Closing the Evaluation Gap: Ensembling LLM-Judges Generates More Reliable Inference-Time Reference-Free Critiques

Anonymous ACL submission

Abstract

LLM-as-a-Judge allows for efficient and scalable natural language evaluations of complex generated outputs, such as code, without the need for a ground-truth reference. These evaluation protocols have become a crucial part for inference-time refinement approaches like prompt optimization. However, an important question arises of whether a pre-trained LLM can generate a reliable evaluation of the output. In this work, we derive an interesting, insightful result showing that a single LLM-based judge is insufficient in generating an optimal critique. We then provide a solution by demonstrating that aggregating multiple LLM-generated evaluations can better model the optimal critique. We empirically show the merits of ensembling multiple LLM-judges via prompt optimization experiments for code generation. Ensembling judges leads to up to a $\sim 9\%$ increase in solved coding problems over using a single-judge. We perform ablations utilizing different aggregation methods and diverse evaluation instructions, emphasizing the non-trivial design of ensembling LLM-judges to suggest further research. We provide anonmyzied code: https://anonymous.4open.science/r/ensemble_eval-891B/ReadMe.md

1 Introduction

Large Language Models (LLMs) (Achiam et al., 2023; Touvron et al., 2023) have demonstrated ever-increasing performance in various tasks such as code generation, document summarization, and image captioning (Chen et al., 2024; Gulwani, 2010; Basyal and Sanghvi, 2023; Chen et al., 2022). With the broad applicability of LLMs in society, reliably evaluating LLM outputs during inference time is a pressing matter. For example, although LLMs in deployment can output realistic code, the code may not necessarily run as intended (Stroebl et al., 2024). Training or fine-tuning an LLM for every task has high computational cost (Zan et al., 2022; Kaplan et al., 2020). Thus, inference-time approaches, such as prompt optimization, refine the output without updating the model with an end-to-end feedback loop.

To achieve this inference-time refinement, we need a natural language evaluation, or *critique* as part of the feedback loop to direct the update direction of the next output (Cheng et al., 2023; Wang et al., 2023; Zhou et al., 2022; Yuksekgonul et al., 2024). For example, a critique such as "this code has a logical error..." catches errors and can be used as part of the prompt for updating the generated code in the next iteration. Thus, critiques are a fundamental component of iterative inference-time improvement.

However, obtaining critiques is a challenge. Human evaluators are expensive, time-consuming, and may require domain experts. Therefore, LLM-as-a-Judge (LLM-judges) has been utilized to verify the LLM output automatically (Verga et al., 2024; He et al., 2024; Kim et al., 2024). While an LLM can judge efficiently and automatically, for complex generative tasks such as code generation, they can output incorrect critiques (Stroebl et al., 2024). Many prior work assumes that the LLM-judge has reference information, such as unit test results or even a ground-truth solution (He et al., 2024; Yuksekgonul et al., 2024). This information is often unavailable (Chen et al., 2024; Nguyen et al., 2024) Therefore, we must rely on *point-wise, reference-free* critique that is generated from an LLM-judge that is only given the output. A natural question arises for this practical, yet challenging scenario: *Does a reference-free LLM judge generate an optimal critique for iterative refinement?*

In this work, we tackle this question by deriving an interesting result demonstrating that using a single reference-free LLM-based judge cannot perfectly model the unknown, oracle critique, creating a suboptimality gap in critique. This gap can cause issues where a ground-truth solution is not

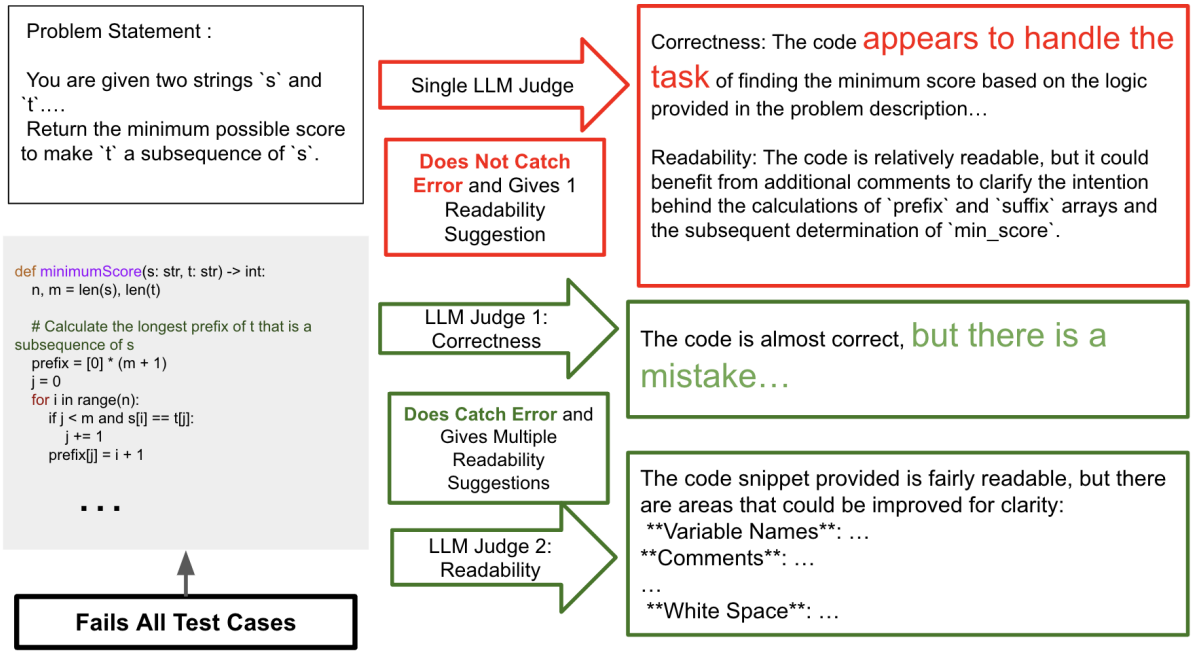

Figure 1: On the left, generated zero-shot output from GPT-4o for a LeetCodeHard problem that fails all test cases. On the right, critiques were generated from a single judge (red) and from multiple judges (green). The ensemble of judges detects errors that the single judge misses. The readability judge of the ensemble gives more suggestions for improvement while the single-judge approach gives just one.

Reference	Multiple LLM Judges	Point-Wise/ Reference-Free	Critique(s)	Theoretical Insights	Prompt Optimization Experiments
Madaan et al. (2024)	✗	✓	✓	✗	✓
Shinn et al. (2024)	✗	✗	✓	✗	✓
Verga et al. (2024)	✓	✓	✗	✗	✗
Xu et al. (2024)	✓	✗	✗	✓	✗
Kim et al. (2024)*	✓	✓	✓	✗	✗
He et al. (2024)	✓	✗	✓	✗	✓
Yuksekgonul et al. (2024)	✗	✗	✓	✗	✓
Badshah and Sajjad (2024)	✓	✗	✓	✗	✗
This Work	✓	✓	✓	✓	✓

Table 1: Summary of key elements of our work compared to recent relevant work. Our work is the first to use multiple LLM natural language evaluators for prompt optimization without reference information and provide theoretical insights. *Kim et al. (2024) does have prompt optimization experiments but does not use multiple judges for those experiments.

available, such as code generation. Incorrect critiques can lead to negative update directions for inference-time processes. To mitigate the impact of the suboptimality gap, we prove that an ensemble of reference-free LLM evaluators decreases it. We empirically validate this insight extensively by utilizing multiple LLM-judges in a prompt optimization process for code generation. We explore different aggregation methods, concatenation and summarization, and utilizing diverse LLM-judges where each one judges the output on distinct criteria. These empirical results emphasize that designing the ensemble LLM-judge protocol is non-trivial and warrants further research for more reliable reference-free critique.

While multiple LLM judges have been analyzed in other work (cf. Table 1), to the best of our knowledge, we are the first to include both a theoretical motivation for multiple reference-free LLMs to generating critiques and an empirical validation of the theory via prompt optimization experiments. Prior work that introduces multiple LLM judges such as Verga et al. (2024) do not generate critiques to be used for inference-time improvement methods. Here, we show the direct benefits of ensemble judges that generate critiques as part of a LLM feedback loop.

Our contributions are summarized as follows:

- **Theoretical Motivation for Ensemble LLM-Judges:** We propose a novel formulation for the prompt optimization task that specifically highlights the suboptimality gap in critiques for a single LLM-judge. With this formulation, we prove that increasing the number of LLM judges reduces this gap with a linear additivity assumption.
- **Empirical Performance Over Single-Evaluation Approach:** We thoroughly test this method via prompt optimization pipeline for code generation on three benchmarks. We show up to $\sim 9\%$ in coding problems solved over a single-judge approach.
- **Extensive Study of Evaluation Design:** We provide multiple studies that demonstrate that the choice of aggregation method, the number of judges, and the combination of criteria can significantly affect performance, emphasizing that the design of an ensemble of LLM-judges is non-trivial.

2 Related Works

LLM-as-a-Judge. LLM-as-a-Judge (Zheng et al., 2023; Li et al., 2024), has been growing in interest due to the ability of LLMs to evaluate large outputs like text (Sellam et al., 2020; Kocmi and Federmann, 2023) quickly and to align with human preferences. Prior work has also studied finetuning LLMs to be judges (Zhou et al., 2022; Xiong et al., 2024). Ankner et al. (2024) used LLM-generated critiques to augment the scalar reward from a reward model. Li et al. (2023) used discussion between multiple LLMs to select a strong LLM-judge for question-answering. Strong LLM judges have been shown to generalize across tasks (Huang et al., 2024). Weak LLM evaluators have been used to judge the debate between two stronger LLMs (Kenton et al., 2024).

Ensemble LLM-Judges. Verga et al. (2024) showed a panel of smaller LLM judges can provide numeric scores correlating to human judgment than a single larger LLM model can. Similarly, (Kim et al., 2024) has shown repeated sampling of evaluations can also correlate to human judgement better. (Badshah and Sajjad, 2024) used multiple reference-guided LLM judges for question-answering. Other work has used multiple LLM-judges for iterative fine-tuning (Xu et al., 2024; Agrawal et al., 2024). While for prompt optimization, we theoretically characterize that increasing evaluators reduces a evaluation suboptimality gap and provide results with various aggregation methods. Concurrent work, CRISPO by He et al. (2024), also looks at multiple evaluators for prompt optimization for tasks with multiple criteria like we do. However, they rely on access to reference information to generate a critique. Here, we remove that access that may not be available and only input the AI output to the evaluators.

Natural Language Critiques and Prompt Optimization. Many prior works have studied using critiques for prompt optimization. Madaan et al. (2024) was one of the first works to propose a prompt iterative feedback loop for refining LLMs, and Pryzant et al. (2023) established prompt gradients, or Textual Gradients, as feedback to an AI system. Concurrent work, CRISPO by He et al. (2024), also looks at multiple evaluators for prompt optimization for tasks with multiple criteria. Prompt reinforcement learning with natural language critiques has also been used to improve LLM-based systems (Shinn et al., 2024; Feng et al., 2024). Due to the abstract nature of raw text, theoreicall

3 Problem Formulation

Reference-Free LLM for Generating Critiques. Let y be the output for a given task and y^* be the optimal response. For code generation, y^* would be a functionally correct, readable, and efficient solution code snippet to the problem and y would be a snippet attempting to solve the problem. We would want y to match y^* as closely as possible. Mathematically, we can write the optimization problem,

$$\arg\min_{y} l(y^*, y), \tag{1}$$

where l is an objective function to capture the closeness of sampled response y to the ground truth y^*. l is akin to a loss function in machine learning. $l(y^*, y)$ is a natural language critique c of y comparing it to y^*. We use the terms "critique" and "loss" interchangeably as prior literature has established the analogy (Yuksekgonul et al., 2024).

Limitations and Challenges in Critiques. In an ideal setting, if we had access to y^* as a ground-truth label for a supervised loss (Tiwari, 2022), then we can achieve the optimal performance. However, in practice, they are hard to obtain or simply unknown for many tasks such as code generation (Chen et al., 2024). Therefore, a direct comparison to an optimal output y^* and the resulting calculation of c are both infeasible. Current SoTA work

instead *sample* an evaluation c from an evaluation LLM policy conditioned by the response output y and prompt x as $c \sim \pi(\cdot|x, y)$. Let us denote $\pi_c = \pi(\cdot|x, y)$ for notation simplicity. When parameterizing π_c with an LLM, it is known as the LLM-judge. Ideally, we would like the evaluation c of y to be $l(y^*, y)$. More specifically, let us assume the existence of an optimal LLM-judge denoted by π_c^*, sampling from which will give us samples of the true loss function $l(y^*, y)$. However, LLM-judges tend to fail with no reference (Stureborg et al., 2024). Figure 1 demonstrates this with an example LLM-judge letting an error go undetected.

A Single Reference-Free LLM-Judge Outputs Suboptimal Critques. As π_c^* is unavailable as discussed before, current SoTA methods sample the critique loss from a single evaluator as $c \sim \pi_c$. Now, we know that in the majority of the scenarios, π_e will not be the true evaluator policy π_c^*. We now define the suboptimality between π_c and π_c^*.

Definition 1. *Let $c = l(\hat{y}, y)$, where $\hat{y}$ is an implicit approximation of y^* from π_c. Under this scenario, we define the suboptimality gap in the critique of prior SoTA as,*

$$\begin{aligned}\Delta^{\pi}{}_{\textit{Cri-sub-opt}} &= \mathbb{E}_{c^*\sim\pi^*(\cdot|x,y)}\left[c^*\right] - \mathbb{E}_{c\sim\pi(\cdot|x,y)}\left[c\right] \\ &\leq |c_{max}| d_{TV}(\pi_c^*(\cdot|x, y), \pi(\cdot|x, y)). \quad (2)\end{aligned}$$

In this definition, we first expand upon the sub-optimality in the critique and then upper-bound using the total variation distance (Sripe-rumbudur et al., 2009). We see that the term $d_{\text{TV}}(\pi_c^*(\cdot|x, y), \pi(\cdot|x, y))$ is fixed and it cannot be improved once we have the evaluator π. This result shows the hardness of a single evaluator reaching π_c^* due to this constant gap and it will only reduce if our current LLM evaluator is near-optimal which is not true in the majority of the scenarios.

An Ensemble of Reference-free LLM-Judges Better Models the Optimal Critique. Our key idea is to utilize multiple critiques. The thought that multiple LLM-judges would work better than one sounds intuitive but a naive introduction of multiple evaluators does not work in practice.

We start our theoretical justification by defining the sub-optimality metric to measure the critique performance between π_c^* and Π as

Definition 2. *Let $\Pi = \{\pi_k(\cdot|x, y)\}_{k=1}^K$ be the set of diverse judges for x, y. We then define the sub-optimality metric, $\Delta^{\Pi}_{\textit{Cri-sub-opt}}$, as*

$$\begin{aligned}\Delta^{\Pi}{}_{\textit{Cri-sub-opt}} &= \mathbb{E}_{c\sim\pi_c^*(\cdot|x,y)}\left[c\right] \\ &\quad - \mathbb{E}_{\{c_k\sim\pi_k(\cdot|x,y)\}_{k=1}^K}\left[g(c_1, \cdots, c_K)\right]. \quad (3)\end{aligned}$$

$\Delta^{\Pi}_{\text{Cri-sub-opt}}$ is difference between the expected value of the critique under the optimal unknown critique distribution, and the expected function g which maps the K different critiques to one. In practice, g can be seen as an aggregation function such as concatenation. For the following theorem, we provide the following assumption.

Assumption 1. *g is a linear function.*

If we had access to the optimal evaluator π_c^*, we would have been able to get the ground-truth critique $c^* = l(y^*, y)$ to perform the prompt optimization. However, in place of that, we have a set of evaluators $\Pi = (\pi_1, \pi_2 \cdots \pi_K)$ and $g(c_1, c_2 \cdots c_K)$ is the aggregation function to combine the critiques. We now present the following theorem to relate the number of critiques to $\Delta^{\Pi}_{\text{Cri-sub-opt}}$.

Theorem 1. *Let d_{TV} denote the total variation distance between two distributions and let $\sum_{k=1}^K \alpha_k = 1$. Assuming all pairs $\pi_1, \pi_2 \in \Pi$ are independent of one another,*

$$\Delta^{\Pi}_{\textit{Cri-sub-opt}} \leq |c|_{max} d_{TV}(\pi_c^*, \sum_{k=1}^K \alpha_k \pi_k). \quad (4)$$

Proof. First, we characterize the sub-optimality of our proposed critique method as $\Delta = \mathbb{E}_{c^*\sim\pi_c^*}\left[c^*\right] - \mathbb{E}_{c_1\sim\pi_1(\cdot|x,y), c_2\sim\pi_2(\cdot|x,y)\cdots\pi_K}\left[g(c_1, c_2, c_3 \cdots c_K)\right]$. Note that if Δ is zero, we have the optimal critique. Thus, we want Δ to be as low as possible. For notation simplicity of the expression, we will keep to two evaluators without loss of generality. We provide a version with K evaluators in Appendix A.1.

$$\begin{aligned}\Delta &= \mathbb{E}_{c^*\sim\pi_c^*}\left[c^*\right] \\ &\quad - \mathbb{E}_{c_1\sim\pi_1(\cdot|x,y), c_2\sim\pi_2(\cdot|x,y)}\left[g(c_1, c_2)\right] \\ &= \underbrace{\mathbb{E}_{c^*\sim\pi_c^*}\left[c^*\right] - \mathbb{E}_{c\sim\pi_d(\cdot|x,y)}\left[c\right]}_{\Delta_1} + \\ &\quad \underbrace{\mathbb{E}_{c\sim\pi_d(\cdot|x,y)}\left[c\right] - \mathbb{E}_{c_1\sim\pi_1, c_2\sim\pi_2}\left[g(c_1, c_2)\right]}_{\Delta_2},\end{aligned}$$

where we add and subtract the terms $\mathbb{E}_{c\sim\pi_d(\cdot|x,y)}$, with $\pi_d = \alpha\pi_1 + (1-\alpha)\pi_2$ $(0 < \alpha < 1)$ and then separate the two terms as Δ_1, Δ_2. We next individually analyze the terms Δ_1, Δ_2.
We can now bound Δ_1 as,

$$\begin{aligned}\Delta_1 &= \mathbb{E}_{c^*\sim\pi_c^*}[c^*] - \mathbb{E}_{c\sim\pi_d(\cdot|x,y)}[c] \\ &\leq |c^*| d_{\text{TV}}(\pi^*, \pi_d) \\ &= |c^*| d_{\text{TV}}(\pi^*, \alpha\pi_1 + (1-\alpha)\pi_2),\end{aligned}$$

where we use the property of integral probability metric to bound Δ_1 as the total variation distance between the optimal critique policy and the mixture critique policy. Next, we proceed to Δ_2,

$$\begin{aligned}\Delta_2 =& \mathbb{E}_{c\sim\pi_d(\cdot|x,y)}[c] - \\ & \mathbb{E}_{c_1\sim\pi_1(\cdot|x,y), c_2\sim\pi_2(\cdot|x,y)}[g(c_1, c_2)] \\ =& \mathbb{E}_{c\sim\pi_d(\cdot|x,y)}[c] - \\ & \mathbb{E}_{c_1\sim\pi_1(\cdot|x,y), c_2\sim\pi_2(\cdot|x,y)}[\alpha c_1 + (1-\alpha)c_2] \\ =& \mathbb{E}_{c^*\sim\pi_d(\cdot|x,y)}[c^*] - \\ & \alpha\mathbb{E}_{c_1\sim\pi_1(\cdot|x,y)}[c_1] - (1-\alpha)\mathbb{E}_{c_2\sim\pi_2(\cdot|x,y)}[c_2] \\ =& 0, \end{aligned} \tag{5}$$

where we expand upon the definition of Δ_2 and use Assumption 1 on the aggregation function. Under this assumption, the two terms cancel out with the final result $\Delta_2 = 0$. Combining both terms concluded the proof. This bound indicates that the sub-optimality in critique can be expressed as the total variation distance between the optimal evaluator and the available mixture of evaluators. We know from Blei et al. (2003); Nguyen et al. (2016) that as we increase the number of mixture components and diversity amongst the components increase, it can approximate any distribution under certain assumptions.

Our theoretical finding is for a one-shot critique generation. In the following section, we will discuss how to introduce them into the iterative prompt optimization pipeline. Our idea is that at any iteration, aggregating multiple critiques will better model the unknown, optimal critique for the current output, thus leading to faster improvement than using a single LLM-judge.

3.1 Prompt Optimization with Ensemble LLM-Judges

Optimal Prompt Search. Let $\pi(\cdot|x)$ be the LLM system parameterized by fixed LLM policy that samples an output response $y \sim \pi(\cdot|x)$ given an input prompt $x \in \mathcal{X}$ from the set of prompts $\mathcal{X}$. We aim to sample a $y \sim \pi(\cdot|x^*)$ by finding an input prompt x^* corresponding to x prompt such that y is closer to the optimal response y^*. For code generation, π_θ would be the LLM generator; x would be the input prompt; y is the generated code; and the y^* here would be a code snippet that is a functionally correct, readable, and efficient solution to the problem. Mathematically, we can write,

$$x^* = \arg\min_{x\in\mathcal{X}} \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[l(y^*, y)]. \tag{6}$$

Iterative prompt optimization. Given an initial prompt x_1, we perform an iterative prompt optimization method to find x^* as follows. For each iteration $t = 1$ to T, we start by (i) sampling $y_t \sim \pi_\theta(\cdot|x_t)$, (ii) evaluate the response y_t to obtain critique $c_t = l(y^*, y_t)$, and then finally (iii) generate the next prompt $x_{t+1} \sim \pi(\cdot|y_t, c_t, x_t)$. Recent work by Yuksekgonul et al. (2024) decompose step (iii) into two separate steps and (iii.a) first generate the feedback $f_t \sim \pi(\cdot|y_t, c_t, x_t)$, and then (iii.b) generate the next prompt $x_{t+1} \sim \pi(\cdot|y_t, f_t, x_t)$. For simplicity, we use the same variable π for all LLM policies because the outputs are dependent on the input variables the policy is conditioned on, so the same LLM model can be utilized.

The success of this method is heavily dependent on step (ii), obtaining the LLM-generated critiques. A suboptimal critique can hinder the optimization process. We now show that in the reference-free case, using an ensemble of LLM-judges will provide faster prompt optimization and improved LLM system output.

4 Experiments and Results

Code Generation Experiments: We test the merits of our ensemble judge approach via the code generation task because of its practicalness and its multiple plausible criteria (e.g., correctness, efficiency). Here, the LLM generator is given a code prompt and must produce a code snippet that passes the unit tests for that prompt. This code generation task is a form of instance optimization (Yuksekgonul et al., 2024), whereby the optimization variable, the input prompt, is defined as $x_{t+1} := (y_t, f_t)$. y_0, f_0 are empty strings. We provide empirical results showing that prompt optimization with an ensemble of judges achieves higher success in test cases than single-judge-based optimization. Experiments were run on an Apple M1 Pro and macOS 14.5.

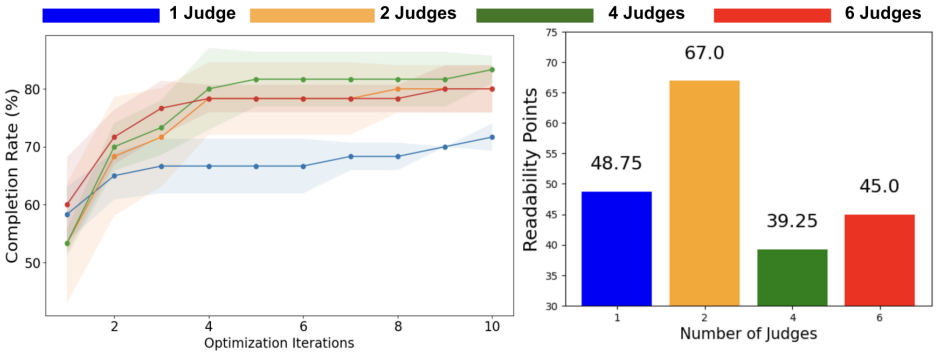

Figure 2: Completion Rate (CR) over 10 iterations Readability Points (RP) for code generation on LeetCodeHard. Over 10 iterations for each coding problem, increasing the number of judges significantly increases the functional correctness, and having 2 judges greatly increases the RP. The line plot shows the average over the 3 trials with a 95% confidence interval. However, for readability, continual increase does not continuously improve RP. This shows empirically that increasing judges does not monotonically improve all aspects of task.

Judge Method	Agg	CR (%)	RP
Self-Refine (Baseline)	–	70.0 ± 4.08	52.3
Vanilla Feedback Loop (Baseline)	–	65.0 ± 14.72	52.8
1 Judge (Baseline)	–	71.67 ± 2.36	52.6
6 Judges – All Criteria	C	$\mathbf{80.0 \pm 4.08}$	54.4
6 Judges – All Criteria	Sum	71.67 ± 8.5	**57.7**
6 Judges – All Criteria	Sel	78.33 ± 6.24	44.8
6 Judges – One Criterion Each	C	78.33 ± 2.36	37.0
6 Judges – One Criterion Each	Sum	76.67 ± 6.24	29.2
6 Judges – One Criterion Each	Sel	75.0 ± 4.08	19.2

Table 2: The Completion Rate (CR) and Readability Points (RP) over LeetCodeHard comparing various ensemble evaluation methods against inference-time improvement baselines. Ensemble methods consistently outperform baselines in terms of CR and the two highest-ranking methods in terms of readability are ensemble. The difference in CR and RP between ensemble methods emphasizes the non-trivial nature of designing the ensemble evaluation protocol.

Implementation Details: We use TextGrad from (Yuksekgonul et al., 2024) to implement the prompt optimization pipeline. We chose TextGrad because it separates the critique and feedback into two separate LLM calls, making it better to analyze the critique module in isolation. In TextGrad, the system prompt that generates the initial code, p_{init}, is different from the system prompt that updates the code in the following refinement iterations p_{update}. At $t = 0$, p_{init} specifies to the LLM that it is a code generator while the p_{update} from $1 \leq t \leq T$ specifies that it generates a new version y_{t+1} given the current code y_t and the feedback f_t. The transition from p_{init} to p_{update} is explicitly programmed and not caused by the optimization process.

LLM Setup Details: We use GPT-4o for all LLM calls. In the Appendix, we provide additional experiments and ablations. Across all trials for both methods, we use the same initial generated code for a given problem so both critique protocols can judge the same code in the initial iteration. We share the critique system prompt for both methods in Appendix A.2. Because we want a diversity of critiques, we set the temperature of all LLM-judge call to be 1. We ablate on the judge call temperature in the Appendix. All other LLM calls in the Textgrad pipeline with call temperature set to 0 similar to Yuksekgonul et al. (2024). For all experiments, the top_p $= 0.99$.

Criteria for Critiquing Code. The set of critique criteria we used for this task are as follows: syntax errors, logic errors, correctness, readability, runtime, and code redundancy. The following results are based on utilizing all these roles. We

Criteria	Judge Method	Agg	CR (%)
Correctness, Logic, Readability, Redundancy, Runtime, Syntax	Single (Baseline)	–	66.67 ± 4.71
	Ensemble - All Criteria	C	**78.33 ± 2.36**
	Ensemble - One Criterion Each	C	71.67 ± 8.5
	Ensemble - All Criteria	Sum	66.67 ± 8.5
	Ensemble - One Criterion Each	Sum	75.0 ± 4.08
	Ensemble - All Criteria	Sel	75.0 ± 8.16
	Ensemble - One Criterion Each	Sel	73.33 ± 4.71
Readability, Redundancy, Runtime	Single (Baseline)	–	66.67 ± 2.36
	Ensemble - All Criteria	C	71.67 ± 4.71
	Ensemble - One Criterion Each	C	**76.67 ± 4.71**
	Ensemble - All Criteria	Sum	73.33 ± 6.24
	Ensemble - One Criterion Each	Sum	70.00 ± 7.07
	Ensemble - All Criteria	Sel	70.00 ± 4.08
	Ensemble - One Criterion Each	Sel	71.67 ± 6.24
Correctness, Logic, Syntax	Single (Baseline)	–	**78.33 ± 6.24**
	Ensemble - All Criteria	C	73.33 ± 2.36
	Ensemble - One Criterion Each	C	76.67 ± 2.36
	Ensemble - All Criteria	Sum	**78.33 ± 2.36**
	Ensemble - One Criterion Each	Sum	75.00 ± 7.07
	Ensemble - All Criteria	Sel	**78.33 ± 6.24**
	Ensemble - One Criterion Each	Sel	75.00 ± 4.08
Logic, Readability	Single (Baseline)	–	76.67 ± 4.71
	Ensemble - All Criteria	C	72.5 ± 7.50
	Ensemble - One Criterion Each	C	70.0 ± 7.07
	Ensemble - All Criteria	Sum	75.00 ± 10.80
	Ensemble - One Criterion Each	Sum	75.00 ± 7.07
	Ensemble - All Criteria	Sel	**80.0 ± 4.08**
	Ensemble - One Criterion Each	Sel	70.00 ± 7.07

Table 3: **Utilzing Different Roles Affects CR:** This table summarizes the CR and RP for the various evaluation methods given different combinations of roles. We report the mean and standard deviation of 3 trials for CR. We use 10 problem of LeetCodeHard for 4 iterations each.

chose three roles that correlate to maximizing the number of passed test cases: correctness, logic, and syntax. We specifically chose these three to incorporate an overall correctness role with two more specific roles.

Ensemble Design: One decision in design is to give the separate LLM calls different criteria to judge the output. In **all criteria**, we specify to each LLM judge call that it should generate a critique of the output based on all the criteria. Effectively, we are doing repeated sampling of the LLM-judge. In **one criterion each**, we give each judge a single criterion to focus its judgment. Once we have generated the critiques from all the judge LLM calls, we aggregate them. We experiment with three different aggregation methods. 1) String **concatenation (C)**: a form of addition for string objects that maintains the semantic meaning of the individual critiques; we chose concatenation to model a linear function for g with uniform weights α. 2) **Summarization (Sum)**: another LLM to take in the critiques to give a final response; summarization is analogous to applying a non-linear g aggregation method to the critiques. 3) **Selection (Sel)**: an LLM selects the one critique that it believes will help improve the output the most, modeling a max operator on the critiques.

Baselines: For baselines other than a single-judge approach, we chose **Self-Refine** (Madaan et al., 2024) where the LLM code generator iteratively reflects and updates on its own output. We implement this by having a consistent system prompt throughout all the LLM calls and only changing the user prompts. We also compared with a **vanilla feedback loop**, where there is a separate feedback LLM call but there is no LLM call for explicitly critique generation. Please see the Appendix for more details on the prompts.

Metrics for Code: For correctness, we report the **Completion Rate (CR)**, the percentage of coding problems with all test cases passed (Yuksekgonul et al., 2024). Since we are focused on the effect of the evaluation protocol, we report the best-performing code generated in the optimization process after the initial zero-shot generation. Specifically, if a generated snippet at any iteration after the initial generation passes all test cases, that problem is considered completed. For readability, we take the code snippet of the last iteration of each method we are comparing and ask a panel of

LLM-judges, GPT-4o, GPT-4o-mini, GPT-4-turbo, and GPT-4, to rank their readability of the code snippets. We then calculate the Borda Count for each method. The Borda Count for a method is the number of methods that rank below it. For example, a method that is the highest ranked out of four methods gets 3 points. For each method, we sum all of the Borda Counts across all problems. To normalize across experiments that have varying sets of methods, we divide the total Borda Count by the number of methods. We call this value the **Readability Points (RP)**.

Dataset. We use the LeetCodeHard (Shinn et al., 2024) dataset containing a set of coding problem prompts and multiple unit tests for each problem to evaluate the generated code. We use 20 LeetCodeHard (Shinn et al., 2024) dataset problems with an average of $2-3$ unit tests per problem. We withhold giving any of the evaluators of either method any information on unit tests to simulate the scenario where unit tests may be unavailable to help judge (Chen et al., 2024). Please see the Appendix where we provide additional results on Humaneval (Chen et al., 2021) and EvoEval benchmarks (Xia et al., 2024).

How does increasing judges help empirically? We plot the performance over 3 trials on LeetCodeHard in Figure 2. In this experiment, we give the judge LLMs for all approaches all the criteria to critique the output and we use concatenation to aggregate. For functional correctness, ensembling judges achieve higher CR rates than a single judge. Furthermore, while using a single judge achieves similar RP to using 6 judges, it has significantly less than RP using 2 evaluators. These results empirically show that increasing judges can improve code in both aspects but not necessarily monotonically.

How Do Design Choices for Ensemble Impact Performance? In Table 2 we report the mean and standard deviation for CR. We see that the lowest-performing ensemble method, 6 judges with all criteria with summarization, achieves the same mean CR as the highest-performing baseline, single-judge. Showing the superiority of ensembling over baselines for correctness. There is a 9% difference in mean CR between ensemble methods. For readability, the two highest-ranked methods are ensemble methods. All other ensemble methods fall below the baselines in the rankings. This difference in CR and RP between ensemble methods highlights the importance of design.

Criteria Order	CR (%)
Correctness, Logic, Readability, Redundancy, Runtime, Syntax	71.67 ± 8.50
Redundancy, Logic, Correctness, Runtime, Readability, Syntax	73.33 ± 2.36
Syntax, Runtime, Redundancy, Readability, Logic, Correctness	$\mathbf{75.00 \pm 7.07}$

Table 4: **Impact of Criteria Order** CR from 10 problems of LeetCodeHard for 4 iterations. We used LLM judges with separate criteria with concatenation. Criteria Order specifies the order the LLM judges, affecting the resulting concatenated critique string.

Combination and Order of Evaluation Criteria Affects Optimization Performance. In Table 6, we analyze the effect the different combinations of evaluation criteria have on the CR over LeetCodeHard. Similar studies have been performed with finetuning using diverse reward models (Rame et al., 2024). Surprisingly and counter-intuitively, we see some methods increase in correctness when the three criteria for correctness are removed. We do see an overall increase in methods when the non-correctness criterion are removed, suggesting the LLM judges can better focus on analyzing the functionality of the code. Because concatenating strings is not commutative like adding scalar numbers. in Table 4, we provide an ablation where we change the order of the criteria in the judge system prompt. In this experiment, we used judges with one criterion each. Thus, changing the order of the call changes the concatenated string. We do not see a significant change in performance between the orderings suggesting that using ensemble judges is unaffected by the order of critiques.

5 Conclusion

In this work, we tackle reference-free LLM-judges for generating natural language critiques. Our key insight was that aggregating multiple generated critiques reduces the suboptimality gap in evaluations for a given output. We theoretically motivate ensemble LLM-judges and empirically validate the paradigm with extensive prompt optimization experiments in code generation. We also provide ablations such as on the diversity of roles, role combinations, and evaluation temperature, consistently demonstrating the need for multiple evaluators.

Limitations and Further Work

We only empirically study our approach in code generation. Further work could extend this evaluation approach to other tasks that require multiple criteria like molecule optimization or text generation. In terms of system complexity, we only study multiple evaluators for AI systems comprising a single LLM-based agent, and using a compound system with multiple elements such as a web search agent (Agentic AI system) could be interesting. Another aspect of the work that can be explored further is weighting the different LLM-based evaluations. We gave uniform weighting via concatenation. However, further work could try and adaptively change the weighting as the output progresses, representing the need to change the focus of evaluation over time. Another research direction involves removing the linearity assumption on g.

Acknowledgements

ChatGPT (4o) was used to help with coding experiments.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Aakriti Agrawal, Mucong Ding, Zora Che, Chenghao Deng, Anirudh Satheesh, John Langford, and Furong Huang. 2024. Ensemw2s: Can an ensemble of llms be leveraged to obtain a stronger llm? *arXiv preprint arXiv:2410.04571*.

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D Chang, and Prithviraj Ammanabrolu. 2024. Critique-out-loud reward models. *arXiv preprint arXiv:2408.11791*.

Sher Badshah and Hassan Sajjad. 2024. Reference-guided verdict: Llms-as-judges in automatic evaluation of free-form text. *arXiv preprint arXiv:2408.09235*.

Lochan Basyal and Mihir Sanghvi. 2023. Text summarization using large language models: a comparative study of mpt-7b-instruct, falcon-7b-instruct, and openai chat-gpt models. *arXiv preprint arXiv:2310.10449*.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Jun Chen, Han Guo, Kai Yi, Boyang Li, and Mohamed Elhoseiny. 2022. Visualgpt: Data-efficient adaptation of pretrained language models for image captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18030–18040.

Liguo Chen, Qi Guo, Hongrui Jia, Zhengran Zeng, Xin Wang, Yijiang Xu, Jian Wu, Yidong Wang, Qing Gao, Jindong Wang, et al. 2024. A survey on evaluating large language models in code generation tasks. *arXiv preprint arXiv:2408.16498*.

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. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. 2023. Black-Box Prompt Optimization: Aligning Large Language Models without Model Training. *arXiv e-prints*, arXiv:2311.04155.

Xidong Feng, Ziyu Wan, Mengyue Yang, Ziyan Wang, Girish A Koushik, Yali Du, Ying Wen, and Jun Wang. 2024. Natural language reinforcement learning. *CoRR*.

Sumit Gulwani. 2010. Dimensions in program synthesis. In *Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming*, pages 13–24.

Han He, Qianchu Liu, Lei Xu, Chaitanya Shivade, Yi Zhang, Sundararajan Srinivasan, and Katrin Kirchhoff. 2024. Crispo: Multi-aspect critique-suggestion-guided automatic prompt optimization for text generation. *arXiv preprint arXiv:2410.02748*.

Hui Huang, Yingqi Qu, Jing Liu, Muyun Yang, and Tiejun Zhao. 2024. An empirical study of llm-as-a-judge for llm evaluation: Fine-tuned judge models are task-specific classifiers. *arXiv preprint arXiv:2403.02839*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Zachary Kenton, Noah Y Siegel, János Kramár, Jonah Brown-Cohen, Samuel Albanie, Jannis Bulian, Rishabh Agarwal, David Lindner, Yunhao Tang, Noah D Goodman, et al. 2024. On scalable oversight with weak llms judging strong llms. *arXiv preprint arXiv:2407.04622*.

Seungone Kim, Juyoung Suk, Ji Yong Cho, Shayne Longpre, Chaeeun Kim, Dongkeun Yoon, Guijin Son, Yejin Cho, Sheikh Shafayat, Jinheon Baek, et al. 2024. The biggen bench: A principled benchmark for fine-grained evaluation of language models with language models. *arXiv preprint arXiv:2406.05761*.

Tom Kocmi and Christian Federmann. 2023. Large language models are state-of-the-art evaluators of translation quality. In *Proceedings of the 24th Annual Conference of the European Association for Machine Translation*, pages 193–203, Tampere, Finland. European Association for Machine Translation.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. 2024. From generation to judgment: Opportunities and challenges of llm-as-a-judge. *arXiv preprint arXiv:2411.16594*.

Ruosen Li, Teerth Patel, and Xinya Du. 2023. Prd: Peer rank and discussion improve large language model based evaluations. *arXiv preprint arXiv:2307.02762*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Hien D Nguyen, Luke R Lloyd-Jones, and Geoffrey J McLachlan. 2016. A universal approximation theorem for mixture-of-experts models. *Neural computation*, 28(12):2585–2593.

Huyen Nguyen, Haihua Chen, Lavanya Pobbathi, and Junhua Ding. 2024. A comparative study of quality evaluation methods for text summarization. *arXiv preprint arXiv:2407.00747*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. 2024. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *Advances in Neural Information Processing Systems*, 36.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Bharath K Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernhard Schölkopf, and Gert RG Lanckriet. 2009. On integral probability metrics,\phi-divergences and binary classification. *arXiv preprint arXiv:0901.2698*.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. 2024. Inference scaling Flaws: The limits of llm resampling with imperfect verifiers. *Preprint*, arXiv:2411.17501.

Rickard Stureborg, Dimitris Alikaniotis, and Yoshi Suhara. 2024. Large language models are inconsistent and biased evaluators. *arXiv preprint arXiv:2405.01724*.

Ashish Tiwari. 2022. Chapter 2 - supervised learning: From theory to applications. In Rajiv Pandey, Sunil Kumar Khatri, Neeraj kumar Singh, and Parul Verma, editors, *Artificial Intelligence and Machine Learning for EDGE Computing*, pages 23–32. Academic Press.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. Replacing judges with juries: Evaluating llm generations with a panel of diverse models. *Preprint*, arXiv:2404.18796.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. 2023. PromptAgent: Strategic Planning with Language Models Enables Expert-level Prompt Optimization. *arXiv e-prints*, arXiv:2310.16427.

Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. 2024. Top leaderboard ranking = top coding proficiency, always? evoeval: Evolving coding benchmarks via llm. *arXiv preprint*.

Tianyi Xiong, Xiyao Wang, Dong Guo, Qinghao Ye, Haoqi Fan, Quanquan Gu, Heng Huang, and Chunyuan Li. 2024. Llava-critic: Learning to evaluate multimodal models. *arXiv preprint arXiv:2410.02712*.

Tengyu Xu, Eryk Helenowski, Karthik Abinav Sankararaman, Di Jin, Kaiyan Peng, Eric Han, Shaoliang Nie, Chen Zhu, Hejia Zhang, Wenxuan Zhou, et al. 2024. The perfect blend: Redefining rlhf with mixture of judges. *arXiv preprint arXiv:2409.20370*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. 2022. Large language models meet nl2code: A survey. *arXiv preprint arXiv:2212.09420*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

A Appendix

A.1 Proof of Theorem 1 Extended to K evaluators

We present the proof of Theorem 4 generalized to K evaluators.

Proof. First, we characterize the sub-optimality of our proposed critique method as $\Delta = \mathbb{E}_{c^* \sim \pi_c^*}[c^*] - \mathbb{E}_{c_1 \sim \pi_1(\cdot|x,y), c_2 \sim \pi_2(\cdot|x,y) \cdots \pi_K}[g(c_1, c_2, c_3 \cdots c_K)]$. Note that if Δ is zero, we have the optimal critique. Thus, we want Δ to be as low as possible.

$$\begin{aligned}\Delta &= \mathbb{E}_{c^* \sim \pi_c^*}[c^*] \\ &\quad - \mathbb{E}_{c_1 \sim \pi_1(\cdot|x,y), c_2 \sim \pi_2(\cdot|x,y) \cdots \pi_K}[g(c_1, c_2, ..., c_K)] \\ &= \underbrace{\mathbb{E}_{c^* \sim \pi_c^*}[c^*] - \mathbb{E}_{c \sim \pi_d(\cdot|x,y)}[c]}_{\Delta_1} + \\ &\underbrace{\mathbb{E}_{c \sim \pi_d(\cdot|x,y)}[c] - \mathbb{E}_{c_1 \sim \pi_1(\cdot|x,y), \cdots \pi_K}[g(c_1, c_2, ..., c_K)]}_{\Delta_2},\end{aligned}$$

where we add and subtract the terms $\mathbb{E}_{c \sim \pi_d(\cdot|x,y)}$, with $\pi_d = \sum_{i=1}^K \alpha_i \pi_i$ ($\sum_{i=1}^K \alpha = 1$) and then separate the two terms as Δ_1, Δ_2. We next individually analyze the terms Δ_1, Δ_2.

We can now bound Δ_1 as,

$$\begin{aligned}\Delta_1 &= \mathbb{E}_{c^* \sim \pi_c^*}[c^*] - \mathbb{E}_{c \sim \pi_d(\cdot|x,y)}[c] \\ &\leq |c^*| d_{\text{TV}}(\pi^*, \pi_d) \\ &= |c^*| d_{\text{TV}}(\pi^*, \sum_{i=1}^K \alpha_i \pi_i),\end{aligned}$$

where we use the property of integral probability metric to bound Δ_1 as the total variation distance between the optimal critique policy and the mixture critique policy. Next, we proceed to Δ_2,

$$\begin{aligned}\Delta_2 =& \mathbb{E}_{c \sim \pi_d(\cdot|x,y)}[c] - \\ & \mathbb{E}_{c_1 \sim \pi_1(\cdot|x,y), \cdots, c_K \sim \pi_K(\cdot|x,y)}[g(c_1, \cdots, c_K)] \\ =& \mathbb{E}_{c \sim \pi_d(\cdot|x,y)}[c] - \\ & \mathbb{E}_{c_1 \sim \pi_1(\cdot|x,y), \cdots, c_K \sim \pi_K(\cdot|x,y)}\left[\sum_{i=1}^K \alpha_i c_i\right] \\ =& \mathbb{E}_{c^* \sim \pi_d(\cdot|x,y)}[c^*] - \sum_{i=1}^K \alpha_i \mathbb{E}_{c_i \sim \pi_i(\cdot|x,y)}[c_i] \\ =& 0, \end{aligned} \tag{7}$$

where we expand upon the definition of Δ_2 and use Assumption 1 on the aggregation function. Under this assumption, the two terms cancel out with the final result $\Delta_2 = 0$. Combining both terms concluded the proof.

A.2 Judge System Prompt

We provide the judge system prompt in Figure 3. For Single-Eval the system prompt is given to only one LLM call and all the roles utilized are listed together in [INSERT UTILIZED CRITERIA]. For ensemble with separate criteria, each evaluator gets one specified in [INSERT UTILIZED CRITERIA].

A.3 Baseline Details

Here are the details for the two baselines that do not incorporate a separate evaluation protocol.

Self-Refine: In self-refine (Madaan et al., 2024), the system prompt p is constant throughout the initial generation, feedback, and update stages. During the feedback and update stages, the user prompts is modified to specify that an output is already given and the LLM must either now self-reflect to generate feedback or must use both the output and feedback to generate and update the response. We provide the feedback and user prompts in Figure 4.

Vanilla Feedback Loop: A separate LLM provides feedback to the LLM generator. The system prompt for the update generation is different than the one for the initial generation.

We provide addtional results with HumanEval (Chen et al., 2021) and EvoEval (Xia et al., 2024) benchmarks. HumanEval is a standard code generation benchmark and EvoEval (we specifically use EvoEval-Difficult) is a more recent one that adapts the questions of HumanEval to have more additional constraints and requirements. We see that for the harder benchmark of EvoEval, the benefits of ensembling LLM judges is more clear.

Evaluation System Prompt for Evaluator LLM

"You are a smart language model that evaluates code snippets. You do not solve problems or propose new code snippets, only evaluate existing solutions critically and give very concise feedback. Please focus on [INSERT UTILIZED ROLE]. DO NOT PROPOSE NEW CODE!!!"

Figure 3: Judge System Prompt.

Feedback System Prompt for Vanilla Feedback Loop	Feedback User Prompt for Self-Refine and Vanilla Feedback Loop
You are a Python programming assistant. \n You will be given a function implementation.\n Your goal is to write a few sentences to explain why your implementation is wrong if you think it is. \n You will need this as a hint when you try again to implement the function later. \n Please be as concrete and actionable as you can to improve the code. \n Please focus your feedback on the following **[INSERT NUMBER OF CRITERIA]**criteria: **[INSERT CRITERIA]**. \n Only provide the few sentence description in your answer, DO NOT PROPOSE NEW CODE!!!!!!!	Please provide feedback on the code you have just generated below. \n **[INSERT CODE]** \n Please be as concrete and actionable as you can to improve the code. \n Please focus your feedback on the following **[INSERT NUMBER OF CRITERIA]** criteria: **[INSERT CRITERIA]**.' DO NOT PROPOSE NEW CODE!!!!!
Update System Prompt for Vanilla Feedback Loop	**Update User Prompt for Self-Refine and Vanilla Feedback Loop**
You are a Python programming assistant. \n You will be given a function implementation and feedback on how to improve it.\n Please generate new code based on the current generated code and the feedback generated for it. \n YOUR RESPONSE SHOULD ONLY CONTAIN CODE!!! NO ENGLISH OR NATURAL LANGUAGE!!!! \n Restate the function signature in the beginning of your output.	Please generate new code based on the current generated code and the feedback you just generated for it. \n Current code: \n **[INSERT CODE]** \n Feedback: **[INSERT FEEDBACK]** \n YOUR RESPONSE SHOULD ONLY CONTAIN CODE!!! NO ENGLISH OR NATURAL LANGUAGE!!!!

Figure 4: System Prompts for Vanilla Feedback Loop (Left) and User Prompts for Self-Refine and Vanilla Feedback Loop (Right)

Judge Method	Agg	EvoEval CR (%)	HumanEval CR (%)
Vanilla Feedback Loop (Baseline)	–	40.0 ± 16.33	$\mathbf{90.0 \pm 0.0}$
Self-Refine (Baseline)	–	33.33 ± 9.43	$\mathbf{90.0 \pm 0.0}$
1 Judge (Baseline)	–	50.0 ± 0.0	$\mathbf{90.0 \pm 0.0}$
6 Judges – All Criteria	C	40.0 ± 8.16	86.67 ± 4.71
6 Judges – All Criteria	Sum	50.0 ± 0.0	86.67 ± 4.71
6 Judges – All Criteria	Sel	44.44 ± 7.86	86.67 ± 4.71
6 Judges – One Criterion Each	C	50.0 ± 8.16	$\mathbf{90.0 \pm 0.0}$
6 Judges – One Criterion Each	Sum	50.0 ± 8.16	86.67 ± 4.71
6 Judges – One Criterion Each	Sel	$\mathbf{60.0 \pm 8.16}$	82.59 ± 5.32

Table 5: The CR over HumanEval and EvoEval-Difficult comparing various ensemble evaluation methods against inference-time improvement baselines. Ensemble methods consistently outperform baselines CR and the two highest ranking methods in terms of readability are ensemble. The difference in CR and RP between ensemble methods emphasize the non-trivial nature of designing the ensemble evaluation protocol. We use 10 questions from each dataset and use 4 iterations of prompt optimization per question.

Judge Temperature	Judge Method	CR (%)
0	Single (Baseline)	76.67 ± 4.71
	Ensemble - All Criteria	71.67 ± 4.71
	Ensemble - One Criterion Each	73.33 ± 4.71
0.25	Single (Baseline)	71.67 ± 4.71
	Ensemble - All Criteria	73.33 ± 6.24
	Ensemble - One Criterion Each	71.67 ± 4.71
0.50	Single (Baseline)	75.0 ± 4.08
	Ensemble - All Criteria	73.33 ± 6.24
	Ensemble - One Criterion Each	78.33 ± 2.36
0.75	Single (Baseline)	75.0 ± 4.08
	Ensemble - All Criteria	76.67 ± 4.71
	Ensemble - One Criterion Each	71.67 ± 8.5
1	Single (Baseline)	66.67 ± 4.71
	Ensemble - All Criteria	78.33 ± 2.36
	Ensemble - One Criterion Each	71.67 ± 8.50

Table 6: **Temperature Ablation:** This table summarizes the CR for the various evaluation methods given different combinations of roles. We report the mean and standard deviation of 3 trials for CR.

A.4 Ensemble Methods Outpeform Single-Judge Method with Incorrect Judge.

To highlight the robustness of ensemble judges to incorrect evaluations, we introduce an adversarial evaluator. For ensemble, methods with separate evaluation instructions, we specify in the system prompt of the correctness judge to always generate a critique stating that the code solution works. Similarly, for methods where the system prompt has all the criteria, we specify to output a critique claiming that code works when discussing correctness. We repeat the same experiment but attack the readability criteria by instructing the evaluation protocols to generate a critique stating that the code is readable. We run all prompt optimization processes for 4 iterations.

The results of both experiments are shown in Table 7. In both experiments, ensemble methods still outperforms the single-judge approach in terms of CR, with the worst-performing ensemble method having at least the same mean CR as a single-judge. We believe this results is intuitively because having multiple critiques will lessen the influence of one wrong one.

Judge Method	Aggregation	CR (%)
Single	–	68.33 ± 6.24
6 - All Criteria	C	75.0 ± 4.08
6 - All Criteria	Sum	75.0 ± 4.08
6 - All Criteria	Sel	$\mathbf{78.33 \pm 4.71}$
6 - Separate Criteria	C	76.67 ± 8.5
6 - Separate Criteria	Sum	76.67 ± 6.24
6 - Separate Criteria	Sel	71.67 ± 11.79

Table 7: CR on LeetCodeHard with one purposefully incorrect judge that always says the code is correct. Unsurprisingly, ensemble still outperforms single.