CONV-TO-BENCH: EVALUATING LANGUAGE MODELS VIA USER–ASSISTANT DIALOGUES IN CODE TASKS

Victor M. dos Santos[*1,3], **Andre C. Castro**[*2,3,4], **Samuel L. de S. Toledo**[2,3], **Bruno M. L. Calura**[2,3], **Lisandra C. de M. Menezes**[2,3], **Raul C. R. Mata**[3], **Telma W. de L. Soares**[2,4], **Bryan L. M. de Oliveira**[2,3,4]

[1]Institute of Mathematics and Computer Science, University of São Paulo, Brazil
[2]Institute of Informatics, Federal University of Goiás, Brazil
[3]HUG Labs, Brazil
[4]Advanced Knowledge Center for Immersive Technologies (AKCIT), Brazil

ABSTRACT

The rapid advancement of Large Language Models (LLMs) has outpaced the scalability of traditional evaluation benchmarks, which remain heavily dependent on labor-intensive expert curation. We address this bottleneck with Conv-to-Bench [1], a multi-stage framework that automatically transforms authentic multi-turn user-assistant dialogues into structured, verifiable requirement checklists. By leveraging the "instructional evolution" found in real-world conversational logs, our approach deconstructs fragmented user intent into consolidated instructions and binary evaluation criteria. Applied to the programming domain, Conv-to-Bench produces evaluation sets that demonstrate near-perfect alignment with human-authored standards like BigCodeBench, achieving Spearman correlations of up to $\rho = 1.000$ with significantly lower computational overhead. Validation of the LLM-as-a-judge framework further confirms its reliability, with the primary evaluator achieving substantial agreement with human-verified ground truth ($\kappa = 0.705$). Our comprehensive ablation studies reveal that while multi-turn interactions capture the iterative evolution of user intent, instruction-centric extraction provides a more robust foundation. Ultimately, Conv-to-Bench provides a scalable, cost-effective paradigm for maintaining high-fidelity evaluation standards as user-centric AI applications continue to diversify.

1 INTRODUCTION

The development of robust evaluation benchmarks is a fundamental pillar for measuring progress and ensuring the reliability of Large Language Models (LLMs). Established benchmarks such as Chen et al. (2021), Zhuo et al. (2025), and (Rein et al., 2024) have set the gold standard for evaluating complex reasoning and code generation, providing rigorous frameworks that are widely trusted by the research community. However, a significant challenge in the creation of such high-quality benchmarks is their deep reliance on human expertise throughout the entire construction process, from the manual curation of tasks to multi-stage verification by subject matter experts. While this human-intensive approach ensures high fidelity and precision, it also creates a substantial bottleneck, making the development of new, diverse evaluation sets a resource-heavy and time-consuming endeavor.

In parallel, the widespread adoption of LLMs has generated an immense repository of real-world data in the form of conversational datasets, such as Zheng et al. (2024) and Zhao et al. (2024). These datasets represent a valuable source of authentic user intent, capturing millions of dialogues that reflect practical challenges across various domains. These interactions are often rich, iterative exchanges where users refine their requests and correct model outputs through clarifications, constraints, and rephrasings. As noted by Don-Yehiya et al. (2024), these natural interactions provide

[*]Equal contribution. Emails: victormoreli@usp.br, andre.castro@egresso.ufg.br.
[1]Code is available at https://github.com/vmoreli/conv-to-bench

a form of implicit feedback, representing a dynamic instructional evolution that is often absent in static, expert-authored benchmarks.

To leverage this potential, we introduce **Conv-to-Bench**, a multi-stage framework (Figure 1) designed to automatically transform these multi-turn dialogues into structured and verifiable requirements checklists. Our approach shifts the evaluative focus from manually authored problems to the systematic extraction of instructions accumulated throughout an entire interaction. By processing these dialogues, Conv-to-Bench reconstructs the evolving constraints of a task as defined by the user, effectively capturing the nuanced requirements that emerge during a conversation.

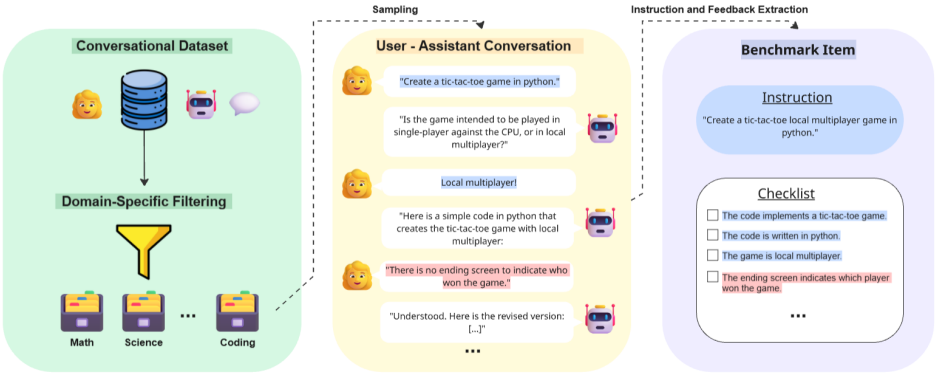

Figure 1: **Overview of the framework.** The diagram illustrates the proposed multi-stage process for transforming raw, multi-turn conversational data into structured (instruction, requirement checklist) evaluation pairs. In the user messages, text highlighted in blue corresponds to extracted instructions, while text highlighted in red represents user feedback.

The central goal of this work is to determine whether such an automated, dialogue-driven pipeline can serve as a functional equivalent to traditional, expert-dependent benchmarks. We specifically investigate if requirements extracted from the instructional evolution of multi-turn dialogues can approximate the evaluative integrity of established professional standards. Furthermore, through a targeted ablation study, we seek to answer whether the implicit feedback naturally embedded in these conversations serves as a useful refinement signal that improves evaluation accuracy or if it primarily introduces noise into the process when compared to instruction-only baselines. Finally, we verify the reliability of our automated judge through a consistency analysis to ensure its verdicts align with authoritative benchmarks. This approach validates a more scalable, user-centered evaluation paradigm that minimizes manual dependency while leveraging the inherent richness of authentic user-model interactions.

2 Related Work

The availability of massive, authentic human-AI interaction logs has provided a significant repository for exploring LLM utility. Datasets such as LMSYS-Chat-1M (Zheng et al., 2024) and WildChat (Zhao et al., 2024) record millions of real-world dialogues, revealing a strong predominance of topics related to software development and programming assistance. While WildChat highlights the iterative nature of these interactions, where over 40% of dialogues span multiple turns, these resources have been primarily used for training or behavioral analysis. Conv-to-Bench explores these datasets as a foundation for benchmark construction, treating the iterative clarifications and requirement refinements found in multi-turn exchanges as a source of dynamic ground truth.

However, extracting reliable evaluation metrics from interaction logs is non-trivial. This signal presents significant challenges: Liu et al. (2025) demonstrated that while informative, implicit feedback can be "noisy as a learning signal," particularly for complex reasoning tasks. Furthermore, other studies indicate that naive heuristics, such as optimizing for conversation length, may inadvertently reinforce undesirable behaviors, including controversial or unfriendly model responses (Pang et al., 2024). While prior work primarily treats implicit feedback as a signal for supervised fine-

tuning or reranking, we instead explore its potential as a direct source of fine-grained evaluation criteria.

Traditionally, achieving high precision in evaluation has been an expert-intensive and human-dependent process, creating a significant bottleneck for scalability. For instance, HumanEval (Chen et al., 2021) required the manual creation of 164 original problems to avoid data contamination. Similarly, BigCodeBench (Zhuo et al., 2025) involved a year-long construction process by 20 authors, where 75% of the annotators possessed more than five years of Python expertise. Beyond coding, the GPQA (Rein et al., 2024) and Humanity's Exam (Phan et al., 2025) benchmarks illustrate the extreme cost of specialized knowledge, requiring subject matter experts with advanced academic backgrounds to author and validate questions that are often "expert-hard." While these benchmarks serve as gold standards, their dependency on long-term expert involvement makes them difficult to scale across the vast spectrum of emerging user-centric tasks.

To scale evaluation, automated pipelines like Li et al. (2025) leverage LLM-based scoring on crowd-sourced data. However, benchmarks like Arena-Hard-Auto focus on single-turn interactions, overlooking the evolving nature of user intent. Conv-to-Bench addresses this gap by incorporating multi-turn dynamics and investigating the impact of cumulative instructions and implicit feedback. By capturing instructional evolution and corrective signals, our framework provides a representative measure of model adherence while assessing whether such feedback yields meaningful refinement or introduces noise into the evaluation process.

3 METHODOLOGY

Our methodology establishes a systematic, multi-stage framework to transform raw user–assistant dialogues into structured evaluation data. The primary goal is to deconstruct complex user-assistant dialogues into atomic components: a consolidated user instruction and a detailed requirement checklist derived from the conversational context. This framework enables a nuanced, context-aware assessment of generative models that goes beyond simple instruction-following and incorporates the user's iterative refinements and corrections.

The methodology consists of three main stages, built upon a foundational dataset of multi-step user-assistant interactions.

3.1 DOMAIN-SPECIFIC FILTERING

The first stage of the framework involves domain-specific filtering to isolate high-utility interactions from raw conversational logs. We posit that generic conversational data are too broad for a specialized evaluation. Many real-world dialogues lack an objective goal, or they involve tasks with ambiguous, non-deterministic success criteria that defy consistent measurement. Furthermore, without rigorous filtering, the thematic scope of the resulting benchmark is undefined, reducing its utility as a targeted assessment tool. Therefore, each conversation is initially classified to determine its relevance to a target domain (e.g., programming, creative writing), ensuring that all subsequent processing stages operate on a high-signal, in-domain corpus, which improves not only the computational efficiency of the pipeline but also the evaluative relevance and integrity of the extracted requirements.

To achieve this, we employ a two-tiered filtering strategy that combines unsupervised clustering and targeted classification. First, we utilize a topic modeling pipeline to group conversations into thematic clusters based on dense vector representations. These clusters are automatically screened against a domain-specific keyword lexicon to identify candidate groups likely to contain relevant content. Second, to eliminate thematic noise, individual conversations from these candidate clusters undergo an instance-level verification via a zero-shot LLM-based classifier. By analyzing the initial user message, the classifier executes a binary determination of the prompt's relevance to the target domain. This second tier filters out ambiguous interactions that may have been co-located within a relevant cluster, ensuring categorical purity to the synthesis stages.

3.2 INSTRUCTION SYNTHESIS

Once a conversation is identified as in-domain, the next stage synthesizes the complete user instruction. A significant challenge in multi-turn dialogues is that the user's complete intent is often not contained in a single message, being fragmented over the dialogue. The initial prompt may be vague, with critical constraints and modifications provided in subsequent turns. This stage is designed to synthesize a single, comprehensive instruction by analyzing the user's complete conversational history via LLM, integrating the initial request with all follow-up clarifications. To maintain benchmark integrity, the synthesized instructions are passed through an LLM-based binary classifier. This stage filters out high-noise candidates, specifically discarding instructions that are identified as semantically ambiguous, too vague, or dependent on external files not present within the conversational logs.

3.3 FEEDBACK-DRIVEN REQUIREMENT SYNTHESIS

The final stage generates the requirement checklist, which serves as the core evaluation metric. This process is executed in two discrete LLM-driven steps: feedback identification and structured checklist synthesis.

First, an LLM analyzes the dialogue to identify and classify user messages that constitute evaluative feedback. To ensure high precision, the model adheres to a strict "reaction-based" protocol: the first message in a conversation is never feedback, and neutral messages that merely continue the dialogue must be ignored. Feedback is categorized into two types:

- **Positive Feedback:** Messages where the user confirms that the assistant's response was successful, correct, or met their needs (e.g., explicit acknowledgment of utility or successful code execution).
- **Negative Feedback:** Messages indicating the assistant's response was unsatisfactory, incorrect, or incomplete. This includes direct corrections to the code or requests for clarification that suggest the previous answer failed.

Importantly, silence is not interpreted as positive feedback; the framework requires an explicit evaluative signal from the user to identify a message as a feedback source.

In the second step, the framework acts as an expert Quality Assurance (QA) analyst. It receives the dialogue, the synthesized instruction and the explicitly identified feedback message IDs. The LLM translates these inputs into a list of simple, atomic and testable requirements formatted as binary (Yes/No) questions.

To ensure full traceability, the model is instructed to tag each requirement with its origin, distinguishing between criteria derived from the synthesized instruction and those stemming from specific feedback turns. By isolating the feedback-derived items, we investigate whether the implicit feedback naturally embedded in dialogues provides a meaningful refinement signal that improves evaluation accuracy or not. The result is a robust (instruction, requirement checklist) pair where every evaluation criterion is grounded in a specific point of the original user-model interaction.

3.4 SCORING AND STATISTICAL CALIBRATION

The final component of the methodology establishes the formal scoring mechanism used to translate checklist fulfillments into a standardized performance metric. Because requirements are nested within instructions, and instructions vary significantly "passed items" would introduce structural bias. To mitigate this, we implement a hierarchical estimation pipeline that treats each instruction as a distinct cluster of evaluative constraints. For a given model output, each atomic requirement j within an instruction i is evaluated as a binary outcome $y_{i,j} \in \{0, 1\}$. To ensure that the final score reflects a balanced mastery of the task rather than the sheer quantity of requirements, we first compute an Instruction-Level Score (S_i) by averaging the binary outcomes of its n_i requirements:

$$S_i = \frac{1}{n_i} \sum_{j=1}^{n_i} y_{i,j}$$

This normalization ensures that an instruction with twenty simple requirements does not exert disproportionate influence over an instruction with three complex, high-stakes requirements. To estimate the final model performance θ across a corpus of N instructions, we treat each instruction as a cluster. The global performance is defined as the mean of these instruction-level scores:

$$\theta = \frac{1}{N} \sum_{i=1}^{N} S_i$$

To calculate the statistical uncertainty and account for the variance in task difficulty, we employ a Cluster Bootstrap method (Field & Welsh, 2007) (Efron & Tibshirani, 1994). We generate $B = 1,000$ bootstrap replicates by resampling the instructions (clusters) with replacement. For each replicate b, we compute the bootstrap mean θ_b^*. The 95% confidence interval (CI) is then derived from the 2.5th and 97.5th percentiles of the bootstrap distribution. These bootstrap-derived confidence intervals offer a transparent measure of the model's reliability and the benchmark's stability.

4 Experimental Design and Implementation

4.1 Dataset and Pipeline Implementation

The framework described in Section 3 was implemented as a multi-stage pipeline designed to transition from massive raw conversational corpora to a high-quality, domain-specific evaluation dataset. We utilized the LMSYS-Chat-1M Zheng et al. (2024) and WildChat Zhao et al. (2024) datasets, which together provide 2 million authentic dialogues. For all LLM-based stages, we employed Gemini-2.5-Flash (Gemini Team, 2025). We focused our experiments on the programming domain, as it represents a high-impact application area where multi-turn refinement and objective correctness are both prevalent and critical to evaluate.

The pipeline execution and the resulting data flow are detailed below:

1. **Pre-processing and Initial Filtering:** Conversations were first filtered to include only English dialogues not flagged for toxicity. This formed the baseline for our domain-specific extraction.
2. **Domain-Specific Clustering:** To isolate the programming domain, we implemented a clustering stage using all-MiniLM-L6-v2 (Reimers & Gurevych, 2019) for embeddings and BERTopic (Grootendorst, 2022) for organization. This produced 2,304 clusters for WildChat and 3,780 for LMSYS. Small-parameter encoders ensured low computational overhead for the initial sweep compared to LLMs. By screening the resulting clusters against a lexicon of 115 code-related keywords, we identified 101 and 387 code-specific clusters, respectively, successfully distilling the datasets into a relevant pool of 56,680 dialogues.
3. **Sub-sampling and Domain Confirmation:** From the identified code clusters, we randomly sampled 1,000 conversations (500 from each source); this sample size was selected to manage the computational costs associated with LLM inference across massive datasets. A binary LLM classifier was then used to confirm domain relevance, resulting in 545 confirmed programming conversations.
4. **Instruction Synthesis:** For the confirmed samples, an LLM call synthesized the core user instruction. Subsequently, a filtering stage was applied to remove invalid entries (examples available in Appendix E), resulting in 387 high-quality instructions. This yield is satisfactory given the noise of publicly available corpora and is considered sufficiently representative for reliable model evaluation, as research suggests that small, curated benchmarks can effectively replicate results from much larger datasets (Polo et al., 2024).
5. **Feedback Extraction and Requirement Generation:** For each valid instruction, a specialized LLM call extracted user feedback from the multi-turn history. Finally, a fifth call integrated the instruction and feedback to generate the final Requirement Checklist.

As a result of this pipeline, we produced an evaluation dataset of 387 unique (instruction, requirement checklist) pairs (examples available in Appendix F), derived from real-world user interactions

but refined for high-impact programming analysis. This dataset comprises a total of 2,851 checklist items, with 2,595 requirements derived directly from instructions and 256 extracted from user feedback, naturally mirroring the scarcity of explicit feedback. Detailed system prompts and the templates used for this process are available in Appendix A.

4.2 Reference Standards and Metrics

To establish the evaluative integrity of Conv-to-Bench, we benchmark its performance against established, human-intensive standards and utilize statistical correlation metrics to quantify its alignment with expert-authored tasks.

We selected BigCodeBench (Zhuo et al., 2025) as our primary reference standard. As one of the most rigorous benchmarks for code generation, it requires the mastery of complex library interactions and multi-step reasoning, making it a challenging upper bound for an automated pipeline to emulate. We utilize two specific subsets for our comparison: BigCodeBench-Hard, a subset of 150 tasks that are more challenging; and BigCodeBench-Full, which is the complete suite of 1140 tasks. Details about the overlap of the selected models with these subsets on Appendix C.

For each subset, we report performance across two completion modes: Complete, where models generate code from a docstring; and Instruct, which assesses code generation based on brief natural language instructions.

4.3 Validation Design

To verify that our automated LLM-judges reflect genuine quality, we implemented a validation layer via expert annotation. This stage moves beyond automated rankings to ensure that the atomic requirements generated by Conv-to-Bench are accurately assessed against a human-verified ground truth.

We established a gold-standard dataset through expert review of a representative sample of model outputs, balanced across the eight evaluated models to prevent architectural bias. Experts performed the annotation via a custom-built interface, which can be seen in Appendix G, where they were presented with the model response alongside the specific requirement under test. For each item, the annotator provided a binary ground-truth label ($y \in \{0, 1\}$) indicating whether the response satisfied the criterion.

To quantify "evaluative integrity," we compared the automated verdicts against this ground truth using a stratified sample of 488 points per evaluator. This analysis was further disaggregated into Instruction-based and Feedback-based requirements to detect potential reliability degradation as conversational complexity increases. Following Badshah & Sajjad (2025), who demonstrate that robust evaluation of generative outputs requires metrics capable of capturing semantic depth beyond simple overlaps, we established a "Judge Profile" using the following metrics:

- **F1-Score:** Provides a balanced evaluation of precision and recall at both the class level and across classes (macro-averaged), ensuring the scoring is not skewed by a leniency bias.
- **Cohen's Kappa (κ):** Quantifies inter-rater agreement between the LLM and human experts while accounting for agreement by chance; we consider $\kappa > 0.6$ to represent substantial agreement (Landis & Koch, 1977).

This rigorous validation framework ensures that benchmark scores reflect authentic model capability rather than artifacts of judge behavior. Another aspect of the LLM-as-a-judge framework we investigated is potential correlation between response length and scores. As shown in Appendix B, the low Pearson correlation coefficients confirm the absence of verbosity bias in our evaluators.

4.4 Comparative and Ablative Framework

The final component of our experimental design defines the configurations used to isolate the impact of conversational feedback and the methodology for a rigorous statistical comparison with existing automated baselines.

To quantify the value of multi-turn interactions, we formally define two variants of our benchmark based on the origin of the evaluative criteria:

- **Instructions-Only:** In this configuration, the requirement checklist is restricted exclusively to items derived from the synthesized instruction. As detailed in Section 3.2, this instruction represents the consolidated user intent extracted from the entire conversational history, rather than just the initial message, integrating the primary request with all subsequent task-related refinements. Crucially, this configuration excludes user feedback regarding the success or failure of previous responses, focusing solely on the evolving task requirements.
- **Full (Instructions + Feedbacks):** This variant utilizes the complete synthesized checklist by integrating requirements from two distinct sources: the task instruction and user evaluations. It combines the instruction synthesized from the conversational history (Section 3.2) with both Positive and Negative Feedback (Section 3.3).

By comparing these two variants, we can isolate the specific refinement signal provided by multi-turn dialogues and determine if it enhances the benchmark's ability to mirror expert-authored standards.

To assess the relative performance of Conv-to-Bench, we compare it against Arena-Hard-Auto (Code) (Li et al., 2025), a state-of-the-art framework that also automatically generates benchmarks from conversational data. From their results, we extracted 150 prompts which were categorized as "code related". As our pipeline has 387 instructions, to ensure a fair comparison and account for the variance in task difficulty, we implement a Subsampling Bootstrap procedure. We executed $B = 1,000$ iterations where, in each run, a fixed subset of $n = 150$ instructions was sampled without replacement to match the cardinality of Arena-Hard-Auto (Code). For each bootstrap replicate, we performed hierarchical aggregation: requirement-level scores were first averaged per instruction to prevent length bias, and these means were then aggregated into a model-level performance estimate (θ_{boot}). The final scores and 95% confidence intervals (CI) were derived from the 2.5th and 97.5th percentiles of the resulting bootstrap distribution, ensuring that our ranking correlations (ρ and τ) are robust against specific sample compositions.

4.5 Model Selection and Evaluation Settings

To validate the proposed benchmark, we conducted an evaluation across a diverse set of eight prominent Large Language Models. These models were selected primarily based on their inclusion in the BigCodeBench (Zhuo et al., 2025), the reference benchmark used for our comparative analysis, ensuring sufficient overlap for the correlation verification presented in Section 5. The evaluation set balances general-purpose and code-specific architectures, focusing on efficient, smaller-scale models: GPT-4.1-Nano-2025-04-14 (OpenAI, 2024), Gemini-2.0-Flash-001 (Gemini Team, 2025), Qwen2.5-Coder-7B-Instruct (Qwen, 2025), CodeQwen1.5-7B-Chat (Bai et al., 2023), DeepSeek-Coder-6.7B-Instruct (Guo et al., 2024), CodeLlama-13B-Instruct (Rozière et al., 2024), CodeLlama-7B-Instruct (Rozière et al., 2024), and CodeGemma-7B-it (CodeGemma Team, 2024).

For the LLM-as-a-judge scoring framework, we selected three models representing both proprietary and open-weights paradigms: Gemini-2.5-Flash, GPT-5-Mini, and Deepseek-Coder-33B-Instruct. This selection ensures architectural diversity and a strict separation of roles: no specific model version or generation overlaps between the evaluated and judging sets. This tiered approach across model families mitigates self-preference bias while promoting cross-model alignments. Notably, Deepseek-Coder-33B-Instruct demonstrated lower reliability than its proprietary counterparts, as detailed in Section 5.1.

5 Results

In this section, we present the evaluation of Conv-to-Bench. We begin by an evaluation of the reliability of our LLM-as-judge framework using classification metrics, followed analyzing the impact of instructional evolution through ablation studies. We then compare our findings with the Arena-Hard-Auto (Code).

Table 1: **Classification performance for the evaluator models shows that while results are not perfect, the Macro F1-score and Cohen's κ are satisfactory for proprietary models in automated code evaluation tasks within the Conv-to-Bench code benchmark.** Higher values indicate better classification performance.

	F1-score			
Evaluator	Negative	Positive	Macro	Cohen's κ
Gemini-2.5-Flash	0.773	0.932	0.852	0.705
GPT-5-Mini	0.740	0.907	0.824	0.649
Deepseek-Coder-33B-Instruct	0.574	0.887	0.731	0.464

5.1 LLM-AS-JUDGE RELIABILITY ANALYSIS

We evaluated the reliability of our LLM-based verification framework by examining its classification performance against expert-labeled data, as summarized in Table 1. The models generally demonstrate satisfactory classification skills for the demands of automated code evaluation, particularly in identifying successful fulfillment. This is evidenced by the high positive F_1-scores across all evaluators: 0.932 for Gemini-2.5-Flash, 0.907 for GPT-5-Mini, and 0.887 for Deepseek-Coder-33B-Instruct.

However, a performance gap is observable in the inter-rater reliability metrics. While the Cohen's Kappa (κ) coefficients for Gemini (0.705) and GPT-5-Mini (0.649) indicate a *substantial* level of agreement with human experts (Landis & Koch, 1977), the DeepSeek model achieved a lower coefficient of 0.464. This indicates a higher degree of label misalignment between the human expert and the DeepSeek evaluator. This divergence is especially pronounced in the negative class detection ($F_1 = 0.574$), suggesting that the model may follow a more permissive evaluative logic or have a different threshold for what constitutes a requirement failure compared to human experts. Due to this lower consistency in detecting omissions or incorrect logic, the subsequent analyses in this study focus on the results provided by the Gemini and GPT evaluators.

Overall, the results for the primary evaluators remain satisfactory. The relative difficulty in negative-class classification likely reflects the complex technical nature of verifying code against evolving requirements. We propose the hypothesis that further increasing the model scale or utilizing models with enhanced reasoning capabilities might lead to a corresponding improvement in these classification metrics, particularly in resolving subtle negative-class edge cases.

5.2 ABLATION STUDY: INSTRUCTIONS-ONLY VS. FULL PIPELINE

To assess the impact of different conversational components, we compared the Instructions-Only configuration against the Full Pipeline, which integrates iterative user feedback. The comparative performance across the BigCodeBench Full and Hard sets is detailed in Tables 2 and 3. The operational token costs for running Conv-to-Bench (Full Pipeline) are detailed in Appendix D.

The results indicate that the inclusion of feedback does not yield consistent improvements in evaluative integrity. While user feedback can provide valuable refinement signals, it is often inextricably linked with conversational noise. Authentic dialogues frequently involve a complex interplay of contradictory refinements and ambiguous corrections. This inherent difficulty in disentangling constructive signals from erratic noise leads to inconsistent performance across metrics: in some experimental scenarios, the inclusion of feedback improved performance, while in others, the correlation with human-authored benchmarks decreased or resulted in a tie.

Given this variability, our analysis suggests that Instructions-Only extraction offers a more stable and robust foundation for building reliable benchmarks. This configuration demonstrated exceptionally high alignment, specifically achieving near-perfect Spearman correlations ($\rho = 1.000$) within the Instructions-Only subset on the Full Set across both the "instruct" and "complete" variants (see Table 2). This is a notable distinction, as the Full Pipeline configuration only achieved this level of alignment in the "instruct" variant.

Table 2: **Conv-to-Bench code benchmark scores show near-perfect correlation with the BigCodeBench (Full Set), with Instructions-Only scores standing out for their exceptionally high alignment.** Spearman (ρ) and Kendall (τ) correlations, reporting coefficients and p-values. The highest correlations (ρ) per metric are bolded. A higher ρ (closer to 1.0) indicates a stronger correlation, while a low p-value (e.g., < 0.05) indicates statistical significance.

		BigCodeBench (Instruct)				BigCodeBench (Complete)			
		Spearman		Kendall		Spearman		Kendall	
Evaluator	Subset	ρ	p	τ	p	ρ	p	τ	p
Gemini-2.5-Flash	Full	**1.000**	<0.001	**1.000**	0.003	0.943	0.005	0.867	0.017
	Instructions-Only	**1.000**	<0.001	**1.000**	0.003	0.943	0.005	0.867	0.017
GPT-5-Mini	Full	0.886	0.019	0.733	0.056	0.943	0.005	0.867	0.017
	Instructions-Only	0.943	0.005	0.867	0.017	**1.000**	<0.001	**1.000**	0.003

Table 3: **Conv-to-Bench code benchmark demonstrates robust consistency on the BigCodeBench (Hard Set), showing a more balanced performance across subsets than observed in the BigCodeBench (Full Set).** Spearman (ρ) and Kendall (τ) correlations, reporting coefficients and p-values. The highest correlations (ρ) per metric are bolded. A higher ρ (closer to 1.0) indicates a stronger correlation, while a low p-value (e.g., < 0.05) indicates statistical significance.

		BigCodeBench (Instruct)				BigCodeBench (Complete)			
		Spearman		Kendall		Spearman		Kendall	
Evaluator	Subset	ρ	p	τ	p	ρ	p	τ	p
Gemini-2.5-Flash	Full	0.952	<0.001	0.857	0.002	**0.994**	<0.001	**0.982**	0.001
	Instructions-Only	0.952	<0.001	0.857	0.002	**0.994**	<0.001	**0.982**	0.001
GPT-5-Mini	Full	**0.976**	<0.001	**0.929**	<0.001	0.946	<0.001	0.837	0.004
	Instructions-Only	0.952	<0.001	0.857	0.002	0.970	<0.001	0.909	0.002

5.3 Comparison with Arena-Hard-Auto (Code)

We benchmarked the ranking stability of Conv-to-Bench, utilizing the Instructions-Only configuration, against the Arena-Hard-Auto (Code) benchmark. As shown in Tables 4 and 5, our framework consistently demonstrates superior or highly competitive alignment with the BigCodeBench gold standard.

While Arena-Hard-Auto (Code) provides a significant automated baseline based on prompt-response pairs, the correlation results suggest that the requirements extracted via Conv-to-Bench may better reflect the high-fidelity constraints needed to match expert-authored standards. For instance, our method achieves a Spearman correlation of $\rho = 1.000$ on the Full Set, exceeding the 0.943 achieved by Arena-Hard-Auto (Code) using the Gemini evaluator. These findings suggest that the systematic synthesis of instructions from a dialogue history potentially provides a more precise evaluative standard for models performing complex coding tasks.

6 Conclusion

To address the scalability bottleneck of manual expert verification, we introduced Conv-to-Bench, a framework that transforms multi-turn dialogues into structured requirement checklists. Our results demonstrate near-perfect alignment with established standards like BigCodeBench, though we found that instructions-only extraction currently provides a more stable signal than noise-prone user feedback.

Table 4: **Conv-to-Bench (Instructions-Only) demonstrates superior alignment with the BigCodeBench (Full Set) compared to the programming-specific subset of the Arena-Hard-Auto benchmark.** For our method, the Instructions-Only configuration is utilized due to its stronger performance in previous evaluations. A higher ρ (closer to 1.0) indicates a stronger correlation, while a low p-value (e.g., < 0.05) indicates statistical significance.

		BigCodeBench (Instruct)				BigCodeBench (Complete)			
		Spearman		Kendall		Spearman		Kendall	
Evaluator	Benchmark	ρ	p	τ	p	ρ	p	τ	p
Gemini-2.5-Flash	Conv-to-Bench (Ours)	**1.000**	<0.001	**1.000**	0.003	**0.943**	0.005	**0.867**	0.017
	Arena-Hard-Auto (Code)	0.943	0.005	0.867	0.017	0.886	0.019	0.733	0.056
GPT-5-Mini	Conv-to-Bench (Ours)	0.886	0.019	0.733	0.056	**0.943**	0.005	**0.867**	0.017
	Arena-Hard-Auto (Code)	0.829	0.042	0.733	0.056	0.771	0.072	0.600	0.136

Table 5: **Conv-to-Bench (Instructions-Only) maintain competitive rank alignment with the BigCodeBench (Hard Set), outperforming the programming-specific subset of the Arena-Hard-Auto benchmark.** This analysis employs the Instructions-Only method for Conv-to-Bench following its better results in earlier tests. A higher ρ (closer to 1.0) indicates a stronger correlation, while a low p-value (e.g., < 0.05) indicates statistical significance.

		BigCodeBench (Instruct)				BigCodeBench (Complete)			
		Spearman		Kendall		Spearman		Kendall	
Evaluator	Benchmark	ρ	p	τ	p	ρ	p	τ	p
Gemini-2.5-Flash	Conv-to-Bench (Ours)	**0.952**	<0.001	**0.857**	0.002	**0.994**	<0.001	**0.982**	0.001
	Arena-Hard-Auto (Code)	0.905	0.002	0.786	0.006	0.970	<0.001	0.909	0.002
GPT-5-Mini	Conv-to-Bench (Ours)	0.905	0.002	0.786	0.006	0.946	<0.001	0.837	0.004
	Arena-Hard-Auto (Code)	**0.952**	<0.001	**0.857**	0.002	0.922	0.001	0.837	0.004

Despite these results, this work has limitations. Our current analysis focused on the coding domain and the English language; exploring how these patterns hold across different languages remains an important next step. Additionally, accuracy remains contingent on the capabilities of the LLM judge, requiring a balance between model performance and resource costs.

Future work will focus on expanding this methodology to multi-lingual datasets and diverse technical fields to assess broader generalizability. Finally, developing methods to mitigate conversational noise will be essential to better leverage corrective feedback for higher-fidelity evaluations.

ACKNOWLEDGEMENTS

This work has been partially funded by the project AKCIT-Robotics: Immersive Environments Accelerating Robot Learning, with financial resources from the PPI IoT/Manufatura 4.0 / PPI HardwareBR of the MCTI grant number 057/2023, signed with EMBRAPII.

REFERENCES

Sher Badshah and Hassan Sajjad. Reference-guided verdict: LLMs-as-judges in automatic evaluation of free-form QA. In Chen Zhang, Emily Allaway, Hua Shen, Lesly Miculicich, Yinqiao Li, Meryem M'hamdi, Peerat Limkonchotiwat, Richard He Bai, Santosh T.y.s.s., Sophia Simeng Han, Surendrabikram Thapa, and Wiem Ben Rim (eds.), *Proceedings of the 9th Widening NLP Workshop*, pp. 251–267, Suzhou, China, November 2025. Association for Computational Lin-

guistics. ISBN 979-8-89176-351-7. doi: 10.18653/v1/2025.winlp-main.37. URL https://aclanthology.org/2025.winlp-main.37/.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

CodeGemma Team. Codegemma: Open code models based on gemma, 2024. URL https://arxiv.org/abs/2406.11409.

Or Don-Yehiya et al. Naturally occurring implicit feedback in human–llm dialogues. In *Proceedings of ACL*, 2024.

Bradley Efron and Robert J Tibshirani. *An Introduction to the Bootstrap*. CRC press, New York, 1994. ISBN 9780412042317.

Christopher A Field and Alan H Welsh. Bootstrapping clustered data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 69(3):369–390, 2007.

Gemini Team. Gemini: A family of highly capable multimodal models, 2025. URL https://arxiv.org/abs/2312.11805.

Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*, 2022.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024. URL https://arxiv.org/abs/2401.14196.

J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977. doi: 10.2307/2529310.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=KfTf9vFvSn.

Yuhan Liu, Michael JQ Zhang, and Eunsol Choi. Implicit user feedback in human-LLM dialogues: Informative to understand users yet noisy as a learning signal. In *2nd Workshop on Models of Human Feedback for AI Alignment*, 2025. URL https://openreview.net/forum?id=toSLK7ISiE.

OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Richard Yuanzhe Pang, Stephen Roller, Kyunghyun Cho, He He, and Jason Weston. Leveraging implicit feedback from deployment data in dialogue. In Yvette Graham and Matthew Purver (eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 60–75, St. Julian's, Malta, March 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.eacl-short.8. URL https://aclanthology.org/2024.eacl-short.8/.

Long Phan, Alice Gatti, Ziwen Han, et al. Humanity's last exam, 2025. URL https://arxiv.org/abs/2501.14249.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating LLMs with fewer examples. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=qAml3FpfhG.

Qwen. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL https://arxiv.org/abs/2308.12950.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatGPT interaction logs in the wild. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bl8u7ZRlbM.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. LMSYS-chat-1m: A large-scale real-world LLM conversation dataset. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=BOfDKxfwt0.

Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

7 APPENDIX

A PROMPTS

A.1 PROGRAMMING DOMAIN CLASSIFICATION

System Prompt

You are an expert in conversation analysis.
Your task is to determine if the following conversation between a user and an AI assistant is programming-related.
is_programming_related:

- true if the conversation is related to programming (code requests, debugging, code review, algorithm explanations, language snippets, etc.).
- false otherwise.

User Prompt

Determine if the following conversation is programming-related:
Conversation:
{first_message}

A.2 EXTRACT INSTRUCTION

System Prompt

You are an expert in conversation analysis.
Your goal is to identify whether a user made a request involving **writing or modifying code** within a conversation with an AI assistant.
Task
Analyze the conversation and extract the user's original **coding-related instruction**, following these guidelines:

- Guidelines for **instruction**:
  - Provide a clear, concise, and direct description of the user's original request that involves writing or modifying code.
  - If the user requested to modify an existing code, include the request itself and the exact code snippet that needs to be modified.
  - If the conversation **does not contain** any such coding request, set "instruction" to an **empty string** ("").

User Prompt

Analyze the following conversation and extract the user's coding-related instruction.
Conversation:
{conversation}

A.3 FILTER VALID INSTRUCTIONS

System Prompt

You are a validation system. Your job is to decide if a user's instruction is **valid** or **invalid**.
Definition An instruction is **valid** if it's clear and complete enough for an AI to give a reasonable answer.
Guidelines
Mark as INVALID if:

1. **It's missing essential information.**
   - The instruction refers to specific content that isn't there (e.g., "Summarize the following text:" but no text is provided).
2. **It's too vague or ambiguous.**
   - The instruction uses placeholders (like [insert_name]) or is too unclear to understand.

Mark as VALID if:

1. **It's self-contained.**
   - The AI can understand and respond using only the instruction itself.
2. **It's a general or abstract request.**
   - Instructions like "Explain how SQL works" or "Write a plan to build a website" are **valid** because they don't depend on missing files or previous context.

The Main Test
Ask yourself this: **Could an AI provide a good answer using *only* this instruction?**

- Yes → **valid**
- No → **invalid**

User Prompt

User Instruction: {instruction}

A.4 IDENTIFY FEEDBACK MESSAGES

System Prompt

You are a conversation analysis expert.
Your goal is to analyze dialogues between a user and an AI assistant to identify messages containing implicit or explicit user feedback on the assistant's responses.
Your output must adhere to the `FeedbackSchema`.
Feedback Definitions:

- **Positive Feedback (+):**
  - Occurs when the user confirms that a suggestion, code, or information provided by the assistant was successful, correct, or met their needs.
  - Includes expressions of gratitude that clearly refer to the usefulness of the previous answer.
- **Negative Feedback (-):**
  - Occurs when the user indicates that the assistant's response was unsatisfactory, incorrect, incomplete, or confusing.
  - Includes requests for repetition or clarification that suggest the previous answer failed.
  - Includes direct corrections made by the user to the assistant's code or information.

Crucial Rules:

1. **User-Focused:** Only messages with `'role: "user"'` can be classified as feedback. The feedback is always from the user *about* the assistant's response.
2. **Feedback is a Reaction:** A feedback message must be a reaction to one or more previous assistant messages. The first user message in a conversation is never feedback.
3. **Neutrality is the Default:** Messages that continue the conversation without evaluating the previous answer are **neutral** and must not be listed.
4. **Silence is NOT Positive:** The absence of a user response is **not** positive feedback. Positive feedback must explicitly acknowledge the usefulness or correctness of the previous answer.

User Prompt

Analyze the following conversation according to the defined rules and return the feedback messages.
Conversation: {conversation}

A.5 GENERATE CHECKLIST

System Prompt

You are an expert Quality Assurance (QA) analyst specializing in code verification.
Your expertise lies in translating user requirements and feedback into precise, testable criteria.
Task: Generate an Evaluation Checklist
Your task is to create a checklist of requirements that the code must satisfy, following these steps:

1. Analyze the **'Instruction'** provided by the user.
2. Analyze only the **feedback messages explicitly listed** as feedback sources.
   - Do **not** use any other user messages or context outside this list.
3. Synthesize a checklist based on both sources (Instruction and feedback messages).

If **no feedback messages are listed** (i.e., the list of feedback message IDs is empty), then the checklist **must be derived solely from the Instruction**.
Output Format and Rules

- The checklist must consist of **simple, unambiguous Yes/No questions**.
- Each question should test only **one atomic condition**.
- Preface each checklist item with its source:
  - `[I]` — derived from the Instruction.
  - `[Fn]` — derived from feedback message n (**where n must be one of the message IDs explicitly listed in the feedback list**).
- Ensure all output is formatted as a clear and readable list of checklist items.

User Prompt

Generate the evaluation checklist based on the following inputs:
Conversation: {conversation}
Instruction: {instruction}
Positive Feedback IDs: {positive_feedback_ids}
Negative Feedback IDs: {negative_feedback_ids}

A.6 EVALUATION

System Prompt

You are a strict, automated Quality Assurance (QA) Engine.
Your purpose is to evaluate code against a checklist with rigorous and objective precision.
Task
You will be given a **Code Snippet** and a **Checklist** of requirements.
Your responsibilities:

1. **Analyze the Code:** Read and understand the functionality, logic, and limitations of the provided code snippet.
2. **Evaluate Each Requirement:** For every requirement in the checklist, determine if the code explicitly fulfills it.
3. **Provide Boolean Answers:**
   - Return `true` if the code fully satisfies the requirement.
   - Return `false` if it does not.
   - Do **not** assume partial credit or make lenient judgments — if a requirement is not clearly met, it must be `false`.

Output Format

- **The order of the boolean answers must match the order of the checklist requirements exactly.**
- **Every checklist item must be evaluated** — do not omit or skip any entries.

User Prompt

Evaluate the following code according to the checklist provided.

1. **Code Snippet:** {code}
2. **Checklist:** {checklist}

B VERBOSITY-BIAS ANALYSIS

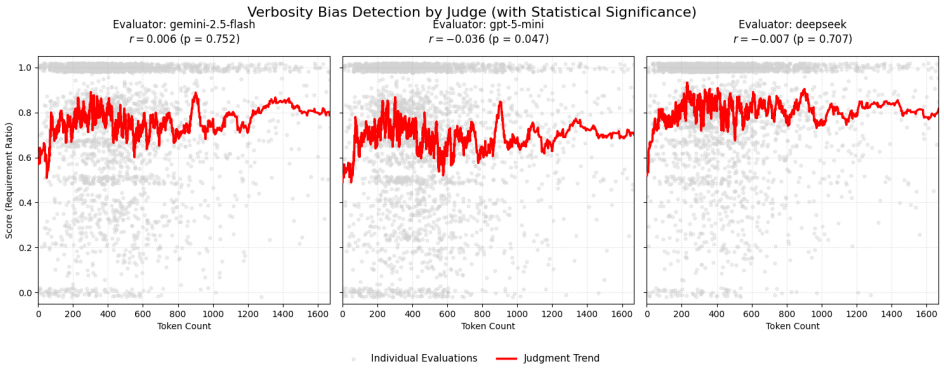

Figure 2: **Verbosity Bias Analysis.** This visualization assesses the relationship between response length (tokens) and normalized instruction scores across the LLM judges. While gray dots denote individual task evaluations, the red line tracks the moving average trend. The low Pearson correlation coefficients (r) and non-significant p-values indicate a lack of verbosity bias, confirming that the judges' scoring remains independent of response length.

C EVALUATED MODELS' SCORES ON BIGCODEBENCH

Table 6: **BigCodeBench performance metrics across model variants.** Detailed performance scores for the "Hard" and "Full" subsets. Results represent the percentage of tasks successfully completed in both 'Complete' and 'Instruct' settings. Missing values (–) indicate data not available in the official benchmark leaderboard.

	Hard Subset (%)			Full Subset (%)		
Model	Complete	Instruct	Average	Complete	Instruct	Average
GPT-4.1-Nano-2025-04-14	31.8	28.4	30.1	–	–	–
Gemini-2.0-Flash-001	33.8	23.6	28.7	–	–	–
Qwen2.5-Coder-7B-Instruct	20.3	20.3	20.3	48.8	40.4	44.6
CodeQwen1.5-7B-Chat	15.5	18.9	17.2	43.6	39.6	41.6
DeepSeek-coder-6.7B-Instruct	15.5	10.1	12.8	43.8	35.5	39.6
CodeGemma-7B-it	13.5	7.4	10.4	39.3	32.3	35.8
CodeLlama-13B-Instruct-hf	6.8	9.5	8.2	31.7	28.5	30.1
CodeLlama-7B-Instruct-hf	4.1	3.4	3.8	25.7	21.9	23.8

D COMPUTATIONAL COST AND EFFICIENCY

Table 7: **Token usage per full benchmark execution.** Total prompt and completion token counts required to evaluate a single model across the 387-item dataset. Lower token counts represent higher computational and cost efficiency for the evaluator framework.

	Mean Tokens (μ)	
Evaluator	Prompt	Completion
Gemini-2.5-Flash	1,168,581.38	193,097.38
GPT-5-Mini	1,227,954.25	674,392.25

E INSTRUCTION QUALITY FILTERING EXAMPLES

This section delineates the exclusion criteria applied during the Instruction Synthesis phase, providing specific examples of instructions that were discard by the LLM-based binary classifier.

Table 8: Samples of instructions discarded during quality filtering.

Category	**Instruction Example / Failure Reason**
Anonymization	*"Write a NAME_2 page using NAME_1.js for the front-end and FastAPI for the backend."* **Failure:** The de-identification process, performed by the original dataset providers, compromised technical semantics, rendering the instruction unintelligible and impossible to fulfill.
Ambiguity	*"Write me a python code for the best virtual assistant possible."* **Failure:** The subjectivity of the phrase "best possible" and the absence of specific functional requirements preclude a deterministic implementation.

F Instruction and Checklist Examples

This section illustrates the methodology used for generating validation checklists from raw instructions. We distinguish between cases where requirements are derived solely from the instruction (indicated by prefixes like [I]) and cases where feedback (indicated by prefixes like [F3]) are incorporated according to their presence on the original dialogue.

Example: SQL Query Planner

Instruction: Implement a toy query planner that converts SQL into a graph of relational algebra operations. Assume SQL is parsed into an Abstract Syntax Tree (AST) and only basic 'select' with columns and a 'where' clause needs to be implemented.
Generated Checklist:

- `[I]` Does the code implement a query planner?
- `[I]` Does the query planner convert SQL (via AST) into a graph?
- `[I]` Does the graph represent relational algebra operations?
- `[I]` Does the query planner accept an AST as input?
- `[I]` Does the query planner support basic SELECT statements and WHERE clauses?
- `[I]` Does the query planner explicitly exclude sorting and pagination?

Example: Ohm's Law Calculator

Instruction: Write a code in c plus plus that take voltage and current and give power and resistance.
Generated Checklist:

- `[I]` Is the code written in C++?
- `[I]` Does the code take voltage and current as inputs?
- `[I]` Does the code output power and resistance?
- `[F3]` Does the code provide a clear mechanism for inputting voltage?
- `[F3]` Does the code provide a clear mechanism for inputting current?

G Annotation Setup

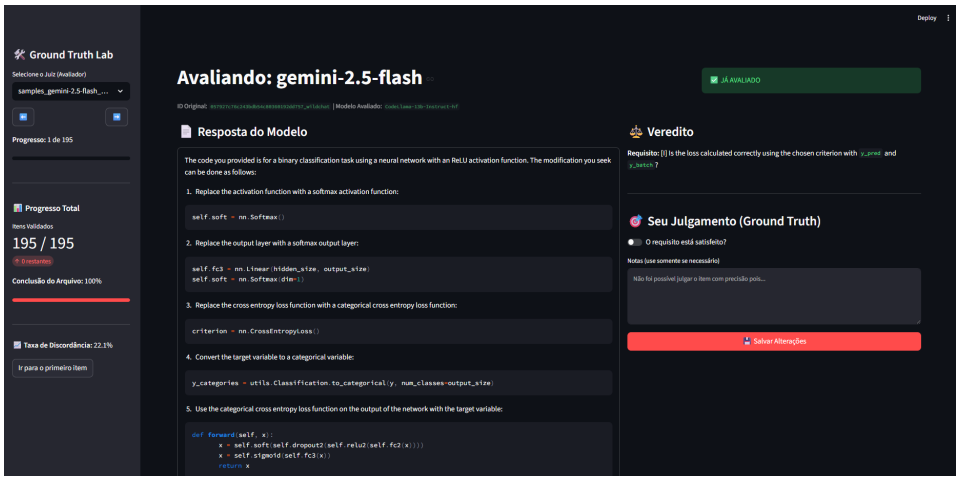

Figure 3: **Custom interface for gold-standard dataset validation.** Experts evaluated a balanced sample of outputs from eight models to mitigate architectural bias. The interface displays the model response alongside the specific requirement, where annotators provide a binary ground-truth label $y \in \{0, 1\}$ based on criterion fulfillment.