From Rankings to Insights: Evaluation Should Shift Focus from Leaderboard to Feedback

Zongqi Wang[1][*], **Tianle Gu**[1], **Chen Gong**[2], **Xin Tian**[3], **Siqi Bao**[3], **Yujiu Yang**[1][†]

[1] Tsinghua Shenzhen International Graduate School, Tsinghua University
[2] School of Cyber Engineering, Xidian University [3] Baidu, Inc

Abstract

Automatic evaluation benchmarks such as MT-Bench, Arena-Hard, and Auto-Arena are seeing growing adoption for the evaluation of Large Language Models (LLMs). Existing research has primarily focused on approximating human-based model rankings using limited data and LLM-as-a-Judge. However, the fundamental premise of these studies, which attempts to replicate human rankings, is flawed. Specifically, these benchmarks typically offer only overall scores, limiting their utility to leaderboard rankings, rather than providing feedback that can guide model optimization and support model profiling. Therefore, we advocate for an evaluation paradigm shift from approximating human-based model rankings to providing feedback with analytical value. To this end, we introduce **Feedbacker**, an evaluation framework that provides comprehensive and fine-grained results, thereby enabling thorough identification of a model's specific strengths and weaknesses. Such feedback not only supports the targeted optimization of the model but also enhances the understanding of its behavior. Feedbacker comprises three key components: an extensible tree-based query taxonomy builder, an automated query synthesis scheme, and a suite of visualization and analysis tools. Furthermore, we propose a novel LLM-as-a-Judge method: PC^2 (Pre-Comparison-derived Criteria) pointwise evaluation. This method derives evaluation criteria by pre-comparing the differences between several auxiliary responses, achieving the accuracy of pairwise evaluation while maintaining the time complexity of pointwise evaluation. Finally, leveraging the evaluation results of 17 mainstream LLMs, we demonstrate the usage of Feedbacker and highlight its effectiveness and potential. Our project homepage and dataset are available at https://liudan193.github.io/Feedbacker/.

1 Introduction

Evaluating large language models (LLMs) has become increasingly important as these models advance in various capabilities [1–14]. Among them, Chatbot Arena [14], which uses over 2.8 million human-labeled cases, provides relatively accurate evaluations for 223 LLMs (as of 2025-04-09). However, its labor-intensive nature limits its applicability, which is particularly pronounced during model development, where timely feedback is essential. These limitations have motivated researchers to explore automatic evaluation [15–19, 8], which uses a small amount of data and LLM-as-a-Judge [9, 20–23] to approximate the model rankings in Chatbot Arena. Automatic evaluation benchmarks not only accelerate leaderboard updates but also enable timely feedback for LLM development.

---

[*] Work done during internship at Baidu, Inc. Contact: <zq-wang24@mails.tsinghua.edu.cn>.

[†] Corresponding author: Yujiu Yang <yang.yujiu@sz.tsinghua.edu.cn>.
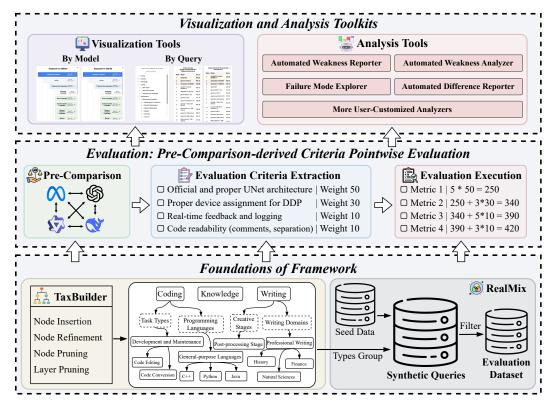
Figure 1: An overview of Feedbacker framework.

Challenge 1. However, although automatic evaluations can provide timely feedback, they are often insufficient to guide model optimization or profiling. This is primarily due to the fundamental premise of approximating final Chatbot Arena rankings, which introduces several critical limitations. First, to fit given rankings with limited data, these benchmarks typically employ a restricted query set that excessively prioritizes mainstream tasks. This narrow scope neglects long-tail queries reflective of real-world use cases [15–19, 8], thereby overlooking some critical aspects of model capabilities. Second, this premise lacks the requirement for fine-grained taxonomic annotations, which restricts these benchmarks to generating only overall scores for rankings. This limitation prevents identifying specific weaknesses and strengths, limiting the benchmarks' analytical value.

Solution 1. To tackle these limitations and make automatic evaluation more suitable for generating feedback, we introduce Feedbacker, an extensible evaluation framework built on TaxBuilder and RealMix. TaxBuilder constructs a query-type taxonomy, which is crucial for providing a solid foundation for comprehensive and fine-grained evaluation. The taxonomy includes first-level nodes representing domains (e.g., writing, coding), second-level nodes for classification principles (e.g., writing style, task type), and third-level nodes and beyond for fine-grained tags. TaxBuilder is extensible, offering features including node insertion, refinement, pruning, and layer pruning to support future expansion. RealMix generates high-quality and realistic queries from a set of seed data, simulating real users, preventing data contamination, and further enhancing extensibility.

Challenge 2. Compared to existing benchmarks, our evaluation dataset presents a unique challenge. Since we aim for a comprehensive evaluation, the overall dataset size is relatively large, yet the number of samples per specific query type remains small to allow for fine-grained analysis. This characteristic poses a significant challenge for existing LLM-as-a-Judge paradigms. Evaluation paradigms typically fall into three categories: pairwise evaluation [19], pointwise evaluation [24, 25], and win rate based on pairwise evaluation [16, 2, 26, 18]. In our evaluation context, pairwise evaluation is impractical due to its quadratic growth in time cost as the number of models increases, despite offering superior accuracy [15]. Pointwise evaluation is more efficient but sacrifices accuracy [15, 4, 27]. Additionally, the win rate metric demands a relatively large dataset to ensure reliable results [16]. Therefore, we pose a critical question: *Can an LLM-as-a-Judge method achieve the accuracy of pairwise evaluation, the efficiency of pointwise evaluation, and reliability with limited data?*

Solution 2. To address this question, we propose PC^2 pointwise evaluation. It begins by generating query-specific criteria, which are then used to assess responses. Its core innovation lies in leveraging responses generated by a diverse set of models and using the differences among these responses to establish evaluation criteria. Since the criteria for each query can be fixed, they can be reused when evaluating new models, achieving an evaluation time similar to pointwise evaluation.

In summary, our main contributions are four-fold:

• We propose shifting evaluation paradigm from rankings to feedback. To this end, we introduce Feedbacker, which employs comprehensive and fine-grained queries to identify specific strengths and weaknesses. The feedback generated by Feedbacker can facilitate model optimization and profiling.

• Utilizing Feedbacker, we develop the initial version of the taxonomy and evaluation dataset for six domains: writing, roleplay, knowledge, coding, mathematics, and reasoning. These are denoted as Feedbacker-T-V0 (Fig. 2) and Feedbacker-D-V0 (Tab. 1), respectively.

• We introduce PC^2 pointwise evaluation, which achieves higher accuracy than pairwise evaluation, maintains efficiency of pointwise evaluation, and ensures reliability with limited evaluation dataset.

• To showcase the usage of Feedbacker, we evaluate 17 mainstream LLMs and conduct in-depth analyses of the results. By uncovering several fine-grained insights, we demonstrate the effectiveness and potential of Feedbacker.

2 Feedbacker

The overview of Feedbacker is shown in Fig. 1. It consists of three components: TaxBuilder for building an extensible query taxonomy, RealMix for generating new queries based on the taxonomy, and visualization and analysis toolkits for user-friendly interaction.

2.1 TaxBuilder: Automatic Tree-based Taxonomy Building

To establish a comprehensive and fine-grained evaluation system, we first need to develop the query taxonomy. To achieve this, we introduce TaxBuilder, an automatic and extensible solution for constructing tree-based taxonomy from large volumes of unstructured queries. The workflow of TaxBuilder is illustrated in Fig. 3.

Preparation. TaxBuilder initiates the process with a manually constructed basic taxonomy tree T_{init}. This basic tree is simple and requires minimal human effort (refer to § C.1.1 for details). Then, we employ a low-cost model (gpt-4o [28]) to annotate query tags for a large set of real user queries [3]. The primary aim of this phase is to generate a large pool of unstructured query tags, named TG_{init}. Further details regarding this step are elaborated in § C.1.2.

Node Insertion. The most core design of TaxBuilder is its node insertion mechanism. A naive approach would be to input the entire taxonomy as context to a powerful LLM and ask it
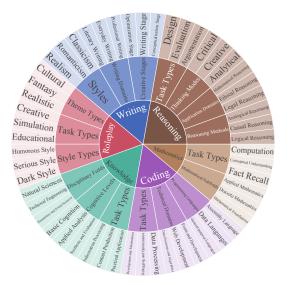

Figure 2: A subset of Feedbacker-T-V0. Please refer to project page for complete taxonomy.

to identify the position for the new node. However, this task involves both long-context and complex reasoning, which is challenging even for the most advanced LLMs. Moreover, as the taxonomy grows, the input context becomes increasingly longer and the reasoning more difficult, thereby limiting the extensibility of this approach.

---

[3]We utilize the dataset made available by the Chatbot Arena team, accessible at `https://www.kaggle.com/competitions/wsdm-cup-multilingual-chatbot-arena/data`.
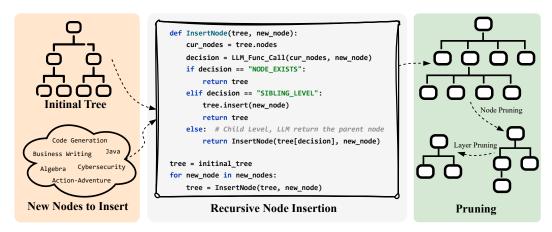
Figure 3: An overview of TaxBuilder, a tree-based automatic taxonomy generation.

To address this issue, we introduce a recursive node insertion process inspired by how trees are constructed in data structures. When a new node needs to be inserted, we traverse the tree step by step. At each step, the LLM only needs to determine the relationship between the new node and the nodes at the current level. In this way, the model only considers a small portion of the tree and reasons over three options: (1) the node already exists, (2) it is a sibling of the current nodes, or (3) it should be added as a child of one of them. This design keeps the context short and the reasoning simple, making TaxBuilder significantly more scalable as the taxonomy expands.

Formally, given a node tg and the current tree T_{cur} (we represent the tree by its root), we invoke LLM-as-Decision-Maker, denoted as DM_{ins}, as follows:

$$DM_{ins}(tg, T_{cur}^c) \rightarrow \{\text{<E>}, \text{<S>}, T_{cur}^{c_i}\}, \tag{1}$$

where $T_{cur}^c = \{T_{cur}^{c_1}, T_{cur}^{c_2}, \ldots, T_{cur}^{c_m}\}$ denotes the set of first-level child nodes of T_{cur}. The function of DM_{ins} is to determine the relationship between tg and T_{cur}^c. The prompt for DM_{ins} is presented in Fig. 15. It will return one of three possible relationship types:

- **<E> (Exists):** This indicates that the node tg already exists in T_{cur}^c or has a semantically equivalent node. In this case, the new node n is discarded to avoid redundancy.
- **<S> (Sibling):** This signifies that tg is a sibling of the nodes T_{cur}^c. Consequently, T_{cur}^c is updated to include tg, i.e., $T_{cur}^c = \{T_{cur}^{c_1}, T_{cur}^{c_2}, \ldots, T_{cur}^{c_m}\} \cup \{tg\}$.
- **$T_{cur}^{c_i}$ (Child Node):** This implies that tg should be a child of the specific node $T_{cur}^{c_i}$. In this scenario, a recursive insertion is performed.

After recursively traversing the entire tree, we successfully insert the node tg and obtain the tree for the next iteration. We iteratively perform this operation for all nodes, and ultimately obtain a tree that contains all tags from TG_{init}.

Node Refinement and Pruning. Node refinement serves two purposes: (1) resolving incorrect parent-child relationships between nodes at the same level, and (2) splitting leaf nodes that conflate multiple concepts into separate, more atomic concepts (e.g., separating "science fiction fanfiction" into "science fiction" and "fanfiction"). Node pruning addresses two issues: (1) merging duplicate nodes at the same level, and (2) removing rarely encountered concepts that unnecessarily complicate the tree. We perform the refinement and pruning processes recursively, guided by the LLM-as-Decision-Maker (see Fig. 16), which effectively corrects faulty nodes and controls node complexity.

Layer Pruning. We introduce two rules to reduce tree depth: (1) Meaningless depth pruning: if a parent node has only a single child, the child is pruned to eliminate unnecessary depth; (2) Complexity pruning: the tree is restricted to a maximum of four layers. Nodes beyond the fourth layer are retained only if they have five or more child nodes, as such structures are usually meaningful. which effectively controls tree complexity and prevents overgrowth.

After completing node insertion, node refinement, node pruning, and layer pruning, we obtain the final taxonomy, which we call Feedbacker-T-V0. The current taxonomy tree is easy to interpret by

human, allowing for convenient manual modification if needed. A simplified version of this tree is shown in Fig. 2. The formal algorithm for the tree construction process is provided in Alg. 1.

2.2 RealMix: New Query Synthesis

We opt for synthetic data over existing datasets for two reasons: (1) existing datasets may not contain enough queries with the required tags in taxonomy, and (2) to prevent data contamination. To ensure the evaluation data includes high-quality, realistic queries that can be encountered in real-world, we propose RealMix. RealMix operates by mixing several appropriate content details from multiple real-user queries and adapting them to align with the specified tags. While there are other data synthesis approaches [29, 30] that mix multiple queries, none are capable of generating queries with specified tags. The core advantage of RealMix in fact lies in that it is built upon the TaxBuilder, which enables controlled and tag-aligned synthesis.

Seed Data. We first use QwQ-32b [31, 32] to annotate a set of real-user queries [33] with three types of labels: domain, query tags, and quality. The domain label covers six domains: writing, roleplay, knowledge, coding, mathematics, and reasoning. Query tags are assigned based on Feedbacker-T-V0. For quality annotation, each domain is paired with a checklist of seven criteria; queries meeting at least four are marked as high-quality. In total, we collect approximately 31,000 high-quality annotated queries. Further details are provided in § C.2.1.

Query Synthesis. RealMix leverages content from real user queries to generate queries aligned with specific tags. This approach allows us to obtain realistic queries associated with desired tags. To synthesize a query, we sample one reference query and three content queries. A query generation model then extracts appropriate real-world content from the content queries and generates a new query that incorporates these contents while matching the tags of the reference query. If no suitable content is found for the specified tags, this generation is discarded. We adopt reference queries instead of random tag combinations from Feedbacker-T-V0 since some combinations do not exist in practice. To mitigate model bias, we employ multiple query generation models (deepseek-v3 [34], gpt-4o, and doubao-1.5pro [35]). More details are provided in § C.2.2.

Verification and Filtration. We validate both the tags and the quality of each generated query using the same procedure applied to the seed data. Then, we discard the low-quality queries. To further reduce model bias, for each query, we randomly sample checking models for quality and tags from three reasoning models (QwQ-32B, deepseek-r1 [36], and o1-mini [37]).

2.3 Visualization and Analysis Toolkits

We provide a suite of visualization and analysis tools to assist users in understanding and interpreting evaluation results. Due to space constraints, the description of each tool is provided in § C.3.

3 Dataset Description

Dataset Statistics. We construct Feedbacker-D-V0, the initial version of the evaluation dataset, which consists of 3,343 queries, each annotated with a domain and tags. Tab. 1 summarizes the key statistics of Feedbacker-D-V0. We ensure that the sample size for each tag in Feedbacker-T-V0 is at least 19 (#MinNum) to guarantee the statistical representativeness of the evaluation results.

Table 1: Statistics of Feedbacker-D-V0.

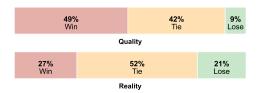| Domain	#Samples	#Tags	#MinNum	Avg.Length
Writing	1108	594	19	772.57
Roleplay	470	429	19	1008.46
Knowledge	540	315	20	608.57
Coding	636	369	19	1232.03
Mathematics	344	189	20	817.02
Reasoning	245	186	19	904.81
Total	3343	2082	19	880.92

Figure 4: Comparison of RealMix and real user queries, judged by five human evaluators.

Quality Control. We validate data quality through three tasks involving human and automatic evaluation: Five graduate-level annotators perform two comparative assessments. For each generated

query and its corresponding real user query (i.e., reference query), annotators evaluate (1) which query is higher in quality and (2) which is more likely to occur in real-world scenarios. Results in Fig. 4 show generated queries are consistently preferred, confirming the effectiveness of RealMix. Details are in § C.4.1. (3) We assess potential data leakage in generated queries in § C.4.2.

4 Pre-Comparison-derived Criteria Pointwise Evaluation

In this section, we first formulate traditional evaluation paradigms and highlight their strengths and limitations in Sec. 4.1. Then, we present the motivation and description of our method in Sec. 4.2.

4.1 Problem Formulation

Pairwise Evaluation. Pairwise evaluation [38, 39] refers to comparing two responses to determine which is better. Given two LLM responses y^A and y^B for a given instruction x, a judge model J is tasked to provide a comparative assessment with a pairwise evaluation prompt p^I.

$$J(y^A, y^B | x, p_i) \rightarrow \{A > B, A = B, A < B\}. \tag{2}$$

Pairwise evaluation allows for capturing relative performance, which is often more reliable than pointwise evaluation. However, as the number of models evaluated increases, the time consumption grows significantly [15, 4], with a quadratic complexity of O(n^2).

Vanilla Pointwise Evaluation. Vanilla pointwise evaluation [40, 41] refers to scoring each response independently. The judge model J is tasked to assign a score to a single response with a vanilla pointwise evaluation prompt p_o. Typically, a baseline response y_b is used to help stabilize evaluation standards.

$$J(y | x, p_o, y_b) \rightarrow s \in \mathbb{R}. \tag{3}$$

Pointwise evaluation is straightforward and does not incur unacceptable evaluation costs as the number of models being evaluated increases. However, it often produces inferior evaluation results compared to pairwise evaluation [4, 15].

4.2 PC^2-Pointwise Evaluation

Motivation. Our core insight is that, given the superior performance of pairwise evaluation over pointwise evaluation, we can adapt the mechanism of pairwise evaluation into the pointwise setting to create an enhanced pointwise evaluation. Pairwise evaluation excels in comparing two responses by identifying their differences, evaluating which response performs better with respect to each difference, and ultimately selecting the winner based on advantages in the most important differences. (See examples in § D.) In contrast, pointwise evaluation struggles to identify such differences because it evaluates each response independently. To bridge this gap, we introduce a pre-comparison phase, where we leverage several auxiliary responses to extract relevant evaluation criteria before scoring. These criteria act as proxies for the key differences identified in pairwise evaluation. At the same time, we prompt the model to assign weights to each criterion, emulating how pairwise evaluation prioritizes more significant differences.
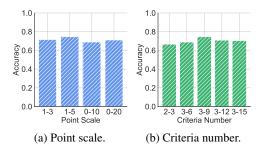
Pre-Comparison-derived Criteria Extraction. The core design of PC^2 pointwise evaluation lies in extracting criteria from a pre-comparison process. Specifically, given an instruction x, we prompt n LLMs $\{M_1, M_2, ..., M_n\}$ to produce n responses $\{y^1, y^2, ..., y^n\}$. The selection of multiple LLMs is intended to diversify the responses, thereby facilitating the extraction of comprehensive, diverse and effective criteria. Subsequently, we assign the judge model J two tasks. The first task involves using J to compare the n responses, thereby surfacing their differences (i.e., criteria). This is done using prompt p^c. The second task is to weight these criteria using prompt p_w:

$$J(\{y^1, y^2, ..., y^n\} \mid x, p_c, p_w) \rightarrow C = \{c_1, c_2, ..., c_m\}, W = \{w_1, w_2, ..., w_m\}$$

where C represents the criteria, and W denotes the corresponding weights, where $\sum_{i=1}^{m} w_i = 100$. Since we use a reasoning LLM as the judge model, we typically assign two tasks to the model simultaneously. If the judge model is a chat model, we recommend handling these tasks separately.

Table 2: Comparison of accuracy across different LLM-as-a-Judge methods.

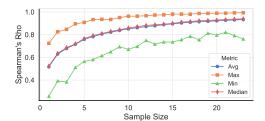| Method	Overall	Writing	Roleplay	Knowledge	Mathematics	Coding	Reasoning
Pairwise Evaluation	0.8176	0.8000	0.8302	0.8716	0.8750	0.8160	0.7672
Vanilla Pointwise Evaluation	0.6116	0.5838	0.7358	0.5229	0.7708	0.6960	0.5259
+ Baseline Answer	0.6525	0.6054	0.6038	0.6881	0.7500	0.6800	0.6466
NCD Pointwise Evaluation (No Pre-Com)	0.6688	0.6378	0.7358	0.6389	0.7917	0.7097	0.6207
PC^2 Pointwise Evaluation (Single Model Pre-Com)	0.6792	0.6432	0.7358	0.6697	0.7500	0.6880	0.6810
+ Diverse Model Pre-Com	0.7453	0.7351	0.8491	0.7248	0.7500	0.7440	0.7328
+ Baseline Answer	0.8318	0.7892	0.7925	0.8257	0.8542	0.8720	0.8707

(a) Point scale. (b) Criteria number.

Figure 5: Ablation study.

Figure 6: Consistency of rankings across 17 models with varying sample sizes.

Evaluation Execution. Once the criteria are determined, we can score a single response y:

$$J(y|x, p_c, y_b, C, W) \rightarrow s \in \mathbb{R}, \tag{4}$$

where p_c represents our criteria-weighted evaluation prompt and s is the final score.

This approach leverages the comprehensive understanding gained through pre-comparison to establish query-specific criteria, enabling more effective evaluation while maintaining computational efficiency as the number of evaluated models increases.

5 Results on PC^2 Pointwise Evaluation

In this section, we conduct the experiments on different LLM-as-a-Judge methods. The experiment details are shown in § B.1.

5.1 Main Results

Main Results. As demonstrated in Tab. 2, we can observe the following: (1) Base pairwise evaluation yields superior performance than vanilla pointwise evaluation, which is consistent with previous study [15]; (2) Vanilla pointwise evaluation performs poorly, but introducing a baseline answer significantly improves accuracy. (3) The naive criteria decomposition (NCD) pointwise evaluation outperforms pointwise evaluation, indicating that performing criterion decomposition alone can significantly enhance performance. (4) Introducing a single model for pre-comparison yields only a marginal improvement in performance compared to NCD, indicating that pre-comparison significantly enhances the quality of the extracted metrics. (5) When diverse models are adopted for pre-comparison, there is a significant improvement, especially in open-domain tasks such as writing and roleplay. (6) Further incorporating baseline answer leads to additional significant improvement, even surpassing the pairwise evaluation. Results on more benchmarks are shown in § A.1.

Ablation Study. As shown in Fig. 5, we observe slight fluctuations in performance across different point scales. Notably, the model demonstrates superior performance when the point scale is smaller. This is intuitive, as multiple evaluations tend to yield more stable scores with a small point scale. As for number of criteria, too few (e.g., 2-3) or too many (e.g., 3-15) criteria can both adversely affect performance. Based on these findings, we use a 1-5 point scale and 3-9 criteria in all experiments.

Determining the Minimum Meaningful Sample Size. If an evaluation dataset has only one sample for the HTML category, can it reliably indicate which models perform well or poorly in this category? Obviously, different samples can lead to inconsistent evaluation results. This raises an important
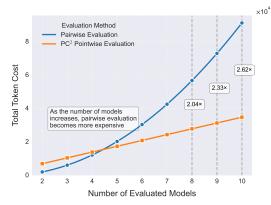
Figure 7: Token cost comparison.

Table 3: Agreement rate on the winner between two similarly performing models with a small data size.

Model Pair	Avg.	Min.	Std. Dev.
qwen-max-2024-10-15 vs. doubao-1-5-pro-32k-250115			
ours	81.52%	45.00%	0.1264
win rate	52.88%	15.00%	0.1586
Meta-Llama-3.1-70B-Instruct vs. Qwen2.5-72B-Instruct			
ours	89.39%	55.00%	0.1260
win rate	70.76%	5.00%	0.1875
Meta-Llama-3.1-8B-Instruct vs. Qwen2.5-7B-Instruct			
ours	81.67%	35.00%	0.1695
win rate	49.24%	20.00%	0.1801

question: *What is the minimum number of samples needed per query type for consistent evaluation?* To address this, we analyze tags with over 32 samples. For each tag, we randomly select N samples 20 times, rank 17 LLMs based on each sample set, and measure consistency using the average Spearman's rho coefficient across the 20 rankings. We then average the consistency scores across all tags. As shown in Fig. 6, consistency improves with larger sample sizes. Our Feedbacker-D-V0 dataset contains at least 19 samples per tag, consistently achieving a score above 0.9, which can be considered reliable.

5.2 Comparison with Other Evaluation Methods

Comparison with Pairwise Evaluation. We compare output token cost with pairwise evaluation and our method. From the results in Fig. 7, we observe that as the number of models increases, the token cost burden of pairwise evaluation becomes significant, whereas our method exhibits linear growth. This indicates that our method is more scalable for comparing a larger number of LLMs.

Comparison with Win Rate. To assess the effectiveness of the win rate metric under limited data conditions, we select pairs of models with similar performance and focus on query types over 100 instances. For each query type, we use the evaluation results from all available data as the ground-truth winner. We then perform 20 random samplings, each consisting of 19 instances, and compare the winner obtained from the sampled instances with the ground-truth winner. The average agreement rate is calculated across all query types and samplings. As shown in Tab. 3, the win rate metric performs poorly with small datasets, highlighting its limitations when data is scarce and model differences are subtle. In contrast, our evaluation method provides a more consistent evaluation.

6 Results on Feedbacker

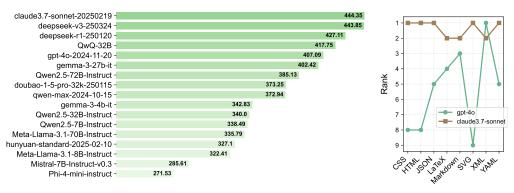### 6.1 A Quick Overview of the Evaluation Results.

To utilize Feedbacker for analysis, we first employ Feedbacker-D-V0 to evaluate 17 mainstream LLMs in this section. The experimental setup is provided in § B.3. An overview of evaluation results is presented in § B.4. For complete results, please refer to the project homepage.

6.2 Exploring Feedbacker: Practical Usage Examples

In this section, we showcase the usage of Feedbacker to analyze an LLM's strengths and weaknesses by several examples. These examples demonstrate the effectiveness and potential of Feedbacker.

To begin with, the *Automatic Weakness Reporter* reveals an interesting anomaly: `gpt-4o-2024-11-20` ranked 1st under the fine-grained query tag `root.coding.Programming Languages.Markup Languages.XML`, while its overall ranking in Tab. 5 is 6th. This suggests an unusual strength in that specific category.

To further explore this phenomenon, we examine the XML's parent node `root.coding.Programming Languages.Markup Languages` using the *Visualization Tool* in the "By Query" view, which dis-

(a) Rankings of 17 LLMs under `Markup Languages`.

(b) Fine-grained rankings.

Figure 8: Model rankings for `root.coding.Programming Languages.Markup Languages`.

plays model rankings across its subdomains. As shown in Fig. 8a, `claude3.7-sonnet-20250219` ranks 1st overall, while `gpt-4o-2024-11-20` ranks 5th. This result indicates that `gpt-4o-2024-11-20` does not perform as strongly across the entire `Markup Languages` category—its lead in XML may be an outlier or the result of targeted optimization.

We then adopt the *Failure Mode Explorer* to get the results in Fig. 8b, which reveals a notable distinction: `claude3.7-sonnet-20250219` maintains consistent high performance across all subdomains (low standard deviation), whereas `gpt-4o-2024-11-20`'s strong ranking is mainly driven by its exceptional performance in XML. Its scores in other subdomains are relatively low. This indicates that `gpt-4o-2024-11-20` may have been specially optimized for XML tasks, but does not exhibit a broad advantage across the entire `Markup Languages` domain. In contrast, `claude3.7-sonnet-20250219` demonstrates stable, and well-rounded performance. This not only exposes a weakness in `gpt-4o-2024-11-20`, but also suggests a need for broader optimization.

7 Related Work

Automatic Evaluation Benchmark. We review existing benchmarks here, focusing on their motivations, dataset designs, and evaluation methods. AlpacaEval-2.0 [16] uses 805 curated general-domain data points to assess instruction-following with win rate metric. MT-Bench [15] includes 80 high-quality queries across 8 domains. It adopts pointwise evaluation on a 1–10 scale. Arena-Hard [18] filters 500 challenging Chatbot Arena datasets and uses win rate metric for evaluation. LiveBench [3] manually updates queries in real-time with rule-based scoring to reduce LLM biases. Auto-Arena [19] employs pairwise debates for Elo-based ranking, with a 40-query evaluation set. WildBench [2] updates real user queries to reflect real-world distributions, using both win rate and pointwise evaluation. We discuss the differences between our methods and existing studies in § A.1.

8 Conclusion

We propose shifting the focus of evaluation from only rankings and leaderboard to providing valueable feedback. To this end, we introduce Feedbacker, an extensible evaluation framework featuring comprehensive and fine-grained queries, along with tools for visualization and analysis. Our proposed PC^2 pointwise evaluation achieves superior accuracy compared to pairwise evaluation while retaining pointwise efficiency. This approach effectively balances precision and speed, offering a practical solution for LLM evaluation. We demonstrate that utilizing Feedbacker enables the convenient identification of a model's strengths and weaknesses, thereby providing valuable feedback.

While our work employs comprehensive, fine-grained queries to deliver feedback, this approach is not the sole option. We believe Feedbacker can serve as a starting point to inspire the community to develop diverse, impactful feedback mechanisms, enhancing the analytical value of evaluations.

We conduct more in-depth discussions including (1) the implications for future research in § A.2, and (2) the limitations of our work in § A.3.

9

References

[1] Yidong Wang, Zhuohao Yu, Zhengran Zeng, Linyi Yang, Cunxiang Wang, Hao Chen, Chaoya Jiang, Rui Xie, Jindong Wang, Xing Xie, et al. Pandalm: An automatic evaluation benchmark for llm instruction tuning optimization. *arXiv preprint arXiv:2306.05087*, 2023.

[2] Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Faeze Brahman, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. Wildbench: Benchmarking llms with challenging tasks from real users in the wild. *arXiv preprint arXiv:2406.04770*, 2024.

[3] Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, et al. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

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

[5] Yanyang Li, Tin Long Wong, Cheung To Hung, Jianqiao Zhao, Duo Zheng, Ka Wai Liu, Michael R Lyu, and Liwei Wang. C2leva: Toward comprehensive and contamination-free language model evaluation. *arXiv preprint arXiv:2412.04947*, 2024.

[6] Seungone Kim, Juyoung Suk, Ji Yong Cho, Shayne Longpre, Chaeeun Kim, Dongkeun Yoon, Guijin Son, Yejin Cho, Sheikh Shafayat, Jinheon Baek, Sue Hyun Park, Hyeonbin Hwang, Jinkyung Jo, Hyowon Cho, Haebin Shin, Seongyun Lee, Hanseok Oh, Noah Lee, Namgyu Ho, Se June Joo, Miyoung Ko, Yoonjoo Lee, Hyungjoo Chae, Jamin Shin, Joel Jang, Seonghyeon Ye, Bill Yuchen Lin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. The BiGGen bench: A principled benchmark for fine-grained evaluation of language models with language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5877–5919, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

[7] Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. Infobench: Evaluating instruction following ability in large language models. *arXiv preprint arXiv:2401.03601*, 2024.

[8] Xiao Liu, Xuanyu Lei, Shengyuan Wang, Yue Huang, Andrew Feng, Bosi Wen, Jiale Cheng, Pei Ke, Yifan Xu, Weng Lam Tam, et al. Alignbench: Benchmarking chinese alignment of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11621–11640, 2024.

[9] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.

[10] Gayathri Saranathan, Mahammad Parwez Alam, James Lim, Suparna Bhattacharya, Soon Yee Wong, Martin Foltin, and Cong Xu. Dele: Data efficient llm evaluation. In *ICLR 2024 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2024.

[11] Eliya Habba, Ofir Arviv, Itay Itzhak, Yotam Perlitz, Elron Bandel, Leshem Choshen, Michal Shmueli-Scheuer, and Gabriel Stanovsky. Dove: A large-scale multi-dimensional predictions dataset towards meaningful llm evaluation. *arXiv preprint arXiv:2503.01622*, 2025.

[12] Tempest A van Schaik and Brittany Pugh. A field guide to automatic evaluation of llm-generated summaries. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2832–2836, 2024.

[13] Jiaxuan You, Mingjie Liu, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Llm-evolve: Evaluation for llm's evolving capability on benchmarks. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16937–16942, 2024.

[14] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

[15] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

[16] Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

[17] Jinjie Ni, Fuzhao Xue, Xiang Yue, Yuntian Deng, Mahir Shah, Kabir Jain, Graham Neubig, and Yang You. Mixeval: Deriving wisdom of the crowd from llm benchmark mixtures. *arXiv e-prints*, pages arXiv–2406, 2024.

[18] 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. *arXiv preprint arXiv:2406.11939*, 2024.

[19] Ruochen Zhao, Wenxuan Zhang, Yew Ken Chia, Weiwen Xu, Deli Zhao, and Lidong Bing. Auto-arena: Automating llm evaluations with agent peer battles and committee discussions. *arXiv preprint arXiv:2405.20267*, 2024.

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

[21] Zahra Ashktorab, Michael Desmond, Qian Pan, James M Johnson, Martin Santillan Cooper, Elizabeth M Daly, Rahul Nair, Tejaswini Pedapati, Swapnaja Achintalwar, and Werner Geyer. Aligning human and llm judgments: Insights from evalassist on task-specific evaluations and ai-assisted assessment strategy preferences. *arXiv preprint arXiv:2410.00873*, 2024.

[22] Yu-Min Tseng, Wei-Lin Chen, Chung-Chi Chen, and Hsin-Hsi Chen. Are expert-level language models expert-level annotators? *arXiv preprint arXiv:2410.03254*, 2024.

[23] Guijin Son, Hyunwoo Ko, Hoyoung Lee, Yewon Kim, and Seunghyeok Hong. Llm-as-a-judge & reward model: What they can and cannot do. *arXiv preprint arXiv:2409.11239*, 2024.

[24] Jiahao Ying, Yixin Cao, Yushi Bai, Qianru Sun, Bo Wang, Wei Tang, Zhaojun Ding, Yizhe Yang, Xuanjing Huang, and Shuicheng Yan. Automating dataset updates towards reliable and timely evaluation of large language models. *arXiv preprint arXiv:2402.11894*, 2024.

[25] Yuxuan Liu, Tianchi Yang, Shaohan Huang, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. Hd-eval: Aligning large language model evaluators through hierarchical criteria decomposition. *arXiv preprint arXiv:2402.15754*, 2024.

[26] Jian Yang, Jiaxi Yang, Ke Jin, Yibo Miao, Lei Zhang, Liqun Yang, Zeyu Cui, Yichang Zhang, Binyuan Hui, and Junyang Lin. Evaluating and aligning codellms on human preference. *arXiv preprint arXiv:2412.05210*, 2024.

[27] Mingqi Gao, Yixin Liu, Xinyu Hu, Xiaojun Wan, Jonathan Bragg, and Arman Cohan. Re-evaluating automatic llm system ranking for alignment with human preference. *arXiv preprint arXiv:2501.00560*, 2024.

[28] OpenAI. Hello gpt-4o. `https://openai.com/index/hello-gpt-4o/`, 2024. Accessed: 2025-04-30.

[29] Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. Regmix: Data mixture as regression for language model pre-training. *arXiv preprint arXiv:2407.01492*, 2024.

[30] Fengze Liu, Weidong Zhou, Binbin Liu, Zhimiao Yu, Yifan Zhang, Haobin Lin, Yifeng Yu, Xiaohuan Zhou, Taifeng Wang, and Yong Cao. Quadmix: Quality-diversity balanced data selection for efficient llm pretraining. *arXiv preprint arXiv:2504.16511*, 2025.

[31] Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, March 2025.

[32] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[33] Wei lin Chiang, Evan Frick, Lisa Dunlap, Anastasios Angelopoulos, Joseph E. Gonzalez, Ion Stoica, Sohier Dane, Maggie Demkin, and Nate Keating. Wsdm cup - multilingual chatbot arena. `https://kaggle.com/competitions/wsdm-cup-multilingual-chatbot-arena`, 2024. Kaggle.

[34] DeepSeek-AI. Deepseek-v3 technical report, 2024.

[35] ByteDance. Doubao 1.5 pro. `https://seed.bytedance.com/en/special/doubao_1_5_pro/`, 2024. Accessed: 2025-04-30.

[36] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[37] OpenAI. Openai o1 mini: Advancing cost-efficient reasoning. `https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/`, 2024. Accessed: 2025-04-30.

[38] Maosong Cao, Alexander Lam, Haodong Duan, Hongwei Liu, Songyang Zhang, and Kai Chen. Compassjudger-1: All-in-one judge model helps model evaluation and evolution. *arXiv preprint arXiv:2410.16256*, 2024.

[39] Zhengyu Hu, Linxin Song, Jieyu Zhang, Zheyuan Xiao, Jingang Wang, Zhenyu Chen, Jieyu Zhao, and Hui Xiong. Rethinking llm-based preference evaluation. *arXiv e-prints*, pages arXiv–2407, 2024.

[40] Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, et al. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*, 2023.

[41] Hai Ye and Hwee Tou Ng. Self-judge: Selective instruction following with alignment self-evaluation. *arXiv preprint arXiv:2409.00935*, 2024.

[42] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

[43] Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*, 2024.

[44] Yanyang Li, Jianqiao Zhao, Duo Zheng, Zi-Yuan Hu, Zhi Chen, Xiaohui Su, Yongfeng Huang, Shijia Huang, Dahua Lin, Michael R Lyu, et al. Cleva: Chinese language models evaluation platform. *arXiv preprint arXiv:2308.04813*, 2023.

[45] Zhiyuan Zeng, Yizhong Wang, Hannaneh Hajishirzi, and Pang Wei Koh. Evaltree: Profiling language model weaknesses via hierarchical capability trees. *arXiv preprint arXiv:2503.08893*, 2025.

[46] Yaoxiang Wang, Haoling Li, Xin Zhang, Jie Wu, Xiao Liu, Wenxiang Hu, Zhongxin Guo, Yangyu Huang, Ying Xin, Yujiu Yang, et al. Epicoder: Encompassing diversity and complexity in code generation. *arXiv preprint arXiv:2501.04694*, 2025.

[47] Jian Li, Weiheng Lu, Hao Fei, Meng Luo, Ming Dai, Min Xia, Yizhang Jin, Zhenye Gan, Ding Qi, Chaoyou Fu, et al. A survey on benchmarks of multimodal large language models. *arXiv preprint arXiv:2408.08632*, 2024.

[48] Viorica Patraucean, Lucas Smaira, Ankush Gupta, Adria Recasens, Larisa Markeeva, Dylan Banarse, Skanda Koppula, Mateusz Malinowski, Yi Yang, Carl Doersch, et al. Perception test: A diagnostic benchmark for multimodal video models. *Advances in Neural Information Processing Systems*, 36:42748–42761, 2023.

[49] Bohao Li, Yuying Ge, Yixiao Ge, Guangzhi Wang, Rui Wang, Ruimao Zhang, and Ying Shan. Seed-bench: Benchmarking multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13299–13308, 2024.

[50] Jonathan Roberts, Mohammad Reza Taesiri, Ansh Sharma, Akash Gupta, Samuel Roberts, Ioana Croitoru, Simion-Vlad Bogolin, Jialu Tang, Florian Langer, Vyas Raina, et al. Zerobench: An impossible visual benchmark for contemporary large multimodal models. *arXiv preprint arXiv:2502.09696*, 2025.

[51] Paul Pu Liang, Yiwei Lyu, Xiang Fan, Zetian Wu, Yun Cheng, Jason Wu, Leslie Chen, Peter Wu, Michelle A Lee, Yuke Zhu, et al. Multibench: Multiscale benchmarks for multimodal representation learning. *Advances in neural information processing systems*, 2021(DB1):1, 2021.

[52] Richard Ren, Steven Basart, Adam Khoja, Alice Gatti, Long Phan, Xuwang Yin, Mantas Mazeika, Alexander Pan, Gabriel Mukobi, Ryan Kim, et al. Safetywashing: Do ai safety benchmarks actually measure safety progress? *Advances in Neural Information Processing Systems*, 37:68559–68594, 2024.

[53] Paul Röttger, Fabio Pernisi, Bertie Vidgen, and Dirk Hovy. Safetyprompts: a systematic review of open datasets for evaluating and improving large language model safety. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 27617–27627, 2025.

[54] Tianle Gu, Zeyang Zhou, Kexin Huang, Liang Dandan, Yixu Wang, Haiquan Zhao, Yuanqi Yao, Yujiu Yang, Yan Teng, Yu Qiao, et al. Mllmguard: A multi-dimensional safety evaluation suite for multimodal large language models. *Advances in Neural Information Processing Systems*, 37:7256–7295, 2024.

[55] Ruinan Jin, Zikang Xu, Yuan Zhong, Qingsong Yao, DOU QI, S Kevin Zhou, and Xiaoxiao Li. Fairmedfm: fairness benchmarking for medical imaging foundation models. *Advances in Neural Information Processing Systems*, 37:111318–111357, 2024.

[56] Song Wang, Peng Wang, Tong Zhou, Yushun Dong, Zhen Tan, and Jundong Li. Ceb: Compositional evaluation benchmark for fairness in large language models. *arXiv preprint arXiv:2407.02408*, 2024.

[57] Decheng Liu, Zongqi Wang, Chunlei Peng, Nannan Wang, Ruimin Hu, and Xinbo Gao. Thinking racial bias in fair forgery detection: Models, datasets and evaluations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 5379–5387, 2025.

[58] Wen Luo, Tianshu Shen, Wei Li, Guangyue Peng, Richeng Xuan, Houfeng Wang, and Xi Yang. Halludial: A large-scale benchmark for automatic dialogue-level hallucination evaluation. *arXiv preprint arXiv:2406.07070*, 2024.

[59] Selvan Sunitha Ravi, Bartosz Mielczarek, Anand Kannappan, Douwe Kiela, and Rebecca Qian. Lynx: An open source hallucination evaluation model. *arXiv preprint arXiv:2407.08488*, 2024.

[60] Peng Ding, Jingyu Wu, Jun Kuang, Dan Ma, Xuezhi Cao, Xunliang Cai, Shi Chen, Jiajun Chen, and Shujian Huang. Hallu-pi: Evaluating hallucination in multi-modal large language models within perturbed inputs. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 10707–10715, 2024.

[61] Kexin Huang, Xiangyang Liu, Qianyu Guo, Tianxiang Sun, Jiawei Sun, Yaru Wang, Zeyang Zhou, Yixu Wang, Yan Teng, Xipeng Qiu, et al. Flames: Benchmarking value alignment of chinese large language models. *CoRR*, 2023.

[62] Enyu Zhou, Guodong Zheng, Binghai Wang, Zhiheng Xi, Shihan Dou, Rong Bao, Wei Shen, Limao Xiong, Jessica Fan, Yurong Mou, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. Rmb: Comprehensively benchmarking reward models in llm alignment. In *The Thirteenth International Conference on Learning Representations*, 2025.

[63] Youquan Li, Miao Zheng, Fan Yang, Guosheng Dong, Bin Cui, Weipeng Chen, Zenan Zhou, and Wentao Zhang. Fb-bench: A fine-grained multi-task benchmark for evaluating llms' responsiveness to human feedback. *arXiv preprint arXiv:2410.09412*, 2024.

[64] Taneesh Gupta, Shivam Shandilya, Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Huaxiu Yao, and Saravan Rajmohan. Unveiling context-aware criteria in self-assessing llms. *arXiv preprint arXiv:2410.21545*, 2024.

[65] Anthropic. Claude 3.7 sonnet and claude code, 2025. Accessed: 2025-05-05.

[66] Tencent. Tencent hunyuan, 2025. Accessed: 2025-05-05.

[67] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[68] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[69] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[70] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

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

[72] Ariel N Lee, Cole J Hunter, and Nataniel Ruiz. Platypus: Quick, cheap, and powerful refinement of llms. *arXiv preprint arXiv:2308.07317*, 2023.

[73] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

Appendices Contents

A More Discussion

A.1 Distinctions from Prior Works Enabled by Fine-Grained Dataset

Several existing studies, such as SELF-INSTRUCT [42], Nemotron-4-340B [43], WildBench [2], CLEVA [44], and C^{2}LEVA [5], provide fine-grained evaluation of LLM capabilities. However, their taxonomies are relatively coarse and lack extensibility. For instance, C^{2}LEVA classifies LLM abilities into four broad categories—Language, Knowledge, Reasoning, and Harms—and further subdivides Language into only two types: Typo-Fixing and Transliteration. In contrast, our benchmark introduces an unprecedentedly detailed, fine-grained, and extensible taxonomy. Additionally, none of the aforementioned benchmarks provide visualization or analytical tools tailored for fine-grained evaluation, which is a key contribution of our work.

A concurrent study, EvalTree [45], also organizes queries into a tree structure. However, our TaxBuilder differs fundamentally. (1) Unlike EvalTree, which reorganizes existing datasets, our method starts with a fixed, extensible taxonomy and automatically synthesizes data to populate it. (2) TaxBuilder supports multi-dimensional classification based on topic, style, domain, and programming language, whereas EvalTree organizes queries solely by model capability. (3) Our method fully leverages the power of advanced reasoning models, resulting in more accurate classifications. In contrast, EvalTree relies on sentence embeddings—which may lack precision—and K-Means clustering, with cluster labels generated by LLMs. (4) We provide analysis tools to identify model strengths and weaknesses, which are absent in EvalTree. (6) We also addresses sample size requirements for reliable evaluation within subcategories, an important factor not considered by EvalTree.

A.2 Implications for Future Research

In the future, we believe Feedbacker will provide valuable insights to the community.

(1) For model training, Feedbacker can help identify a model's strengths and weaknesses. By quantifying fine-grained performance, it enables developers to adjust training parameters accordingly, such as data distribution.

(2) For model evaluation, Feedbacker prevents overestimating model capabilities. Its broad coverage across diverse categories allows it to reveal that models excelling in mainstream tasks may still struggle significantly in less common or specialized areas.

(3) For data synthesis, to the best of our knowledge, TaxBuilder offers an unprecedentedly scalable taxonomy that empowers developers to generate comprehensive, diverse and complex synthetic data for model training. For synthetic data, a concurrent work focusing on constructing code feature trees for code generation tasks [46] conveys ideas similar to those of TaxBuilder.

A.3 Limitations of Our Work

Currently, our framework is limited to language models and does not yet support multimodal models [47–51]. In future work, we plan to introduce a multimodal extension, Feedbacker-Anything. Moreover, our framework currently lacks support for evaluations related to AI mechanisms such as safety, honesty, factuality, hallucination, and fairness [52–57, 57–61]. However, our approach is highly extensible and can be easily adapted to include them as new evaluation domains.

B Experiments

B.1 Experimental Setup for Evaluation Method

Models. The P^{2}C pointwise evaluation involves two types of model. In the pre-comparison stage, multiple models are utilized to generate responses. Here we employ gpt-4o-2024-11-20 [28], doubao-1-5-pro-32k-250115 [35], and deepseek-v3-250324 [34] to ensure the diversity of the responses. For judge model, we adopt QwQ-32B [31] due to its lightweight nature and superior reasoning performance. Unless otherwise specified, we use the officially recommended decoding parameters.

Dataset. All our experiments are based on pairwise human preference datasets Feedbacker-HPD and RMB [62]. Pairwise human preference datasets usually contain a prompt and two responses, and

the model's task is to determine which response is better. For pairwise evaluation, the judge model directly selects the better response. For pointwise evaluation, the judge model assigns scores to both responses, with the higher-scoring response being chosen as the winner.

Feedbacker-HPD is a high-quality human preference dataset containing 636 samples. Queries are carefully filtered, and responses are annotated with majority human labels, making the dataset more representative of human preferences. A detailed description of Feedbacker-HPD is provided in § C.4.4. RMB is a recently popular large-scale human preference dataset comprising 10K queries for helpfulness and 7K for harmlessness. We only use the helpfulness subset in paper.

Unless otherwise specified, our experiments are conducted using our own constructed dataset, Feedbacker-HPD, which will be publicly released alongside Feedbacker-D-V0. Results on RMB are presented only in § B.2.

Baseline. We employ pairwise evaluation, vanilla pointwise evaluation, and naive criteria decomposition (NCD) as baselines. NCD refers to generating query-specific criteria based only on the query, without the process of pre-comparison [63, 7, 64]. Additionally, we also tested the impact of various components on P^2C pointwise evaluation. "Single Model Pre-Com" refers to using only gpt-4o-2024-11-20 with a temperature of 1, generating three responses. "Diverse Model Pre-Com" involves using gpt-4o-2024-11-20, doubao-1-5-pro-32k-250115, and deepseek-v3-250324 as models for generating auxiliary responses. For the baseline answer, we use gpt-4o-2024-11-20 with a temperature of 1.0 and top-p set to 0.95, generating a single response as the baseline answer. Regardless of whether we use vanilla pointwise evaluation or our method, when adding the baseline answer, we first evaluate it using the method without the baseline answer and include the evaluation result in the prompt. This process helps standardize the evaluation. Detailed instructions can be found in the respective prompts.

Prompt. For PC^2 pointwise evaluation, the prompt for pre-comparison-derived criteria extraction is shown in Fig. 9, the prompt for evaluation execution is shown in Fig. 10, the prompt for evaluation execution with baseline answer is shown in Fig. 11. For pairwise evaluation and vanilla pointwise evaluation, we directly use the prompts from MT-Bench [15]. For NCD, the corresponding prompt is displayed in Fig. 12.

PC² Pointwise Evaluation (Pre-Comparison-derived Criteria Extraction)

You are an impartial judge responsible for evaluating the quality of responses provided by different LLMs to a given [question]. Your task is to design a comprehensive evaluation framework that includes clearly defined metrics and their respective weights. You shuold answer step by step. You should answer in English. Please carefully follow these steps:

1. **Analyze Responses**: You must first compare several provided [answers] and identify their differences. The objective of this comparison is to pinpoint distinguishing factors that significantly influence the quality of the responses.
2. **Develop Metrics**: Establish a hierarchical set of evaluation metrics. There should be 3 to 9 primary metrics. Each primary metric should have several detailed sub-metrics to provide specific, measurable criteria for evaluating the responses.
3. **Assign Weights**: Allocate appropriate weights to each metric based on its relative importance in distinguishing the quality of the responses. The weights should be integers, and the sum of all weights should equal 100.
4. **Output Format**: Present the final evaluation framework in a structured list format. You do not need to include the primary metrics; only the secondary metrics are required, in the following format:
<Evaluation_Framework>
1. Description of Secondary Metric 1 | Weight 1
2. Description of Secondary Metric 2 | Weight 2
3. Description of Secondary Metric 3 | Weight 3
...
<Evaluation_Framework>

[User Question]
{question}

[The Start of Assistant 1's Answer]
{answer_1}
[The End of Assistant 1's Answer]

[The Start of Assistant 2's Answer]
{answer_2}
[The End of Assistant 2's Answer]

[The Start of Assistant 3's Answer]
{answer_3}
[The End of Assistant 3's Answer]

Figure 9: Prompt for pre-comparison-derived criteria extraction.

PC^2 Pointwise Evaluation (Evaluation Execution)

Please act as an impartial judge and evaluate the quality of the response provided by an AI assistant to the user question displayed below. Your evaluation should consider the evaluation system displayed below. Begin your evaluation by providing an explanation for each metric, assessing the response objectively. Score each metric on a scale of 1 to 3, where 1 represents complete failure to meet the criterion and 3 represents perfection. The weights of the metrics must sum to 100, and the final weighted score should be calculated on a scale up to 300, reflecting the weighted sum of the individual scores. After evaluation, you must summarize the results within <The Start of Evaluation Result> and <The End of Evaluation Result>. Below is an example output:

<The Start of Evaluation Result>
Metric 1 | score: [2]
Metric 2 | score: [3]
Metric 3 | score: [1]
...

Final Weighted Score: [[200]]
<The End of Evaluation Result>

[Question]
{question}

[The Start of Evaluation System]
{eval_system}
[The End of Evaluation System]

[The Start of Assistant's Answer]
{answer}
[The End of Assistant's Answer]

Figure 10: Prompt for evaluation execution.

PC^2 Pointwise Evaluation (Evaluation Execution with Baseline Answer)

Please act as an impartial judge and evaluate the quality of the response provided by an AI assistant to the user question displayed below. Your evaluation should consider the evaluation system displayed below. Begin your evaluation by providing an explanation for each metric, assessing the response objectively. Score each metric on a scale of 1 to 3. Use the baseline answer as a baseline; a higher score indicates a better response compared to the baseline answer, while a lower score indicates a worse response. The weights of the metrics must sum to 100, and the final weighted score should be calculated on a scale of 100 to 300, reflecting the weighted sum of the individual scores. After evaluation, you must summarize the results within <The Start of Evaluation Result> and <The End of Evaluation Result>. Below is an example output:
<The Start of Evaluation Result>
Metric 1 | score: [2]
Metric 2 | score: [3]
Metric 3 | score: [1]
...

Final Weighted Score: [[200]]
<The End of Evaluation Result>

[Question]
{question}

[The Start of Evaluation System]
{eval_system}
[The End of Evaluation System]

[The Start of Baseline Answer]
{answer_baseline}
[The End of Baseline Answer]

[The Start of Evaluation for Baseline Answer]
{critic_baseline}
[The End of Evaluation for Baseline Answer]

[The Start of Assistant's Answer]
{answer}
[The End of Assistant's Answer]

Figure 11: Prompt for evaluation execution with baseline answer.

Prompt of Naive Criteria Decomposition Pointwise Evaluation

You are an impartial judge responsible for evaluating the quality of responses provided by different LLMs to a given [question]. Your task is to design a comprehensive evaluation framework that includes clearly defined metrics and their respective weights. You shuold answer step by step. You should answer in English. Please carefully follow these steps:

1. **Analyze Question**: You must first analyze the question. The objective of this comparison is to find important factors that significantly influence the quality of the responses.
2. **Develop Metrics**: Establish a hierarchical set of evaluation metrics. There should be 3 to 9 primary metrics. Each primary metric should have several detailed sub-metrics to provide specific, measurable criteria for evaluating the responses.
3. **Assign Weights**: Allocate appropriate weights to each metric based on its relative importance in distinguishing the quality of the responses. The weights should be integers, and the sum of all weights should equal 100.
4. **Output Format**: Present the final evaluation framework in a structured list format. You do not need to include the primary metrics; only the secondary metrics are required, in the following format:
<Evaluation_Framework>
1. Description of Secondary Metric 1 | Weight 1
2. Description of Secondary Metric 2 | Weight 2
3. Description of Secondary Metric 3 | Weight 3
...
<Evaluation_Framework>

[User Question]
{question}

Figure 12: Prompt for naive criteria decomposition.

B.2 More Experiment on PC^2-Pointwise Evaluation

In this section, we present the accuracy of PC^2 pointwise evaluation on the RMB [62] benchmark. The results shown in Tab. 4 demonstrate that while PC^2 maintains the pointwise evaluation paradigm, it achieves results comparable to pairwise evaluation.

Table 4: Comparison of accuracy across different LLM-as-a-Judge methods on RMB [62].

Method	Overall	Generation	Chat	Role Playing	Brainstorming	Summarization	Rewrite	Classification	Translation	Open QA	Closed QA	Code	Reasoning
Pairwise Evaluation	0.7427	0.8000	0.6566	0.7949	0.7469	0.7544	0.7374	0.8333	0.7267	0.7014	0.6923	0.8280	0.7331
Vanilla Pointwise Evaluation	0.6197	0.6723	0.5060	0.7436	0.5768	0.6316	0.5354	0.7222	0.6333	0.6082	0.5692	0.7640	0.5657
PC^2 Pointwise Evaluation	0.7353	0.8000	0.6687	0.8333	0.7676	0.6667	0.6667	0.7778	0.6867	0.7315	0.6718	0.7600	0.7689
+ Baseline Answer	0.7394	0.7940	0.7108	0.7821	0.7083	0.6842	0.6263	0.6667	0.6846	0.7890	0.7077	0.7800	0.7490

B.3 Experimental Setup for Evaluation and Leaderboard

Models. We select 17 mainstream LLMs for evaluation. For closed-source models, we select claude3.7-sonnet-20250219 [65], gpt-4o-2024-11-20 [28], qwen-max-2024-10-15 [], doubao-1-5-pro-32k-250115 [35], and hunyuan-standard-2025-02-10 [66]. For open-source models, we select deepseek-v3-250324 [34], deepseek-r1-250120 [36], QwQ-32B [31], Gemma-3-27B-IT [67], Qwen2.5-72B-Instruct [68], Gemma-3-4B-IT [67], Qwen2.5-32B-Instruct [68], Meta-Llama-3.1-70B-Instruct [69], Qwen2.5-7B-Instruct [68], Meta-Llama-3.1-8B-Instruct [69], Mistral-7B-Instruct-v0.3 [], and Phi-4-Mini-Instruct [70]. Among these, deepseek-v3-250324 and deepseek-r1-250120 are deployed with fp8, while the other models are deployed with bf16. For all models, if their official decoding parameters are provided, we use the default settings. If not, we set the temperature to 0.7 and top_p to 0.95.

B.4 Leaderboard

We propose in the paper how to quickly check the weaknesses of a LLM through the tree structure. Due to the limitation of the paper's length, we only present the first-level nodes (i.e., the six domain) in the paper, while the more fine-grained results are presented on our project website. The leaderboard results, as presented in Tab. 5, reveal some valuable weaknesses of existing LLMs. These weaknesses

can significantly guide the model's development if they are timely available during the development process of the model. Deepseek-v3-250324 ranks at the forefront among all tested models due to its superior coding, mathematics, and reasoning capabilities. However, it is noteworthy that its roleplay ability exhibits significant deficiencies. Claude3.7-sonnet-20250219 demonstrates high proficiency in coding, yet its overall capabilities are relatively weak. Gemma-3-4B-IT achieves exceptionally high scores in liberal arts subjects, but it performs poorly in coding and mathematics, which are more science-oriented. Additionally, it is worth mentioning that doubao-1-5-pro-32k-250115 possesses a notable advantage in mathematical abilities.

Table 5: Overall and domain-level scores (fine-grained results on project page).

Model	Size	Overall	Writing	Roleplay	Knowledge	Coding	Mathematics	Reasoning
Closed-source and Open-source (>100B) LLMs								
deepseek-v3-250324	671B	454.47(1)	454.08(3)	443.75(5)	464.54(2)	452.89(1)	450.28(1)	464.53(1)
deepseek-r1-250120	671B	453.76(2)	468.82(1)	461.19(1)	460.32(4)	434.31(3)	418.86(3)	456.35(4)
claude3.7-sonnet-20250219	🔒	433.34(5)	441.32(6)	413.21(7)	440.30(6)	443.67(2)	413.26(4)	425.93(6)
gpt-4o-2024-11-20	🔒	433.30(6)	443.45(5)	450.16(4)	446.93(5)	410.70(5)	387.72(6)	447.76(5)
qwen-max-2024-10-15	🔒	392.19(8)	405.38(9)	381.76(8)	408.78(9)	368.64(9)	369.27(9)	409.36(8)
doubao-1-5-pro-32k-250115	🔒	384.80(10)	390.39(10)	356.37(11)	395.58(10)	375.10(8)	399.26(5)	395.22(10)
hunyuan-standard-2025-02-10	🔒	345.51(14)	361.80(15)	336.03(14)	351.35(14)	322.40(14)	328.22(12)	361.42(14)
Open-source (<100B) LLMs								
QwQ-32B	32B	452.75(3)	465.50(2)	455.78(2)	464.94(1)	426.15(4)	430.91(2)	462.08(3)
Gemma-3-27B-IT	27B	437.61(4)	449.74(4)	452.91(3)	462.22(3)	399.08(6)	392.74(4)	462.13(2)
Qwen2.5-72B-Instruct	72B	395.04(7)	406.01(8)	378.90(9)	409.28(8)	383.07(7)	372.91(8)	407.08(9)
Gemma-3-4B-IT	4B	391.92(9)	420.14(7)	431.55(6)	426.82(7)	326.42(12)	292.43(14)	421.03(7)
Qwen2.5-32B-Instruct	32B	370.52(11)	386.67(12)	356.79(10)	380.65(11)	342.44(10)	360.14(10)	388.93(11)
Meta-Llama-3.1-70B-Instruct	70B	362.23(12)	388.10(11)	360.94(9)	368.04(12)	336.79(11)	305.13(13)	381.05(12)
Qwen2.5-7B-Instruct	7B	357.59(13)	378.48(13)	348.52(12)	367.58(13)	324.79(13)	339.03(11)	369.78(13)
Meta-Llama-3.1-8B-Instruct	8B	338.07(15)	366.84(14)	345.72(13)	345.94(15)	310.33(15)	271.33(16)	341.63(16)
Mistral-7B-Instruct-v0.3	7B	314.22(16)	351.71(16)	331.22(15)	324.97(16)	272.39(16)	223.40(17)	324.49(17)
Phi-4-Mini-Instruct	3.8B	309.55(17)	328.02(17)	320.75(16)	307.78(17)	268.48(17)	287.93(15)	345.47(15)

C Taxonomy and Dataset

C.1 TaxBuilder

C.1.1 Manually Initialized Basic Taxonomy Tree

TaxBuilder requires a manually crafted initial taxonomy as a starting point. This step is quite straightforward. First, we need to define a root node—typically a domain—such as coding.

Next, to capture the various aspects of this domain, TaxBuilder asks users to manually define several classification principles, such as task types, technical domains, and programming languages.

Finally, we manually construct a basic fine-grained taxonomy. Fig. 13 shows an example of a manually created taxonomy for the coding domain. Additional examples can be found in our open-source code repository.
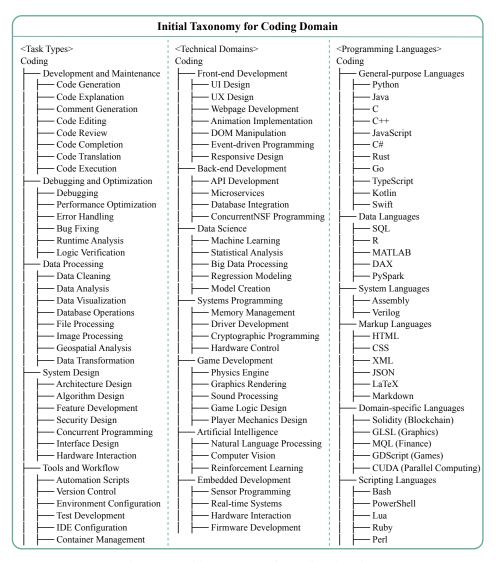
Initial Taxonomy for Coding Domain

```
<Task Types>
Coding
├── Development and Maintenance
│   ├── Code Generation
│   ├── Code Explanation
│   ├── Comment Generation
│   ├── Code Editing
│   ├── Code Review
│   ├── Code Completion
│   ├── Code Translation
│   └── Code Execution
├── Debugging and Optimization
│   ├── Debugging
│   ├── Performance Optimization
│   ├── Error Handling
│   ├── Bug Fixing
│   ├── Runtime Analysis
│   └── Logic Verification
├── Data Processing
│   ├── Data Cleaning
│   ├── Data Analysis
│   ├── Data Visualization
│   ├── Database Operations
│   ├── File Processing
│   ├── Image Processing
│   ├── Geospatial Analysis
│   └── Data Transformation
├── System Design
│   ├── Architecture Design
│   ├── Algorithm Design
│   ├── Feature Development
│   ├── Security Design
│   ├── Concurrent Programming
│   ├── Interface Design
│   └── Hardware Interaction
└── Tools and Workflow
    ├── Automation Scripts
    ├── Version Control
    ├── Environment Configuration
    ├── Test Development
    ├── IDE Configuration
    └── Container Management
```

```
<Technical Domains>
Coding
├── Front-end Development
│   ├── UI Design
│   ├── UX Design
│   ├── Webpage Development
│   ├── Animation Implementation
│   ├── DOM Manipulation
│   ├── Event-driven Programming
│   └── Responsive Design
├── Back-end Development
│   ├── API Development
│   ├── Microservices
│   ├── Database Integration
│   └── ConcurrentNSF Programming
├── Data Science
│   ├── Machine Learning
│   ├── Statistical Analysis
│   ├── Big Data Processing
│   ├── Regression Modeling
│   └── Model Creation
├── Systems Programming
│   ├── Memory Management
│   ├── Driver Development
│   ├── Cryptographic Programming
│   └── Hardware Control
├── Game Development
│   ├── Physics Engine
│   ├── Graphics Rendering
│   ├── Sound Processing
│   ├── Game Logic Design
│   └── Player Mechanics Design
├── Artificial Intelligence
│   ├── Natural Language Processing
│   ├── Computer Vision
│   └── Reinforcement Learning
└── Embedded Development
    ├── Sensor Programming
    ├── Real-time Systems
    ├── Hardware Interaction
    └── Firmware Development
```

```
<Programming Languages>
Coding
├── General-purpose Languages
│   ├── Python
│   ├── Java
│   ├── C
│   ├── C++
│   ├── JavaScript
│   ├── C#
│   ├── Rust
│   ├── Go
│   ├── TypeScript
│   ├── Kotlin
│   └── Swift
├── Data Languages
│   ├── SQL
│   ├── R
│   ├── MATLAB
│   ├── DAX
│   └── PySpark
├── System Languages
│   ├── Assembly
│   └── Verilog
├── Markup Languages
│   ├── HTML
│   ├── CSS
│   ├── XML
│   ├── JSON
│   ├── LaTeX
│   └── Markdown
├── Domain-specific Languages
│   ├── Solidity (Blockchain)
│   ├── GLSL (Graphics)
│   ├── MQL (Finance)
│   ├── GDScript (Games)
│   └── CUDA (Parallel Computing)
└── Scripting Languages
    ├── Bash
    ├── PowerShell
    ├── Lua
    ├── Ruby
    └── Perl
```

Figure 13: Initial taxonomy for coding domain.

C.1.2   Initial Annotation for Query Type Tags

We use gpt-4o-2024-11-20 to generate a large number of low-quality query type tags. The prompt used for generation is shown in Fig. 14.

Initial low-quality Type Labeling

Task
Your task is to categorize a given query based on the following rules. For each rule, explain your reasoning before assigning a label. Evaluate the query against each domain in the order provided below until a match is found. Once a domain is identified, proceed with its specific sub-rules. If no domain matches, proceed to the next domain check. If no specific domain applies, return a generic `"other"` label.

Rules
Step 1: Determine the Domain
Evaluate the query against the following domains in this order. Stop and proceed with the corresponding sub-rules once a domain is matched. If the domain is labeled `"other"`, return {"domain": "other", "domain_name": "<custom_domain_name>"} and do not proceed further. Explain your reasoning before assigning a label.
1. Writing - Condition: If the query relates to writing, purposeful professional writing, literature, storytelling, grammar, or language-related topics. - Label: `"writing"` - If not matched, proceed to the next domain.
2. Roleplay - Condition: If the query involves roleplay, character interactions, storytelling, or immersive scenario-based dialogue. - Label: `"roleplay"` - If not matched, proceed to the next domain.
3. Coding - Condition: If the query relates to programming, algorithms, or code-related topics. - Label: `"coding"` - If not matched, proceed to the next domain.
4. Mathematics - Condition: If the query pertains to mathematical concepts, computations, proofs, or problem-solving. - Label: `"mathematics"` - If not matched, proceed to the next domain.
5. Reasoning - Condition: If the query pertains to reasoning, logic, critical thinking, or problem-solving without direct reference to specific knowledge or programming. - Label: `"reasoning"` - If not matched, proceed to the next domain.
6. Knowledge - Condition: If the query pertains to factual knowledge or general subject areas such as science, history, literature, philosophy, or current affairs. - Label: `"knowledge"` - If not matched, label as `"other"` with a custom `"domain_name"` and stop.

Step 2: Domain-Specific Categorization
Once the domain is determined, apply the corresponding sub-rules below. If the domain is `"other"`, skip this step. Explain your reasoning before assigning a label.
- Domain: "writing" - Label 2: Writing Topic - Specify the writing topic in lowercase (e.g., `"creative writing"`, `"speech"`, `"LLM prompt"`, `"game"`, `"academic"`, `"social media"`, `"shopping"`, `"formal"`, `"casual"`, etc.). - If unclear or not explicitly mentioned, use `"null"`. - If none of the predefined topics fit, provide a custom description. - Label 3: Task Type - Choose applicable task types (multiple may apply): - `"Creative Writing"`: Crafting original content, poems, or creative pieces. - `"Polishing"`: Improving clarity, style, and flow. - `"Grammar Check"`: Identifying and correcting grammar mistakes. - `"Content Refining"`: Enhancing word choice, tone, or style. - `"Summarization"`: Condensing text while retaining the core message. - `"Expanding"`: Adding details or elaboration. - `"Paraphrasing"`: Restating content differently. - `"Writing Feedback"`: Providing constructive feedback. - `"Translating"`: Converting text between languages. - `"Tone Adjustment"`: Modifying tone for audience/context. - `"Text Summarization"`: Analyzing meaning, themes, or key points. - If none fit, provide a custom description.
- Domain: "roleplay" - Label 2: Genre - Specify the genre in lowercase (e.g., `"sci-fi"`, `"fantasy"`, `"historical"`, `"romance"`, `"mystery"`, `"horror"`, `"action/adventure"`, `"supernatural"`, etc.). - If multiple genres apply, list them all. - If none fit, provide a custom description. - Label 3: Task Type - Choose applicable task types (multiple may apply): - `"Dialogue Generation"`: Creating natural character conversations. - `"Content Creation"`: Writing from a character's perspective. - `"Question Answering"`: Responding within character constraints. - `"Emotional Simulation"`: Modeling character-appropriate emotions. - `"Decision Support"`: Providing character-based guidance. - `"Skill Training"`: Simulating expert/practice partners. - `"Behavior Simulation"`: Replicating character behaviors. - `"Evaluation & Feedback"`: Assessing from a character's perspective. - If none fit, provide a custom description.
- Domain: "coding" - Label 2: Programming Language - Specify the language in lowercase (e.g., `"python"`, `"java"`, `"c++"`, `"sql"`, etc.). - If unclear or not mentioned, use `"null"`. - Label 3: Task Type - Choose applicable task types (multiple may apply): - `"Code Generation"`: Creating code from a prompt. - `"Debugging"`: Resolving code errors. - `"Code Explanation"`: Explaining code functionality. - `"Code Editing"`: Modifying existing code. - `"Conceptual QA"`: Answering programming theory questions. - `"Code Translation"`: Converting code between languages. - `"Bug Fixing"`: Fixing specific code issues. - `"Testing"`: Writing/analyzing tests. - `"Data Structures & Algorithms"`: Solving related problems. - `"System Design"`: Designing software architecture. - `"Optimization"`: Improving code efficiency. - `"Security"`: Addressing vulnerabilities. - `"Documentation"`: Writing/improving documentation. - If none fit, provide a custom description.
- Domain: "mathematics" - Label 2: Subfield - Specify the subfield in lowercase (e.g., `"algebra"`, `"calculus"`, `"geometry"`, `"statistics & probability"`, etc.). - If unclear, use `"null"`. - Label 3: Task Type - Choose applicable task types (multiple may apply): - `"Proof Writing"`: Writing formal proofs. - `"Conceptual Understanding"`: Explaining concepts. - `"Problem-Solving"`: Applying techniques to challenges. - `"Real-World Application"`: Connecting math to scenarios. - `"Graphing & Visualization"`: Interpreting/drawing graphs. - `"Logical Deduction"`: Deriving conclusions logically. - If none fit, provide a custom description.
- Domain: "reasoning" - Label 2: Reasoning Type - Specify the type in lowercase (e.g., `"logical reasoning"`, `"critical thinking"`, `"problem-solving"`, `"analytical thinking"`, etc.). - If unclear, use `"null"`. - Label 3: Task Type - Choose applicable task types (multiple may apply): - `"Logical Deduction"`: Deriving conclusions logically. - `"Puzzle Solving"`: Solving riddles or brainteasers. - `"Argument Evaluation"`: Analyzing argument validity. - `"Scenario Analysis"`: Considering scenario outcomes. - `"Theoretical Reasoning"`: Exploring abstract concepts. - `"Creative Problem-Solving"`: Finding novel solutions. - `"Ethical Reasoning"`: Evaluating based on ethics. - If none fit, provide a custom description.
- Domain: "knowledge" - Label 2: Subject - Specify the subject in lowercase (e.g., `"history"`, `"philosophy"`, `"geography"`, `"medicine"`, `"economics"`, etc.). - If unclear, use `"null"`. - Label 3: Task Type - Choose applicable task types (multiple may apply): - `"Fact Recall"`: Retrieving specific facts. - `"Conceptual Understanding"`: Explaining/interpreting ideas. - `"Comparative Analysis"`: Comparing concepts. - `"Causal Reasoning"`: Examining cause-and-effect. - `"Logical Deduction"`: Applying logic to conclude. - `"Hypothetical Thinking"`: Exploring "what if" scenarios. - `"Problem-Solving"`: Solving practical/theoretical problems. - `"Opinion-Based"`: Providing subjective reasoning. - If none fit, provide a custom description.

Step 3: Final Output
Return a Python dictionary based on the identified domain and labels:
- For `"other"`: {"domain": "other", "domain_name": "<custom_domain_name>"}
- For specific domains:
{"domain": "<domain>", "task_type": ["<task1>", "<task2>"], "content_type": ["<content1>", "<content2>"]}
- Replace <domain> with the identified domain (e.g., `"writing"`).
- Replace `"task"` with the list of identified task types (or empty list [] if none apply).
- Replace <content> with the corresponding value (e.g., `"poetry"`, `"sci-fi"`, `"history"`, etc.).

Figure 14: Prompt for initial annotation of tags.

C.1.3 Algorithm for TaxBuilder

The complete TaxBuilder algorithm is presented in Alg. 1.

Algorithm 1 TaxBuilder Algorithm

1: **Input:** Initial tree T_{init}, new nodes TG_{init}, LLM-as-Decision-Maker of node insertion DM_{ins}
2: **Output:** Final taxonomy tree T_{final}
3:
4: **function** INSERTNODE(T, tg)
5:     $d \leftarrow \text{DM}_{ins}(tg, T^c)$
6:     **if** $d = $ "<E>" **then**
7:         **return** T
8:     **else if** $d = $ "<S>" **then**
9:         $T^c \leftarrow T^c \cup \{tg\}$
10:        **return** T
11:     **else**   ▷ Child level: LLM returns parent node name
12:        $T^{c_i} \leftarrow d$
13:        **return** INSERTNODE(T^{c_i}, tg)
14:     **end if**
15: **end function**
16:
17: **function** TAXBUILDER(T_{init}, TG_{init})
18:     $T_{cur} \leftarrow T_{init}$
19:     **for all** $tg \in TG_{init}$ **do**
20:        $T_{cur} \leftarrow$ INSERTNODE(T_{cur}, tg)
21:     **end for**
22:     $T_{cur} \leftarrow$ REFINEPRUNENODE(T_{cur}) ▷ Node refinement and pruning
23:     $T_{final} \leftarrow$ PRUNELAYER(T_{cur}) ▷ Layer pruing
24:     **return** T_{final}
25: **end function**

C.1.4 Prompts for TaxBuilder

The prompt for node insertion in the LLM-as-Decision-Maker framework is shown in Fig. 15. The prompt for node refinement and pruning in the LLM-as-Decision-Maker framework is shown in Fig. 16.

Prompt of LLM-as-Decision-Maker (Node Insertion)

Your task is to add a new node to a tree-structured classification system. I will give you some [`Current Level Nodes`] and a [`New Node`]. You need to determine the next action and respond strictly in the specified format. Before giving the final answer, you must perform brief analysis.

Task 1
Analyze whether the [`New Node`] shares identical meaning with any [`Current Level Nodes`] (including exact matches, case variations, abbreviations, or conceptually equivalent expressions). If identical, return `<decision>EXIST</decision>` and skip subsequent tasks.
Analyze whether the [`New Node`] represents a specific category or a classification criterion. If it's a classification criterion, return `<decision>EXIST</decision>` and skip subsequent tasks.

Task 2
Analyze whether the [`New Node`] doesn't belong to any subordinate node of [`Current Level Nodes`] but should be at the same level. If yes, return `<decision>ADD</decision>` and skip subsequent tasks.

Task 3
Identify which [`Current Level Node`] the [`New Node`] should belong to as a subordinate node. Return the parent node's name wrapped in `<decision></decision>`.

Example
Current Level Nodes: ["Fruits", "Vegetables", "Electronics"]
New Node: Phone
Output: `<decision>Electronics</decision>`

New Node
{new_node}

Current Level Nodes
{current_keys}

Reference Classification System (for reference only)
{init_tree}

Figure 15: Prompt of LLM-as-Decision-Maker (node insertion).

Figure 16: Prompt of LLM-as-Decision-Maker (node refinement and pruning).

C.2   RealMix

C.2.1   Seed Data

The prompt for labeling the domain is shown in Fig.17.

The prompts for query tagging are shown in Fig.18, Fig.19, Fig.20, Fig.21, Fig.22, and Fig.23.

The prompts for quality labeling are shown in Fig.24, Fig.25, Fig.26, Fig.27, Fig.28, and Fig. 29.

The seed data is available in our dataset repository.

You are tasked with categorizing a given query into one of the following six categories:
1. **roleplay**
2. **coding**
3. **mathematics**
4. **reasoning**
5. **knowledge**
6. **writing**
7. **other**

For each query provided, determine the most appropriate category and output the result in lowcase enclosed within <domain> and </domain> tags.

Example:
Query: "How do I write a compelling essay introduction?"
Output: <domain>roleplay</domain>

Now, analyze the following query:
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 17: Prompt of domain annotation.

You are tasked with categorizing a given query into multiple tags based on the following hierarchical classification system.

Classification System:
{taxonomy}

Rules for Tagging:
1. **Hierarchy Rule**: If a query matches both a parent node and its child nodes, include both in the tags.
2. **Multiple Matches**: If a query matches multiple nodes at the same level, include all matching nodes.
3. **No Match**: If a query does not match any nodes under the second-level main categories, assign the tag "Other" to that category.
4. **Output Format**: The final output must be enclosed within '<tags>' and '</tags>' tags, and the tags should be provided as a JSON object where the keys are the basis for classification and the values are lists of matching tags.

Example:
Query: "I need help brainstorming ideas for a fantasy story involving a hidden kingdom and magical creatures."
Output: <tags>{{"Creative Stages": ["Conceptualization Stage", "Brainstorming", "World Building", "Creative Exploration", "Content Conceptualization"], "Writing Domains": ["Literary Writing", "Fiction", "Fantasy", "Story Creation"], "Styles": ["Other"]}}</tags>

Now, analyze the following query:
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 18: Prompt of query tags annotation (writing).

Prompt of Query Tags Annotation (roleplay)

You are tasked with categorizing a given query into multiple tags based on the following hierarchical classification system.

Classification System:
{taxonomy}

Rules for Tagging:
1. **Hierarchy Rule**: If a query matches both a parent node and its child nodes, include both in the tags.
2. **Multiple Matches**: If a query matches multiple nodes at the same level, include all matching nodes.
3. **No Match**: If a query does not match any nodes under the second-level main categories, assign the tag "Other" to that category.
4. **Output Format**: The final output must be enclosed within '<tags>' and '</tags>' tags, and the tags should be provided as a JSON object where the keys are the basis for classification and the values are lists of matching tags.

Example:
Query: "Write a script for a medieval fantasy story involving a knight's adventure and a magical encounter, with a humorous twist."
Output: <tags>{{"Theme Types": ["Fantasy", "Medieval", "Magic", "Comedy", "Humorous Comedy"], "Task Types": ["Creative", "Scriptwriting"], "Style Types": ["Humorous Style", "Light Humor", "Narrative"]}}</tags>

Now, analyze the following query:
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 19: Prompt of query tags annotation (roleplay).

Prompt of Query Tags Annotation (knowledge)

You are tasked with categorizing a given query into multiple tags based on the following hierarchical classification system.

Classification System:
{taxonomy}

Rules for Tagging:
1. **Hierarchy Rule**: If a query matches both a parent node and its child nodes, include both in the tags.
2. **Multiple Matches**: If a query matches multiple nodes at the same level, include all matching nodes.
3. **No Match**: If a query does not match any nodes under the second-level main categories, assign the tag "Other" to that category.
4. **Output Format**: The final output must be enclosed within '<tags>' and '</tags>' tags, and the tags should be provided as a JSON object where the keys are the basis for classification and the values are lists of matching tags.

Example:
Query: "How can I analyze data from a physics experiment on thermodynamics and present it effectively?"
Output: <tags>{{"Disciplinary Fields": ["Natural Sciences", "Physics", "Thermodynamics"], "Cognitive Levels": ["Applied Analysis"], "Task Types": ["Information Processing", "Data Analysis", "Content Production", "Content Generation"]}}</tags>

Now, analyze the following query:
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 20: Prompt of query tags annotation (knowledge).

Prompt of Query Tags Annotation (coding)

You are tasked with categorizing a given query into multiple tags based on the following hierarchical classification system.

Classification System:
{taxonomy}

Rules for Tagging:
1. **Hierarchy Rule**: If a query matches both a parent node and its child nodes, include both in the tags.
2. **Multiple Matches**: If a query matches multiple nodes at the same level, include all matching nodes.
3. **No Match**: If a query does not match any nodes under the second-level main categories, assign the tag "Other" to that category.
4. **Output Format**: The final output must be enclosed within '<tags>' and '</tags>' tags, and the tags should be provided as a JSON object where the keys are the basis for classification and the values are lists of matching tags.

Example:
Query: "How can I optimize a Python script that processes large datasets and visualizes the results?"
Output: <tags>{{"Task Types": ["Data Processing", "Data Analysis", "Data Visualization", "Debugging and Optimization", "Optimization"], "Technical Domains": ["Data Science", "Data Analysis Methods", "Data Visualization Tools"], "Programming Languages": ["General-purpose Languages", "Python"]}}</tags>

Now, analyze the following query:
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 21: Prompt of query tags annotation (coding).

Prompt of Query Tags Annotation (mathematics)

You are tasked with categorizing a given query into multiple tags based on the following hierarchical classification system.

Classification System:
{taxonomy}

Rules for Tagging:
1. **Hierarchy Rule**: If a query matches both a parent node and its child nodes, include both in the tags.
2. **Multiple Matches**: If a query matches multiple nodes at the same level, include all matching nodes.
3. **No Match**: If a query does not match any nodes under the second-level main categories, assign the tag "Other" to that category.
4. **Output Format**: The final output must be enclosed within '<tags>' and '</tags>' tags, and the tags should be provided as a JSON object where the keys are the basis for classification and the values are lists of matching tags.

Example:
Query: "How do I use calculus to model the growth of a population over time and graph the results?"
Output: <tags>{{"Mathematical Subfields": ["Analysis", "Calculus", "Applied Mathematics"], "Task Types": ["Problem Solving", "Modeling", "Visualization", "Function Graphing"]}}</tags>

Now, analyze the following query:
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 22: Prompt of query tags annotation (mathematics).

Prompt of Query Tags Annotation (reasoning)

You are tasked with categorizing a given query into multiple tags based on the following hierarchical classification system.

Classification System:
{taxonomy}

Rules for Tagging:
1. **Hierarchy Rule**: If a query matches both a parent node and its child nodes, include both in the tags.
2. **Multiple Matches**: If a query matches multiple nodes at the same level, include all matching nodes.
3. **No Match**: If a query does not match any nodes under the second-level main categories, assign the tag "Other" to that category.
4. **Output Format**: The final output must be enclosed within '<tags>' and '</tags>' tags, and the tags should be provided as a JSON object where the keys are the basis for classification and the values are lists of matching tags.

Example:
Query: "How can I determine the cause of an event based on multiple contributing factors?"
Output: <tags>{{"Reasoning Methods": ["Multi-factor Attribution"], "Application Domains": ["Other"], "Thinking Modes": ["Analytical"], "Task Types": ["Reasoning"]}}</tags>

Now, analyze the following query:
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 23: Prompt of query tags annotation (reasoning).

Prompt of Quality Annotation (writing)

Task
Your task is to assess the quality of a given query based on the checklists outlined below. For each checklist, provide a clear explanation of your reasoning before assigning "Yes" or "No". If your answer is "Borderline", please answer "Yes" for this checklist.

Checklists
Determine whether the provided question meets the following criteria. Return a Python array listing the numbers of all satisfied criteria:

1. **Clarity**: Is the question clear and well-defined?
2. **Completeness**: Does the question provide enough information for the LLM to answer the question?
3. **Complexity**: Does the question have enough depth and challenge beyond simple fact recall?
4. **Real-World Application**: Is this writing task something that would be proposed in the real world?
5. **Professionalism**: Does it require professional capabilities or professional knowledge?
6. **Originality:** Does the question encourage or require originality?
7. **User's Requirements**: Does the user have clear, detailed, or unique requests that need to be considered in the response?

For example, if the question meets Clarity, Completeness, and Real-World Application, return '[1, 2, 4]'.

Final Output

For each question provided, return a Python dictionary in the following format:
```python
Final Labels: {{"question_quality": [1, 3, 4]}}
```

The Query You Should Process
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 24: Prompt of quality annotation (writing).

Prompt of Quality Annotation (roleplay)

Task
Your task is to assess the quality of a given query based on the checklists outlined below. For each checklist, provide a clear explanation of your reasoning before assigning "Yes" or "No". If your answer is "Borderline", please answer "Yes" for this checklist.

Checklists
Determine whether the provided question meets the following criteria. Return a Python array listing the numbers of all satisfied criteria:

1. **Clarity**: Is the question clear and well-defined?
2. **Completeness**: Does the question provide enough information for the LLM to answer the question?
3. **Complexity**: Does it involve in-depth understanding of any role-playing content, such as the psychology, characterization, and world-building of characters?
4. **Real-World Application**: Is this role-playing task something that would be proposed in the real world?
5. **Interactivity**: Does the question encourage meaningful interactions between characters, rather than single character?
6. **Engagement**: Does the task motivate active participation and emotional involvement from the audience or participants?
7. **Creativity:** Does it have creativity and novelty, or does solving it require creativity?

For example, if the question meets Clarity, Completeness, and Real-World Application, return '[1, 2, 4]'.

Final Output

For each question provided, return a Python dictionary in the following format:
```python
Final Labels: {{"question_quality": [1, 3, 4]}}
```

The Query You Should Process
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 25: Prompt of quality annotation (roleplay).

Prompt of Quality Annotation (knowledge)

Task
Your task is to assess the quality of a given query based on the checklists outlined below. For each checklist, provide a clear explanation of your reasoning before assigning "Yes" or "No". If your answer is "Borderline", please answer "Yes" for this checklist.

Checklists
Determine whether the provided question meets the following criteria. Return a Python array listing the numbers of all satisfied criteria:

1. **Clarity**: Is the question clear and well-defined?
2. **Completeness**: Does the question provide enough information for the LLM to answer the question?
3. **Complexity**: Does the question have enough depth and challenge beyond simple fact recall?
4. **Real-World Application**: Is the question something that would be encountered in real-world?
5. **Depth of Knowledge**: Does the question require deep expertise in the subject instead of just memory?
6. **Cross-Disciplinary**: Does the question involve cross-disciplinary aspects?
7. **Open-Endedness.**: Does the question encourage open-ended responses rather than simple "yes" or "no" answers, promoting deeper thinking?

For example, if the question meets Clarity, Completeness, and Real-World Application, return '[1, 2, 4]'.

Final Output

For each question provided, return a Python dictionary in the following format:
```python
Final Labels: {{"question_quality": [1, 3, 4]}}
```

The Query You Should Process
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 26: Prompt of quality annotation (knowledge).

Prompt of Quality Annotation (coding)

Task
Your task is to assess the quality of a given query based on the checklists outlined below. For each checklist, provide a clear explanation of your reasoning before assigning "Yes" or "No". If your answer is "Borderline", please answer "No" for this checklist.

Checklists
Determine whether the provided question meets the following criteria. Return a Python array listing the numbers of all satisfied criteria:

1. **Clarity**: Is the question clear and well-defined?
2. **Completeness**: Does the question provide enough information for the LLM to answer the question?
3. **Complexity**: Does it involve multiple components, layers, or nuance?
4. **Real-World Application**: Is the question something that would be encountered in real-world development?
5. **Problem-Solving**: Does it require active problem-solving beyond simple and superficial script or fact recall?
6. **Domain-Specific Expertise**: Does the question require in-depth knowledge of at least one specific area of programming?
7. **Specified Requirements**: Does it specify particular requirements, such as execution time, space constraints, specific programming language, tools, packages, etc.?

For example, if the question meets Clarity, Completeness, and Real-World Application, return '[1, 2, 4]'.

Final Output

For each question provided, return a Python dictionary in the following format:
```python
Final Labels: {{"question_quality": [1, 3, 4]}}
```

The Query You Should Process
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 27: Prompt of quality annotation (coding).

Prompt of Quality Annotation (mathematics)

Task
Your task is to assess the query of a given question based on the rules outlined below. For each rule, provide a clear explanation of your reasoning before assigning a label.

Rule
Determine whether the provided query meets the following criteria. Return a Python array listing the numbers of all satisfied criteria:

1. **Clarity**: Is the question clear and well-defined?
2. **Completeness**: Does the question provide enough information for the LLM to answer the question?
3. **Complexity**: Does it involve multiple steps, analysis, or reasoning instead of simple concept memorization and numerical calculation?
4. **Real-World Application**: Is the question something that would be encountered in real-world?
5. **Problem-Solving**: Does it test the ability to apply math in some scenarios?
6. **Rigorous Logic**: Does it involve content such as theorem derivation and formula understanding, which require rigorous logical abilities?
7. **Creativity:** Does it have creativity and novelty, or does solving it require creativity?

For example, if the question meets Clarity, Completeness, and Real-World Application, return '[1, 2, 4]'.

Final Output

For each question provided, return a Python dictionary in the following format:
```python
Final Labels: {{"question_quality": [1, 3, 4]}}
```

The Query You Should Process
<|begin_of_query|>
{query}
<|end_of_query|>

Figure 28: Prompt of quality annotation (mathematics).

Figure 29: Prompt of quality annotation (reasoning).

C.2.2 Synthesize New Queries

The prompt of RealMix is shown in Fig. 30

37

> **Prompt of RealMix**
>
> I will provide three real user questions and a reference question's type tags. Your task is to create a new question in the {domain} domain with the same tags as reference question. To ensure the question feels authentic, you should utilize real-world details drawn from the three real user questions.
>
> ### You MUST follow these steps to generate the new question:
>
> #### Task1: Real-world Details Selection:
>
> Select a few real-world details (such as the real people, objects, scenes, settings, and any other details mentioned) from three real user questions that fit the given tags. If there are no suitable details, return [[I cannot generate a question based on the provided real user questions.]] and stop.
>
> #### Task2: New Question Generation:
>
> 1. Although you should use details from real user questions, you must not mention the real user question in the new question.
> 2. The new question should be complex and challenging, requiring deep understanding and analysis of the subject. The length of the question should be at least as long as the reference question but should not be overly simplistic or repetitive. The question should be singular, not a multi-task question.
> 3. The new question must be **completely self-contained**, so that others can answer it without any additional information.
> 4. Analyze how to create the new question with chosen real-world details and provided tags. While multiple tags are available, the newly generated question only needs to align with some of them, not all. Even if the original question already fits, generate a different version.
>
> ### Output Format:
>
> `[Anylysis]`: You should first complete the anylysis of task1 and task2 here.
> `[Question]`: Summarize the newly generated question in the following format: <new_query>Insert the final new question here.</new_query>
>
> ---
>
> <|begin_of_reference_query|>
> {reference_query}
> <|end_of_reference_query|>
>
> <|begin_of_reference_query_type_tags|>
> {type_tags}
> <|end_of_reference_query_type_tags|>
>
> <|begin_of_real_user_query_1|>
> {query1}
> <|end_of_real_user_query_1|>
>
> <|begin_of_real_user_query_2|>
> {query2}
> <|end_of_real_user_query_2|>
>
> <|begin_of_real_user_query_3|>
> {query3}
> <|end_of_real_user_query_3|>

Figure 30: Prompt of RealMix.

C.3 Detailed Description of Visualization and Analysis Toolkits

Visualization. Two visualization tools are available: one organizes results by model, enabling side-by-side comparisons across multiple models; the other organizes results by query type, allowing users to examine model rankings under specific query tags.

Analysis. We have implemented four analysis tools, with more under development.

- *Automatic Weakness Reporter* presents a list of notable strengths and weaknesses detected in each model.
- *Automatic Weakness Analyzer* generates natural language summaries of a model's strengths and weaknesses.
- *Failure Mode Explorer* identifies whether a model's failure in a subtask is systematic (low ranking variance among sibling nodes) or sporadic (high variance).
- *Automatic Difference Reporter* compares multiple models and highlights significant anomalies between their performances in a structured list.

C.4 Feedbacker-D-V0

C.4.1 Query Quality: Human Evaluation

We conduct a comprehensive questionnaire study to evaluate the quality and human-likeness of automatically generated questions compared to original real-user questions. Five domain experts (four Master's holders and one PhD holder) participate in the questionnaire. The assessment protocol, shown in Fig. 31, present participants with 20 question pairs (one generated and one original) in randomized order to prevent positional bias.

Query Quality and Reality Evaluation Questionnaire

Dear Participant,
We are conducting a study on question generation for large language model. The attached Excel file contains 20 rows, each presenting two questions generated by different strategies. Please:

1. Assess the **quality** of both questions (poor quality indicators include: oversimplification, incompleteness, or unclear phrasing).
2. Evaluate whether each question appears to be **human-authored**.

Rating Scheme:

- For quality/authenticity:
    - 1 = Question 1 superior
    - 2 = Question 2 superior
    - 3 = Both inadequate
    - 4 = Both excellent

Figure 31: Questionnaire protocol for comparative evaluation.

Tab. 6 presents the detailed evaluation outcomes across all raters. Our generated questions demonstrate superior quality in 49% of cases compared to just 9% where original questions are preferred, with 42% of cases rated as ties. For authenticity assessment, generated questions are preferred in 27% of cases versus 21% for originals, with a majority (52%) considered equally authentic. Notably, 37 out of 100 cases are rated as having both excellent quality, while 47 cases are judged equally authentic.

Table 6: Query quality and reality evaluation results by human rater.

Rating	Rater 1 Quality/Reality	Rater 2 Quality/Reality	Rater 3 Quality/Reality	Rater 4 Quality/Reality	Rater 5 Quality/Reality	Total Quality/Reality
1 (Generated better)	11/5	15/8	10/3	7/7	6/4	49/27
2 (Original better)	0/4	4/10	4/7	0/0	1/0	9/21
3 (Both inadequate)	1/1	0/0	2/3	2/1	0/0	5/5
4 (Both excellent)	8/10	1/2	4/7	11/12	13/16	37/47

C.4.2 Query Quality: Data Contamination

Following Auto-Arena [19], we employ two methods to detect test data contamination. The first is the string match metric, as used in GPT-4 [71], which flags a data point as contaminated if any of three

randomly sampled 50-character substrings (or the entire string, if shorter) from the evaluation sample appears as a substring in the training set. The second method, inspired by Platypus [72], uses sentence embedding similarity: a question is considered contaminated if its BERT-Large [73] embedding has a cosine similarity above 0.8 with any training item, enabling detection of paraphrased overlaps.

Applying these methods, we find a contamination rate of 1.41% using the substring match metric, and a substantially lower rate of 0.21% using the embedding similarity metric. These results indicate that our approach is largely robust to data leakage and that potential contamination is minimal.

C.4.3 Examples of Feedbacker-D-V0

We present several samples of Feedbacker-D-V0 in Tab. 7.

Table 7: Examples from different domains in Feedbacker-D-V0.

Domain	Sample
Writing	Translate and adapt the following mission dialogue into French, ensuring it resonates with a native French speaker. The dialogue should maintain its dramatic tone while incorporating cultural nuances and idiomatic expressions suitable for a fantasy RPG setting. Additionally, optimize the sentence structure to enhance flow and engagement. "Listen carefully, adventurer. The ancient temple lies hidden beyond the Whispering Woods. Within its walls, you will find the Earth Crystal, a source of immense power. Use it wisely, for it can either restore balance or bring about destruction. Be wary of the Guardians; they will test your worth. Prove yourself, and the crystal shall be yours."
Roleplay	Please pretend to be my deceased grandfather, who was a gentle and wise artist from the small coastal town of Lestupsk. He used to live in a house near the old fortress on the hill and would often take me to ride the drowsy trams that clanked through our town. He taught me how to draw and always shared stories about his friend, Vanya, another local artist. I miss him so much, especially when I am trying to fall asleep. We begin now. Hello grandpa, I miss you so much! I am feeling very tired and am having trouble falling asleep. Can you please tell me one of those special stories about Vanya and the old trams, and maybe show me how to sketch the view of the fortress from our window? Your stories always made me feel better and helped me sleep.
Knowledge	Analyze how the infrared absorption spectrum of a greenhouse gas (e.g., CO_2 or methane) impacts the heat transfer efficiency in a thermodynamic system such as a gas turbine or a heat exchanger. Propose strategies to counteract any detrimental effects on system performance caused by such radiative properties while maintaining environmental compliance.
Coding	In Java, which method is commonly used to remove quotes from the beginning and end of a string? a) 'trim()' b) 'replaceAll("\"", "")' c) 'substring(1, str.length() - 1)' d) 'strip()' e) None of the above
Mathematics	Let $F = A'B'C' + A'B'C + A'BC' + AB'C' + ABC'$. Using Boolean algebraic laws, simplify the expression F to its minimal form. Identify which laws or manipulations you use at each step to arrive at the solution.
Reasoning	A teacher places a book inside a drawer of her desk in the classroom. She then locks the drawer and places the key inside a box on her shelf. After school, the teacher moves the box to the staff room and leaves it on a table. Later, a janitor takes the box from the staff room and places it inside a cabinet in the storage room. Where is the key, and where is the book?

C.4.4 Feedbacker-HPD

To validate our evaluation methods, we require a robust human preference dataset. We construct this dataset from seed data, which is well-suited for our purposes as it contains two responses and a corresponding human preference label for each query. We begin by randomly sampling 650 queries rated 6 or 7 from the seed data, followed by manual quality control, yielding a final set of 636 high-quality queries.

Although the seed data includes human preference labels, prior work [15] has raised concerns about their reliability due to variability in human judgment. To enhance label accuracy, we enlisted two additional graduate-level annotators and determined the final preference label via majority voting, thereby mitigating individual annotator bias. Our annotation interface is illustrated in Fig. 32.

Since our questions are all compliant and commonly seen in real-world scenarios, they do not pose significant potential participant risks. Additionally, we pay 300 yuan per day for each participant, and we ultimately spent 1,200 yuan.

The distribution of Feedbacker-HPD is shown in Fig. 33. Similar to Feedbacker-D-V0, these queries span six domains: writing, roleplay, knowledge, coding, mathematics, and reasoning.
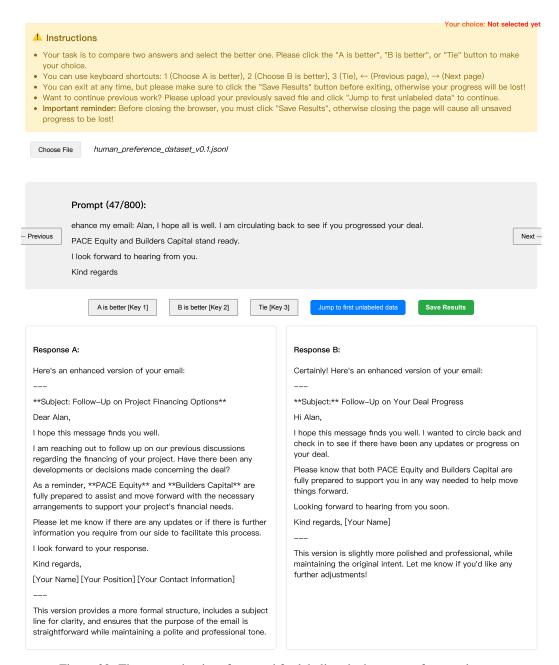

Figure 32: The annotation interface used for labeling the human preference dataset.
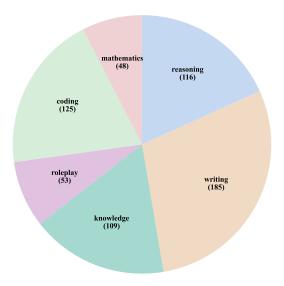
Figure 33: Domain distribution of Feedbacker-HPD.

D   Evaluation Methods

Example (1) of Pairwise Evaluation

(prompt and thinking process) ...

`[[A]]`

Explanation:
- **Correctness:** Assistant A's answer directly addresses the user's requirement to track the *number of open tabs* on the same domain using 'localStorage'. It provides a straightforward 'TabTracker' class with methods to increment/decrement the count and store it in 'localStorage'. While it lacks automatic triggering of decrement on tab closure (relying on manual calls in the example), it correctly structures the core logic for counting tabs. The problem does not explicitly require automatic event handling, so the code is technically correct for the stated goal.

- **Helpfulness:** The code is concise and focused on the core requirement. The explanation clearly describes how each method works and how to use the class. The example usage demonstrates the basic workflow, making it easy to integrate into a project.

- **Assistant B's Issues:** While B's approach is more detailed (tracking individual tabs with IDs and URLs), it diverges from the user's request to track *open tabs as a count*. B's implementation requires managing individual tabs and resolving complex edge cases (e.g., identifying which tab closed), which complicates the task. Additionally, B's '_handleTabClose' method admits it cannot reliably remove tabs, making the solution incomplete for the stated problem.

- **Key Flaw in B's Answer:** The user asked for a *count* of open tabs, not detailed tracking of individual tabs. B's answer overcomplicates the solution by focusing on tab metadata rather than the numeric count.

Conclusion: Assistant A's answer is better because it aligns directly with the user's requirement, provides a functional core, and is simpler and more relevant. Assistant B's answer, while thorough, solves a different problem (tracking individual tabs) and introduces unnecessary complexity.

Figure 34: Example (1) of Pairwise Evaluation.

Example (2) of Pairwise Evaluation

(prompt and thinking process) ...

`[[B]]`

Explanation:
Both assistants provided valid HTML documents with step-by-step instructions for jump-starting a car. However, **Assistant B's answer is better** for the following reasons:

1. **Self-Contained Diagram:**
- **Assistant B** uses an embedded SVG diagram directly in the HTML, fulfilling the user's requirement for an "all in one document." The SVG illustrates the connection process without relying on an external image file.
- **Assistant A** references an external image ('car_battery_jumpstart.png'), which requires the user to provide or replace the image path. This makes the document incomplete as-is.

2. **Step-by-Step Clarity:**
- **Assistant B** breaks down the steps into a more granular 8-step process, explicitly separating each cable connection into distinct steps (e.g., connecting each end of the red and black cables separately). This enhances clarity for users unfamiliar with the process.
- **Assistant A** combines some steps (e.g., all cable connections under step 3 with sub-steps), which is concise but slightly less explicit.

3. **Safety and Correctness:**
- Both answers include correct procedural steps (e.g., grounding the negative cable on the dead car's metal surface instead of the terminal).
- **Assistant B** adds a final step (step 8) explicitly detailing the order to remove cables, which is important for safety.

4. **HTML Structure:**
- **Assistant B's** use of SVG ensures the diagram is part of the HTML document, whereas **Assistant A's** reliance on an external image breaks the "all in one" requirement.

Verdict: While both answers are correct, **Assistant B's** self-contained implementation of the diagram and clearer step breakdown make it the superior choice.

Figure 35: Example (2) of Pairwise Evaluation.