic judges (Zhu et al., 2025; Kim et al., 2024a; Gu et al., 2024; Li et al., 2024). However, monolithic LLM judges remain brittle for code preference prediction. A single-step judge must implicitly decide which task-specific criteria matter, compare the two candidate responses under those criteria, and aggregate the evidence into an overall preference (Wang et al., 2025b). Because this reasoning process is hidden, the judge can over-emphasize superficial cues such as verbosity or formatting, under-use execution or visual evidence, or apply a generic notion of quality that does not match the specific coding task (Shen et al., 2026; Pathak et al., 2025). Recent code-judge benchmarks further show that LLM judges can be sensitive to response ordering and evaluation format, raising concerns about their robustness in coding scenarios (Jiang et al., 2025).

A natural alternative is criterion-based, or rubric-based, judging. Instead of directly asking a model for a holistic preference, rubric-based methods generate explicit evaluation criteria and judge responses under those criteria. This decomposition improves interpretability and encourages the judge to consider multiple dimensions of quality. Recent methods such as LLM Rubrics, Chasing the Tail, and Recursive Rubric Decomposition (RRD) improve rubric generation by grounding criteria in responses, recursively decomposing broad rubrics, filtering redundant criteria, or reweighting correlated rubric signals (Zhang et al., 2025a; Shen et al., 2026). However, most of these pipelines remain

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.

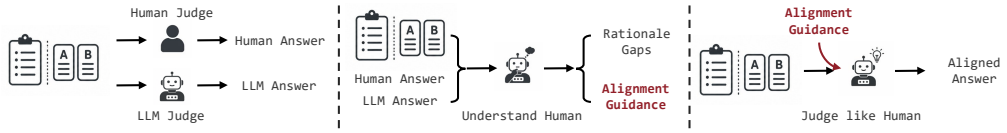

Figure 1: **Learning human preference guidance for code judging.** Given the same coding task and two candidate responses, human judges and LLM judges may produce different preference decisions. CRITERALIGN analyzes the human and LLM decisions on training cases, summarizes rationale gaps into an alignment guidance, and injects the guidance into the judge at inference time. This enables the LLM judge to better match human preferences while keeping frozen.

pointwise: each response is evaluated independently under each criterion, and the final preference is derived by comparing scores. This is poorly matched to pairwise human preference prediction, where the key evidence is often comparative (Jiang et al., 2025).

In this paper, we ask: *How can criterion-based judging be adapted to pairwise human preference prediction for code generation?* We propose CRITERALIGN, a criterion-centric judging framework that aligns criterion generation, criterion judge, and final judge with human pairwise preferences. Given a coding instruction, two candidate responses, and available execution artifacts, CRITERALIGN first generates task-specific criteria. Instead of assigning independent pointwise scores to each response, it asks a criterion judge to produce a pairwise verdict for each criterion: $v \in \{A, B, \text{tie}, \text{insufficient}\}$. Tied criteria are further refined through batched tie-driven criterion refinement, which replaces coarse non-discriminative criteria with finer comparative sub-criteria. Motivated by the response-order sensitivity observed in pairwise code judging (Jiang et al., 2025), we introduce swap-consistency criterion filtering, which removes criterion-level evidence whose verdict is not stable under swapping the candidate order.

Beyond adapting the judging pipeline to pairwise comparison, we further align it with human preferences through Human-Preference-Aligned Guidance (HPAG). Prior work has shown that natural-language principles and rubrics can serve as effective alignment or evaluation signals, guiding model behavior and fine-grained judging (Bai et al., 2022; Kim et al., 2024a; Weyssow et al., 2026). HPAG follows this direction but derives the guidance from observed gaps between human preferences and judge predictions. Specifically, as shown in Figure 1, HPAG is synthesized offline from the training split only. A guidance synthesizer compares human decisions with monolithic judge predictions and extracts recurring rationale gaps: dimensions that humans care about but the judge tends to miss, over-emphasize, or under-emphasize. The resulting guidance is distilled into compact general and category-level preference principles and injected into the criterion generator, criterion judge, and final judge. This enables reasoning with explicit human-aligned preference guidance at inference time.

We evaluate CRITERALIGN on BigCodeReward (Zhuo et al., 2025), a benchmark for pairwise code preference prediction. A monolithic Qwen2.5-VL-32B judge obtains 60.4% accuracy on the held-out validation split. Directly adapting existing pointwise criterion-generation baselines performs worse than this monolithic judge, indicating that more criteria alone do not solve the problem. Consistent with the pairwise-over-pointwise finding in CodeJudgeBench (Jiang et al., 2025), converting criterion judging from pointwise scoring to pairwise comparison already improves performance.

Our contributions are summarized as follows:

- We identify and empirically validate a mismatch between pointwise criterion-generation pipelines and pairwise code preference prediction: independently scoring each response under generated criteria can underperform a strong monolithic pairwise judge.
- We propose CRITERALIGN, a pairwise criterion-centric judging framework that combines criterion-level verdicts, batched tie-driven criterion refinement, swap-consistency criterion filtering, and HPAG distilled from human–judge preference and rationale gaps.
- We demonstrate consistent gains on BigCodeReward, where CRITERALIGN improves a Qwen2.5-VL-32B Bai et al. (2025) judge from 60.4% to 66.3%, with ablations isolating the contributions of pairwise adaptation, criterion filtering, and HPAG.

2 Related Work

LLM-as-a-judge for code evaluation. LLMs are increasingly used not only to generate code but also to evaluate generated code (Gu et al., 2024; Li et al., 2024; Wang et al., 2025b;a; Pathak et al., 2025; Liu et al., 2024a; Zhong et al., 2025; Chi et al., 2025; Weyssow et al., 2026). Early work studies LLM-based code judging for correctness, code understanding, repair, or educational grading, often in pointwise settings where a single response is scored or classified (Tong & Zhang, 2024; Zhao et al., 2025). Recent benchmarks move closer to preference-based evaluation. BigCodeArena and BigCodeReward collect pairwise human preferences for diverse coding tasks with execution or visual artifacts (Zhuo et al., 2025); WebDevJudge focuses on visually grounded web-development preferences (Li et al., 2025); CodeJudgeBench studies LLM-as-a-judge across code generation, repair, and unit-test generation in a monolithic non-rubric setting and finds that pairwise prompting outperforms scalar pointwise judging while remaining sensitive to response order (Jiang et al., 2025). These benchmarks show that code evaluation is not only about functional correctness, but also about comparing task-specific qualities that affect human preference. Our work builds on this pairwise preference and studies how to make criterion-based judges aligned with such human comparisons.

Rubric- and criterion-based judges. A common way to improve LLM judges is to make evaluation criteria explicit. Manual or calibrated rubric prompting methods use natural-language rubrics to guide fine-grained assessment (Liu et al., 2023; Kim et al., 2024a;b; Liu et al., 2024b). More recent work automates rubric construction through checklist generation, criterion decomposition, or adaptive rubric refinement (Cook et al., 2024; Rao & Callison-Burch, 2026; Hong et al., 2026; Bai et al., 2026; Siro et al., 2026). Another line uses rubrics as reward signals or post-training supervision (Gunjal et al., 2025; Rezaei et al., 2025; Liu et al., 2025a; Shen et al., 2026; Jia et al., 2026; Liu et al., 2026). These methods show that generated criteria can be improved through recursive refinement, response-conditioned instantiation, preference-contrast synthesis, or correlation-aware weighting. However, they are mainly developed for general open-ended judging or reward modeling, and most do not directly target code-specific pairwise human preference prediction. CriterAlign instead focuses on code-specific pairwise human preference prediction, using criterion-based judging as a scaffold for human-aligned comparison rather than pointwise scoring.

Guidance from human preferences. Several lines of work show that natural-language principles, rubrics, critiques, or feedback can guide model behavior and evaluation (Bai et al., 2022; Kim et al., 2024a; Weyssow et al., 2026). Constitutional-style methods encode desired behavior as natural-language principles, while critique- or rubric-conditioned evaluators use textual guidance to structure assessment (Bai et al., 2022; Ankner et al., 2024; Zhang et al., 2025b; Sun et al., 2024). Closer to our setting, preference-derived guidance can be distilled from annotations and reused as inference-time context for a frozen judge (Findeis et al., 2025), while training-time approaches bake similar signals into model weights (Liu et al., 2025b). CriterAlign follows the inference-time direction: HPAG is synthesized from training examples by comparing human preferences with monolithic judge predictions, then injected into the criterion generator, criterion judge, and final judge. Unlike general-purpose constitutions or training-time judge tuning, HPAG is designed to capture code-specific human–judge rationale gaps and to guide a pairwise criterion pipeline.

3 CriterAlign

Figure 2 illustrates the overall pipeline of CriterAlign. In this section, we first formulate pairwise human preference prediction for code generation in Sec. 3.1. We then review Recursive Rubric Decomposition (RRD) in Sec. 3.2, which motivates criterion-based judging but is originally built around pointwise rubric scoring. Finally, we present CriterAlign in Sec. 3.3. The pipeline first adapts criterion decomposition to pairwise preference prediction through pairwise criterion judging, batched tie-driven refinement, and swap-consistency filtering. It then incorporates human-preference-aligned guidance to align criterion generation, criterion judging, and final synthesis with human preference rationales.

3.1 PROBLEM SETUP

We study pairwise human preference prediction for code generation. Each instance is denoted as (x, a, b, e, y), where x is the coding instruction, a and b are two candidate code responses, e denotes optional execution evidence such as program outputs, screenshots, or error traces, and y is the human preference label. Following the BigCodeReward protocol, we map the original label space to $\mathcal{Y} = \{A, B, \text{tie}\}$ by merging ambiguous failure cases into `tie`. A judge J takes (x, a, b, e) as input and predicts $\hat{y} \in \mathcal{Y}$. The goal is to maximize agreement between $\hat{y}$ and the human preference y on held-out examples.

A monolithic judge directly predicts $\hat{y}$ in a single step. In contrast, CRITERALIGN decomposes the decision into task-specific criteria. Given an instance (x, a, b, e), it first generates a set of criteria $\mathcal{C} = \{c_i\}_{i=1}^{m}$, then obtains a criterion-level pairwise verdict for each criterion, and finally synthesizes these criterion-level judgments into an overall preference prediction. This design is motivated by the observation that human preferences over code are multi-dimensional and task-dependent: depending on the instruction, humans may emphasize visual fidelity, robustness, simplicity, maintainability, or other latent requirements.

3.2 PRELIMINARY: RECURSIVE RUBRIC DECOMPOSITION

Our method builds on the general idea of rubric-based judging, particularly Recursive Rubric Decomposition (RRD) (Shen et al., 2026). In this setting, rubrics are not ground-truth annotations; they are generated by an LLM conditioned on the evaluation instance and used as intermediate evaluation aids. RRD starts from LLM-generated rubrics, recursively decomposes coarse rubrics into finer-grained criteria, filters misaligned or redundant rubrics, and then aggregates rubric-level signals with a weighting scheme. This framework is designed for open-ended evaluation, where explicit criteria can improve coverage and interpretability compared with a single holistic judgment.

However, directly applying RRD-style pipelines to our setting is suboptimal for two reasons. First, most rubric pipelines evaluate each candidate response independently under each criterion, producing pointwise satisfaction scores. Pairwise human preference prediction is different: the target is not whether each response individually satisfies a criterion, but which response better satisfies the criterion relative to the other. Second, pairwise judging introduces its own failure modes, especially position sensitivity: a judge may prefer the first-presented response even when the candidate order is swapped. CRITERALIGN therefore adapts criterion decomposition to the pairwise setting and adds a criterion-level consistency filter specifically designed for pairwise comparison.

3.3 CRITERALIGN PIPELINE

Pairwise Criterion Judging. For each instance, the criterion generator produces a task-specific criterion set $\mathcal{C} = \{c_i\}_{i=1}^{m}$ conditioned on the instruction, the two candidate responses, and available execution evidence. Each criterion is required to be atomic, comparative, and judgeable from the provided evidence. Rather than asking whether response a and response b separately satisfy c_i, we ask the criterion judge to compare them directly under c_i:

$$v_i = J_{\text{crit}}(x, a, b, e, c_i),$$

where

$$v_i \in \{A, B, \text{tie}, \text{insufficient}\}.$$

Here, A means that response a better satisfies criterion c_i, B means that response b better satisfies it, `tie` means that the two responses are comparable under this criterion, and `insufficient` means that the available evidence is not enough to judge the criterion reliably.

This pairwise reformulation changes both criterion evaluation and final aggregation. In pointwise rubric pipelines such as RRD, each response is independently scored under each rubric, and the final preference is obtained by comparing weighted sums of rubric scores. In CRITERALIGN, each criterion instead produces direct comparative evidence about the two candidate responses, and the final judge synthesizes these criterion-level verdicts into an overall $A/B/\text{tie}$ decision. We use an LLM judge rather than a fixed weighted-sum rule because human preferences often involve non-uniform trade-offs across criteria: a large correctness gap may dominate several minor style advantages, while visual fidelity may be decisive in web-design tasks Jiang et al. (2025).

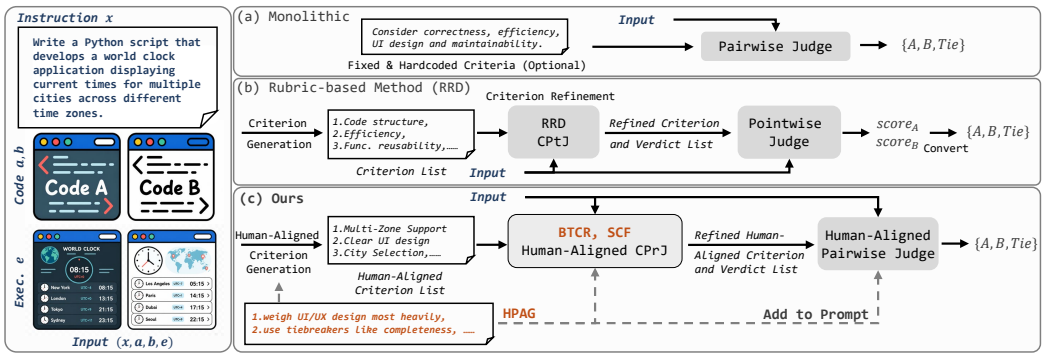

Figure 2: **Inference Pipeline Comparison.** Monolithic judges use fixed or implicit criteria, while rubric-based methods such as RRD (Shen et al., 2026) generate criteria but rely on pointwise criterion refinement and scoring. CRITERALIGN synthesizes human-preference-aligned guidance (HPAG) offline from the training split and injects it into the pairwise rubric-based pipeline for human-aligned inference. **Orange** highlights our main contributions. CPtJ and CPrJ represent criterion pointwise and pairwise judge, respectively. RRD denotes recursive rubric decomposition. BTCR and SCF denote our proposed batched tie-driven criterion refinement and swap-consistency filtering, respectively. Human-Preference-Aligned Guidance (HPAG) is synthesized offline from the training split in a separate guidance-synthesis stage; see Sec. 3.3 for details.

Batched Tie-Driven Criterion Refinement (BTCR). After initial pairwise criterion judging, some criteria receive a `tie` verdict. Rather than treating every tie as final evidence, we refine tied criteria because a tie may indicate that the criterion is too coarse to expose the concrete difference between the two responses.

Given the current criteria and their pairwise verdicts, the refiner identifies criteria judged as `tie`. For each tied parent criterion, it generates up to two finer-grained sub-criteria conditioned on the instruction, candidate responses, available execution evidence, the parent criterion, its tie rationale, and the other existing criteria. Following RRD, we filter generated sub-criteria that are redundant with or conflicting with existing criteria. Accepted sub-criteria are then re-judged by the same pairwise criterion judge and treated as independent criterion-level evidence. If different children favor different responses, we keep both verdicts as fine-grained trade-off evidence and let the final judge synthesize them into the overall preference.

This refinement is triggered by pairwise ties, unlike RRD-style decomposition based on pointwise satisfaction over multiple sampled responses. Thus, its goal is not to indefinitely expand the criterion set, but to replace non-discriminative tied evidence with more fine-grained comparative evidence. For efficiency, we batch decomposition and filtering: all tied criteria in one iteration are decomposed in a single structured call, and candidate sub-criteria are filtered with one batch redundancy check and one batch conflict check.

Swap-Consistency Criterion Filtering (SCF). Pairwise judges are known to be sensitive to candidate order. In our setting, this bias can occur not only in the final preference decision but also at the criterion level. A criterion verdict may favor response a when the input order is (a, b), but fail to favor the same response when the order is swapped to (b, a). Such position-sensitive criterion judgments are unreliable evidence for final aggregation.

We introduce *Swap-Consistency Criterion Filtering* (SCF), a pairwise-specific filtering step applied after tie-driven criterion refinement. Let $\mathcal{C}'$ denote the current criterion set after refinement, and let

$$v_i^{\rightarrow} = J_{\text{crit}}(x, a, b, e, c_i)$$

be the already obtained criterion verdict for criterion $c_i \in \mathcal{C}'$. SCF re-judges the same criterion under the swapped candidate order:

$$v_i^{\leftarrow} = J_{\text{crit}}(x, b, a, e^{\leftarrow}, c_i),$$

where $e^{\leftarrow}$ denotes the execution evidence with the artifacts of responses a and b swapped accordingly.

We then map the swapped verdict back to the original candidate order through a swap operator $\pi(\cdot)$:

$$\pi(A) = B, \quad \pi(B) = A, \quad \pi(\text{tie}) = \text{tie}, \quad \pi(\text{insuf.}) = \text{insuf.}.$$

A criterion-level judgment is kept only when

$$v_i^{\rightarrow} = \pi(v_i^{\leftarrow}).$$

Otherwise, the corresponding criterion–judgment pair is discarded before final synthesis.

SCF differs from holistic order-swap debiasing. Instead of discarding the entire pairwise example when the final answer is inconsistent, SCF operates at the criterion level. This allows the pipeline to remove unstable evidence while retaining other criteria whose pairwise verdicts are order-consistent. As a result, SCF serves both as a debiasing mechanism and as a quality filter for criterion-level evidence.

Human-Preference-Aligned Guidance (HPAG). Criterion decomposition alone does not guarantee human alignment. Even when criteria are valid and pairwise-consistent, the judge may still emphasize dimensions differently from human annotators. We address this mismatch with *Human-Preference-Aligned Guidance*.

HPAG is synthesized ***offline*** using only the ***training split*** (20% of the entire dataset). For each training example, we construct a guidance-synthesis record in three steps. First, we run a monolithic LLM judge on the input (x, a, b, e) and ask it to output both a pairwise preference prediction and a free-form rationale. Second, to make the human side explicit, we ask a rationale-analysis LLM to produce a concise hypothesis for the human-vote rationale based only on the training example and its human preference labels. Third, we package these signals into a structured record:

$$r = (x, a, b, \hat{y}_{\text{mono}}, s_{\text{mono}}, y, y_{\text{aspect}}, s_{\text{human}}),$$

where $\hat{y}_{\text{mono}}$ and s_{mono} denote the monolithic judge prediction and rationale, y is the human vote, y_{aspect} denotes aspect-level labels, and s_{human} is the inferred human-vote rationale.

The guidance synthesizer takes the concatenated list of these training records as input and summarizes recurring gaps between the monolithic judge and human preferences. These gaps describe dimensions that humans consistently care about but the judge tends to miss, over-emphasize, or under-emphasize. The output is a compact natural-language guidance artifact that is frozen after offline construction and never updated using validation examples. We present the synthesized HPAG in Appendix G.

We synthesize two types of guidance. *General guidance* summarizes dataset-level preference principles shared across tasks. *Category-level guidance* captures preference patterns for different task families, such as web design, game development, diagram creation, creative coding, scientific computing, and problem solving. At inference time, HPAG is injected into the criterion generator, criterion judge, and final judge. For the criterion generator, it encourages criteria that reflect human-relevant preference dimensions. For the criterion judge, it reminds the judge how humans tend to compare responses under each criterion. For the final judge, it guides the synthesis of criterion-level evidence toward human preference.

Final Judge. After batched tie-driven refinement and swap-consistency criterion filtering, the remaining criterion–verdict pairs form a set of reliable pairwise evidence:

$$\mathcal{E} = \{(c_i, v_i) : v_i^{\rightarrow} = \pi(v_i^{\leftarrow})\}.$$

The final judge receives the original instance, retained criterion-level evidence, and final-stage HPAG:

$$\hat{y} = J_{\text{final}}(x, a, b, e, \mathcal{E}; g_{\text{final}}), \quad \hat{y} \in \{A, B, \text{tie}\}.$$

Overall, CRITERALIGN adapts criterion-centric judging to pairwise code preference prediction through four key changes: pairwise criterion verdicts replace pointwise satisfaction scores, BTCR replaces coarse tied criteria with finer comparative evidence, SCF removes order-sensitive criterion evidence, and HPAG aligns the criterion generator, criterion judge, and final judge with human preference rationales.

Table 1: **Main results on BigCodeReward validation.** We compare CRITERALIGN with the monolithic Qwen2.5-VL-32B judge and criterion-generation-based baselines. Numbers are accuracy in percent; *imp.* denotes improvement over the monolithic baseline. **Bold** marks the best result.

Method	Acc.	*imp.*
Monolithic (Bai et al., 2025)	60.4	-
Rubric Is All You Need (Pathak et al., 2025)	48.1	−12.3
Chasing the Tail (Zhang et al., 2025a)	53.3	−7.1
LLM Rubrics Shen et al. (2026)	50.3	−10.1
RRD_{UW} Shen et al. (2026)	51.6	−8.8
RRD_{LLM} Shen et al. (2026)	55.0	−5.4
CRITERALIGN **(Ours)**	**66.3**	**+5.9**

4 EXPERIMENTS

4.1 SETUP

Dataset. We mainly evaluate our method on BigCodeReward, a benchmark for pairwise human preference prediction in code generation (Zhuo et al., 2025). Each example contains a coding instruction, two candidate solutions, available execution or visual artifacts, and human preference labels. The instructions cover diverse code-generation scenarios, including web design, game development, diagram creation, creative coding, scientific computing, and general problem solving. To avoid information leakage, we randomly split BigCodeReward into a training split and a held-out validation split. We use the training split only to synthesize the human-preference-aligned guidance, and report all main results on the held-out validation split. Following our protocol, the validation split contains $3{,}785$ examples, corresponding to approximately 80% of the data, while the remaining examples are used for guidance synthesis. Unless otherwise specified, we report overall preference accuracy on the validation split.

Implementation details Our default judge is Qwen2.5-VL-32B (Bai et al., 2025). The monolithic baseline directly asks the judge to compare the two candidate responses and predict the human-preferred answer. CRITERALIGN uses the same judge backbone but decomposes the decision process into multiple stages. Unless otherwise stated, the default human-preference-aligned guidance synthesizer is Claude Sonnet 4.6 (Anthropic, 2024), which supports a long context window that can consume the full training split for guidance synthesis. For synthesizers with shorter context windows, we downsample the training split to the largest subset that fits within the context budget, ensuring that all guidance is still synthesized only from training examples. We serve all open-weight models with vLLM (Kwon et al., 2023) on a node with $4\times$ H200 GPUs, using the same decoding settings across methods unless otherwise specified; proprietary synthesizers are accessed through their API. We provide all prompt templates and synthesized guidance in Appendix F and Appendix G, respectively.

4.2 MAIN RESULTS

Our main comparisons are against criterion-generation-based pipelines, including LLM Rubrics (Shen et al., 2026), Rubric Is All You Need (Pathak et al., 2025), Chasing the Tail (Zhang et al., 2025a), and RRD (Shen et al., 2026). We reproduced and reported their results on the BigCodeReward. Table 1 summarizes the main results. On BigCodeReward validation, the monolithic Qwen2.5-VL-32B judge obtains 60.4% accuracy. Existing criterion-generation-based baselines (Shen et al., 2026; Pathak et al., 2025; Zhang et al., 2025a; Shen et al., 2026) underperform this monolithic judge by a large margin. This result highlights an important mismatch: although criterion decomposition is attractive for open-ended evaluation, directly applying existing pointwise criterion pipelines to pairwise code preference prediction can hurt rather than help. In contrast, CRITERALIGN reaches 66.3% accuracy, improving over the monolithic baseline by $+5.9$ points and over the strongest reproduced criterion-generation baseline (Shen et al., 2026) by $+11.3$ points. This improvement indicates the effectiveness of our design choices. We show one detailed case in

Table 2: **Component ablation on BigCodeReward validation.** Starting from pointwise RRD, we progressively adapt criterion-based judging to the pairwise preference setting and add each component of CRITERALIGN. *imp.* reports the cumulative improvement over the monolithic Qwen2.5-VL-32B.

Method	Acc.	*imp.*
Qwen2.5-VL-32B (Monolithic)	60.4	—
Pointwise RRD (Shen et al., 2026)	55.0	−5.4
Pairwise RRD	61.8	+1.4
+ Batched tie-driven criterion refinement (BTCR)	62.2	+1.8
+ Global human-preference-aligned guidance (G-HPAG)	63.6	+3.2
+ Swap-consistency criterion filtering (SCF)	64.6	+4.2
+ Category-level human-preference-aligned guidance (C-HPAG)	**66.3**	**+5.9**

Table 3: **Pipeline-integrity controls on BigCodeReward validation.** Panel (a) tests whether HPAG helps outside the full criterion pipeline, whether category-level HPAG provides additional gains rather than merely exposing category labels. Panel (b) ablates where HPAG is injected in CRITERALIGN.

(a) Guidance controls

Variant	Acc.
Monolithic baseline	60.4
+ Global HPAG	63.3
+ Category-level HPAG	64.4
w/o Category-level HPAG	64.6
Empty category guidance	64.9
Full CRITERALIGN	**66.3**

(b) Per-stage HPAG ablation

Gen.	Crit. Judge	Final	Acc.
			62.2
✓			63.4
	✓		64.6
✓	✓		65.2
✓	✓	✓	**66.3**

Ours: HPAG injected into all three stages.

Appendix C. We further verify that these gains are statistically robust under paired bootstrap resampling and McNemar tests, and remain stable across stochastic decoding runs in Appendix B.1.

4.3 COMPONENT ABLATION

Table 2 decomposes the improvement of CRITERALIGN. Starting from the monolithic Qwen2.5-VL-32B baseline at 60.4%, pointwise RRD drops to 55.0%, confirming that a direct pointwise rubric pipeline is not a strong baseline for this task. The largest recovery comes from adapting criterion judging to the pairwise setting. Replacing pointwise satisfaction scoring with pairwise criterion comparison raises accuracy from 55.0% to 61.8%, a $+6.8$ point gain over pointwise RRD and a $+1.4$ point improvement over the monolithic judge. We provide a more detailed pointwise-to-pairwise adaptation analysis in Appendix A.1. This result suggests that criteria become useful when they provide comparative evidence about which response better satisfies each criterion. The remaining components further improve this pairwise criterion pipeline. Batched tie-driven criterion refinement increases accuracy from 61.8% to 62.2% by replacing coarse tied criteria with finer comparative evidence. Adding global HPAG improves accuracy to 63.6%, showing that preference-aligned guidance helps the criterion generator and criterion judge better reflect human-relevant dimensions. Swap-consistency filtering further raises accuracy to 64.6%, indicating that filtering order-sensitive criterion evidence improves robustness. Finally, category-level HPAG gives the full CRITERALIGN model at 66.3%, suggesting that category-specific preference guidance provides complementary gains beyond global guidance. Overall, the ablation shows that CRITERALIGN's improvement is cumulative rather than driven by a single prompt change: pairwise criterion judging fixes the main pointwise mismatch, BTCR refines non-discriminative ties, SCF removes unstable criterion evidence, and HPAG aligns the criterion generator, criterion judge, and final judge with human preferences.

Table 4: **Generalization across judges and guidance synthesisers.** We vary both the judge model and the model used to synthesize HPAG. *Mono.* and *Ours* denote the accuracy of the matched monolithic baseline and CRITERALIGN, respectively. *imp.* reports the improvement over the matched monolithic baseline. Numbers are accuracy in percent.

Judge	Guidance synthesiser	Mono.	Ours	*imp.*
Kimi-VL-A3B	Kimi-VL-A3B	46.5	56.9	+10.4
	Sonnet 4.6		60.7	+14.2
Gemma-4-31B	Gemma-4-31B	57.7	64.5	+6.8
	Sonnet 4.6		63.7	+6.0
Qwen2.5-VL-32B	Qwen2.5-VL-32B	60.4	65.2	+4.8
	Sonnet 4.6		66.3	+5.9

4.4 PIPELINE-INTEGRITY CONTROLS

Table 3 provides additional controls for HPAG and its injection points. Adding global HPAG to the monolithic judge improves accuracy from 60.4% to 63.3%; further adding category-level HPAG raises it to 64.4%, showing that HPAG helps even without criterion decomposition. However, the full CRITERALIGN pipeline reaches 66.3%, indicating that guidance alone does not explain the entire gain. Category-guidance controls test whether gains come from category-level guidance itself, rather than category labels or prompt slots. Removing category-level HPAG gives 64.6%, while empty category guidance gives 64.9%, suggesting that the category scaffold alone provides only a small gain. Finally, HPAG is most effective when injected throughout the pipeline. Without HPAG, the decomposed pipeline obtains 62.2%; injecting HPAG only into criterion generation, only into criterion judging, and into both stages gives 63.4%, 64.6%, and 65.2%, respectively. Adding HPAG to final synthesis reaches 66.3%, showing that best performance comes from aligning all three stages.

4.5 GENERALIZATION ACROSS JUDGES AND GUIDANCE SYNTHESIZERS

Table 4 evaluates whether the proposed guidance is tied to a specific judge or synthesizer. We vary the judge model across Kimi-VL-A3B (Team et al., 2025), Gemma-4-31B (Google DeepMind, 2026), and Qwen2.5-VL-32B, and vary the model used to synthesize the guidance. Across all tested judges, CRITERALIGN improves over the matched monolithic baseline. The relative gain is largest for the weakest judge. Kimi-VL-A3B obtains only 46.5% as a monolithic judge, but improves to 56.9% with self-synthesized guidance and to 60.7% with Sonnet-synthesized guidance. Gemma-4-31B improves from 57.7% to 64.5% with self-synthesized guidance and 63.7% with Sonnet-synthesized guidance. For Qwen2.5-VL-32B, the monolithic baseline is already stronger at 60.4%, but CRITERALIGN still improves it to 65.2% with Qwen-synthesized guidance and 66.3% with Sonnet-synthesized guidance. These results show that CRITERALIGN is not tied to Qwen2.5-VL-32B or to a single proprietary guidance synthesizer. The gains from self-synthesized guidance suggest that HPAG provides a transferable alignment signal rather than a brittle model-specific prompt.

5 CONCLUSION

We presented CRITERALIGN, a criterion-centric framework for pairwise code preference prediction. Instead of independently scoring each response under pointwise rubrics, CRITERALIGN adapts criterion decomposition to pairwise human preference prediction through criterion-level A/B/tie judgments, tie-driven refinement, swap-consistency filtering, and human-preference-aligned guidance. On BigCodeReward, CRITERALIGN improves a Qwen2.5-VL-32B monolithic judge from 60.4% to 66.3% accuracy and outperforms the strongest reproduced criterion-generation baseline by $+11.3$ points. Ablations show that the gains come from both pairwise criterion design and HPAG, while cross-model experiments demonstrate the generality of CRITERALIGN across different judge backbones and guidance synthesizers. These results indicate that criterion-based judging for code preference prediction should not only generate more criteria, but also align how those criteria are compared and synthesized with human pairwise preferences.

6 AI Assistance Disclosure

Generative AI tools were used to assist with drafting and language polishing of parts of the manuscript. All AI-assisted text was reviewed and edited by the authors to ensure accuracy, clarity, and consistency with the methods and findings of this work. The authors take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

References

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan Daniel Chang, and Prithviraj Ammanabrolu. Critique-out-loud reward models. In *Pluralistic Alignment Workshop at NeurIPS 2024*, 2024.

AI Anthropic. Claude 3.5 sonnet model card addendum. *Claude-3.5 Model Card*, 3(6), 2024.

Jingwen Bai, Wei Soon Cheong, Philippe Muller, and Brian Y Lim. iruler: Intelligible rubric-based user-defined llm evaluation for revision. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, pp. 1–28, 2026.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025.

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

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.

Wayne Chi, Valerie Chen, Anastasios Nikolas Angelopoulos, Wei-Lin Chiang, Aditya Mittal, Naman Jain, Tianjun Zhang, Ion Stoica, Chris Donahue, and Ameet Talwalkar. Copilot arena: A platform for code LLM evaluation in the wild. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 10354–10382. PMLR, 13–19 Jul 2025.

Jonathan Cook, Tim Rocktäschel, Jakob Nicolaus Foerster, Dennis Aumiller, and Alex Wang. Ticking all the boxes: Generated checklists improve llm evaluation and generation. In *Language Gamification-NeurIPS 2024 Workshop*, 2024.

Arduin Findeis, Timo Kaufmann, Eyke Hüllermeier, Samuel Albanie, and Robert D Mullins. Inverse constitutional ai: Compressing preferences into principles. In *The Thirteenth International Conference on Learning Representations*, 2025.

Google DeepMind. Gemma 4: Our most capable open models to date. Technical report, Google DeepMind, 2026. URL `https://deepmind.google/models/gemma/gemma-4/`. Technical Report.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *The Innovation*, 2024.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean M Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025.

Yihan Hong, Huaiyuan Yao, Bolin Shen, Wanpeng Xu, Hua Wei, and Yushun Dong. Rulers: Locked rubrics and evidence-anchored scoring for robust llm evaluation. *arXiv preprint arXiv:2601.08654*, 2026.

Ruipeng Jia, Yunyi Yang, Yuxin Wu, Yongbo Gai, Siyuan Tao, Mengyu Zhou, Jianhe Lin, Xiaoxi Jiang, and Guanjun Jiang. Open rubric system: Scaling reinforcement learning with pairwise adaptive rubric. *arXiv preprint arXiv:2602.14069*, 2026.

Hongchao Jiang, Yiming Chen, Yushi Cao, Hung-yi Lee, and Robby T Tan. Codejudgebench: Benchmarking llm-as-a-judge for coding tasks. *arXiv preprint arXiv:2507.10535*, 2025.

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

Jaehun Jung, Faeze Brahman, and Yejin Choi. Trust or escalate: Llm judges with provable guarantees for human agreement. In *The Thirteenth International Conference on Learning Representations*, 2025.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, et al. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*, 2024a.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4334–4353, 2024b.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Chunyang Li, Yilun Zheng, Xinting Huang, Tianqing Fang, Jiahao Xu, Lihui Chen, Yangqiu Song, and Han Hu. Webdevjudge: Evaluating (m) llms as critiques for web development quality. *arXiv preprint arXiv:2510.18560*, 2025.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*, 2024.

Dengcan Liu, Fengkai Yang, Xiaohan Wang, Shurui Yan, Jiajun Chai, Jiahao Li, Yikun Ban, Zhendong Mao, Wei Lin, and Guojun Yin. Cdrrm: Contrast-driven rubric generation for reliable and interpretable reward modeling. *arXiv preprint arXiv:2603.08035*, 2026.

Jiawei Liu, Thanh Nguyen, Mingyue Shang, Hantian Ding, Xiaopeng Li, Yu Yu, Varun Kumar, and Zijian Wang. Learning code preference via synthetic evolution. *arXiv preprint arXiv:2410.03837*, 2024a.

Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. Openrubrics: Towards scalable synthetic rubric generation for reward modeling and llm alignment. *arXiv preprint arXiv:2510.07743*, 2025a.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 2511–2522, 2023.

Yuxuan Liu, Tianchi Yang, Shaohan Huang, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. Hd-eval: Aligning large language model evaluators through hierarchical criteria decomposition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7641–7660, 2024b.

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*, 2025b.

Tanha Miah and Hong Zhu. User centric evaluation of code generation tools. In *2024 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pp. 109–119. IEEE, 2024.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.

Aditya Mittal, Ryan Shar, Zichu Wu, Shyam Agarwal, Tongshuang Wu, Chris Donahue, Ameet Talwalkar, Wayne Chi, and Valerie Chen. Comparing developer and llm biases in code evaluation. *arXiv preprint arXiv:2603.24586*, 2026.

Jiwon Moon, Yerin Hwang, Dongryeol Lee, Taegwan Kang, Yongil Kim, and Kyomin Jung. Don't judge code by its cover: Exploring biases in llm judges for code evaluation. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 1364–1389, 2026.

Aditya Pathak, Rachit Gandhi, Vaibhav Uttam, Arnav Ramamoorthy, Pratyush Ghosh, Aaryan Raj Jindal, Shreyash Verma, Aditya Mittal, Aashna Ased, Chirag Khatri, et al. Rubric is all you need: Improving llm-based code evaluation with question-specific rubrics. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V. 1*, pp. 181–195, 2025.

Delip Rao and Chris Callison-Burch. Autorubric: Unifying rubric-based llm evaluation. *arXiv preprint arXiv:2603.00077*, 2026.

MohammadHossein Rezaei, Robert Vacareanu, Zihao Wang, Clinton Wang, Bing Liu, Yunzhong He, and Afra Feyza Akyürek. Online rubrics elicitation from pairwise comparisons. *arXiv preprint arXiv:2510.07284*, 2025.

William F Shen, Xinchi Qiu, Chenxi Whitehouse, Lisa Alazraki, Shashwat Goel, Francesco Barbieri, Timon Willi, Akhil Mathur, and Ilias Leontiadis. Rethinking rubric generation for improving llm judge and reward modeling for open-ended tasks. *arXiv preprint arXiv:2602.05125*, 2026.

Clemencia Siro, Pourya Aliannejadi, and Mohammad Aliannejadi. Learning to judge: Llms designing and applying evaluation rubrics. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6371–6389, 2026.

Zhiqing Sun, Yikang Shen, Hongxin Zhang, Qinhong Zhou, Zhenfang Chen, David Daniel Cox, Yiming Yang, and Chuang Gan. Salmon: Self-alignment with instructable reward models. In *The Twelfth International Conference on Learning Representations*, 2024.

Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, et al. Kimi-vl technical report. *arXiv preprint arXiv:2504.07491*, 2025.

Weixi Tong and Tianyi Zhang. Codejudge: Evaluating code generation with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 20032–20051, 2024.

Xinchen Wang, Ruida Hu, Pengfei Gao, Chao Peng, and Cuiyun Gao. An agent-based evaluation framework for complex code generation. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 2427–2439. IEEE, 2025a.

Yutong Wang, Pengliang Ji, Chaoqun Yang, Kaixin Li, Ming Hu, Jiaoyang Li, and Guillaume Sartoretti. Mcts-judge: Test-time scaling in llm-as-a-judge for code correctness evaluation. *arXiv preprint arXiv:2502.12468*, 2025b.

Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. Self-preference bias in llm-as-a-judge. In *Neurips Safe Generative AI Workshop 2024*, 2024.

Martin Weyssow, Aton Kamanda, Xin Zhou, and Houari Sahraoui. Codeultrafeedback: An llm-as-a-judge dataset for aligning large language models to coding preferences. *ACM Transactions on Software Engineering and Methodology*, 35(3):1–36, 2026.

Junkai Zhang, Zihao Wang, Lin Gui, Swarnashree Mysore Sathyendra, Jaehwan Jeong, Victor Veitch, Wei Wang, Yunzhong He, Bing Liu, and Lifeng Jin. Chasing the tail: Effective rubric-based reward modeling for large language model post-training. *arXiv preprint arXiv:2509.21500*, 2025a.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Yuwei Zhao, Ziyang Luo, Yuchen Tian, Hongzhan Lin, Weixiang Yan, Annan Li, and Jing Ma. Codejudge-eval: Can large language models be good judges in code understanding? In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 73–95, 2025.

Ming Zhong, Xiang Zhou, Ting-Yun Chang, Qingze Wang, Nan Xu, Xiance Si, Dan Garrette, Shyam Upadhyay, Jeremiah Liu, Jiawei Han, et al. Vibe checker: Aligning code evaluation with human preference. *arXiv preprint arXiv:2510.07315*, 2025.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges. In *The Thirteenth International Conference on Learning Representations*, 2025.

Terry Yue Zhuo, Xiaolong Jin, Hange Liu, Juyong Jiang, Tianyang Liu, Chen Gong, Bhupesh Bishnoi, Vaisakhi Mishra, Marek Suppa, Noah Ziems, et al. Bigcodearena: Unveiling more reliable human preferences in code generation via execution. *arXiv preprint arXiv:2510.08697*, 2025.