SCAN: Structured Capability Assessment and Navigation for LLMs

Anonymous ACL submission

Abstract

Evaluating Large Language Models (LLMs) has become increasingly important, with automatic evaluation benchmarks gaining prominence as alternatives to human evaluation. While existing research has focused on approximating model rankings, such benchmarks fail to provide users and developers with a comprehensive and fine-grained understanding of a specific model's capabilities. To fill this gap, we propose **SCAN** (Structured Capability Assessment and Navigation), a practical framework that enables detailed characterization of LLM capabilities through comprehensive and fine-grained evaluation. SCAN incorporates four key components: (1) TaxBuilder, which extracts capability-indicating tags from extensive queries to construct a hierarchical taxonomy automatically; (2) RealMix, a query synthesis and filtering mechanism that ensures sufficient evaluation data for each capability tag; (3) a suite of visualization and analysis tools that facilitate efficient navigation and analysis of model capabilities; and (4) a PC^2-based (Pre-Comparison-derived Criteria) LLM-as-a-Judge approach that achieves significantly higher accuracy compared to classic LLM-as-a-Judge method. Using SCAN, we conduct a comprehensive evaluation of 21 mainstream LLMs. Our detailed analysis of the GPT-OSS family reveals substantial performance variations, even within sub-capabilities belonging to the same category of capability. This finding highlights the importance of fine-grained evaluation in accurately understanding LLM behavior. Project homepage and resources are available at https://anonymous-scan.github.io/SCAN/.

1 Introduction

Evaluating large language models (LLMs) has become increasingly important as these models advance in various capabilities (Wang et al., 2023; Lin et al., 2024; White et al., 2024; Li et al., 2024c,f; Kim et al., 2025; Qin et al., 2024; Liu et al., 2024b; Gu et al., 2024a; Saranathan et al., 2024; Habba et al., 2025; van Schaik and Pugh, 2024; You et al., 2024; Chiang et al., 2024). Among them, Chatbot Arena (Chiang et al., 2024), 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 (Zheng et al., 2023; Dubois et al., 2024; Ni et al., 2024; Li et al., 2024e; Zhao et al., 2024; Liu et al., 2024b), which uses a small amount of data and LLM-as-a-Judge (Gu et al., 2024a; Li et al., 2024b; Ashktorab et al., 2024; Tseng et al., 2024; Son et al., 2024) to approximate the model rankings in Chatbot Arena. These automatic evaluation benchmarks have successfully established a paradigm focused on approximating the rankings of multiple models.

However, when users and developers aim to gain a comprehensive and detailed characterization of an LLM's capabilities, such a paradigm fails to provide the necessary breadth and granularity. To address this critical gap in LLM evaluation, we need an approach that is both comprehensive in scope and fine-grained in detail. In this paper, we thoroughly discuss the challenges of constructing a comprehensive and fine-grained evaluation framework and present our solutions, which are integrated into SCAN (Structured Capability Assessment and Navigation), a practical framework that effectively addresses these challenges. Specifically, SCAN incorporates four key designs to systematically establish such an evaluation framework. (1) To enable comprehensive and fine-grained evaluation, we propose TaxBuilder, which extracts capability-indicating tags (e.g., *Python programming*, *physics Q&A*) from a large volume of real-world human queries. These tags span a broad

range of domains, scenarios, and tasks, thereby ensuring wide coverage of diverse user needs. (2) To facilitate efficient inspection of fine-grained performance, TaxBuilder organizes queries into a hierarchical taxonomy, which is further visualized through a dedicated visualization tool (available on our project page). Together, these components allow users to quickly navigate and analyze model capabilities in a structured and intuitive manner. (3) The evaluation validity of a specific tag would be unreliable if it is only supported by limited assessment data, which is particularly problematic for tags in the long-tail distribution. To ensure each tag is evaluated with sufficient queries, we develop RealMix, which obtains realistic and high-quality queries through synthesis and filtering. RealMix builds upon TaxBuilder, thereby enhancing both the coverage and diversity. (4) Finally, to guarantee evaluation accuracy while maintaining scalability, we propose a novel pointwise LLM-as-a-Judge based on PC^2 (Pre-Comparison-derived Criteria). This method significantly improves the reliability and accuracy of automatic judgments while avoiding the prohibitive costs of pairwise evaluation.

To demonstrate the effectiveness of SCAN, we evaluate 21 mainstream LLMs and conduct in-depth analyses of their results. A key value proposition of SCAN is its ability to quickly provide comprehensive and detailed understanding of newly released LLMs. To this end, we conducted a detailed analysis of the recently released GPT-OSS family using SCAN. Our analysis reveals several interesting findings: (1) In terms of overall ranking, GPT-OSS-120B achieves superior performance, surpassing Deepseek-R1, while GPT-OSS-20B shows relatively weak performance, falling below Qwen3-8B. (2) Despite its overall strong performance, GPT-OSS-120B exhibits limited capabilities in writing and roleplay domains. Conversely, while GPT-OSS-20B generally underperforms, it demonstrates leading capabilities in the coding domain. (3) Although the GPT-OSS models show strong coding abilities overall, a fine-grained analysis by programming language reveals significant variations: they excel in JavaScript but lag behind in C and Java. (4) In the knowledge domain, while the GPT-OSS models perform poorly overall, our fine-grained analysis reveals substantial performance variations across different sub-domains, indicating inconsistent coverage of domain-specific knowledge during training. These fine-grained insights highlight the effectiveness and potential of SCAN.

In summary, our main contributions are three-fold:

- We propose shifting evaluation paradigm from rankings to model understanding. To this end, we introduce SCAN, which facilitates comprehensive understanding of LLM capabilities (strengths and weaknesses) through fine-grained evaluation.
- 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 SCAN-T-V0 (Fig. 2) and SCAN-D-V0 (Tab. 1), respectively.
- To showcase the usage of SCAN, we evaluate 21 mainstream LLMs and conduct in-depth analyses of the results. Through detailed analysis of the GPT-OSS family models, we uncover several fine-grained insights, thereby demonstrating the effectiveness and potential of SCAN.

2 SCAN

The overview of SCAN 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

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 (OpenAI, 2024a)) to annotate query tags for a large set of real user queries [1]. 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

[1] We utilize the dataset made available by the Chatbot Arena team, accessible at https://www.kaggle.com/competitions/wsdm-cup-multilingual-chatbot-arena/data.

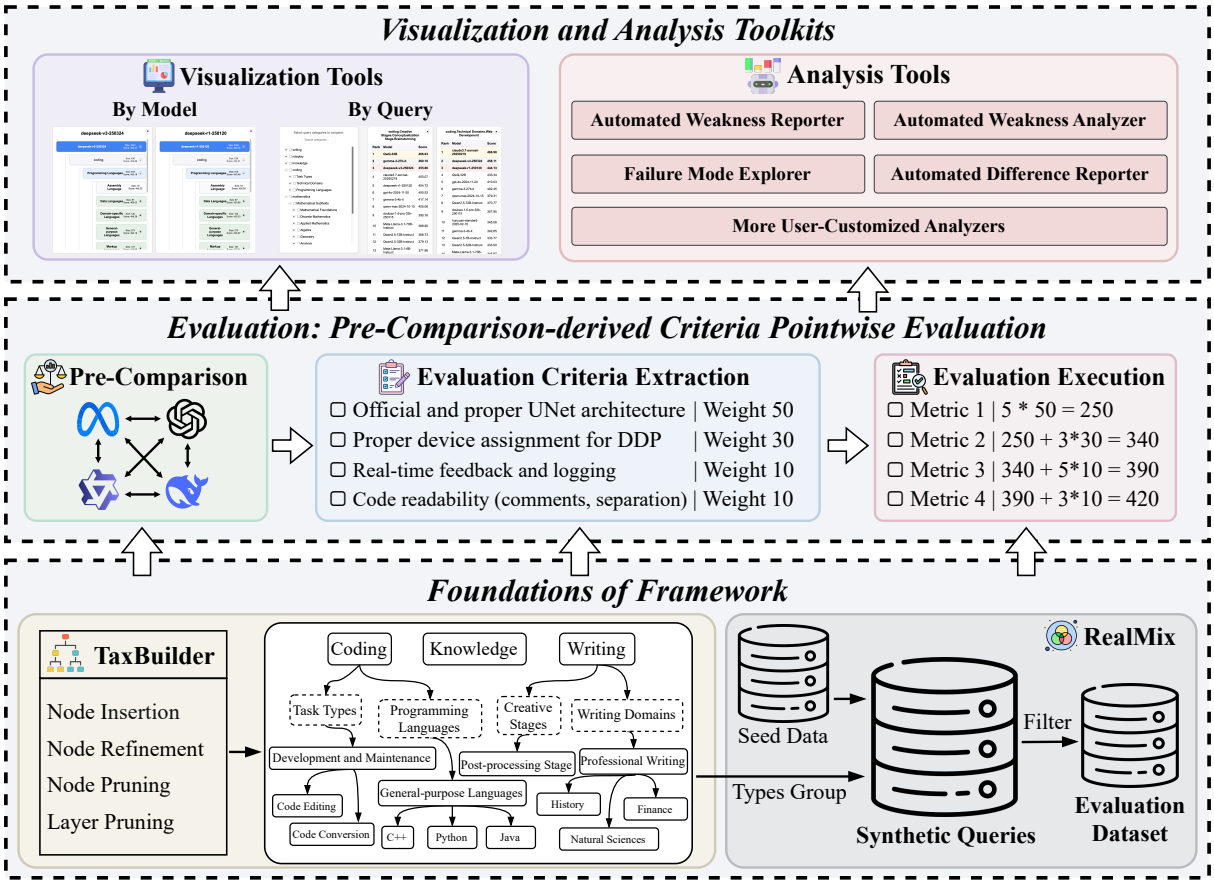

Figure 1: An overview of SCAN framework.

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.

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:

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

where $T^{c}_{cur} = \{T^{c_1}_{cur}, T^{c_2}_{cur}, \ldots, T^{c_m}_{cur}\}$ 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^{c}_{cur}. The prompt for DM_{ins} is presented in Fig. 13. It will return one of three possible relationship types:

- **<E> (Exists):** This indicates that the node tg already exists in T^{c}_{cur} 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^{c}_{cur}. Consequently, T^{c}_{cur} is updated to include tg, i.e., $T^{c}_{cur} = \{T^{c_1}_{cur}, T^{c_2}_{cur}, \ldots, T^{c_m}_{cur}\} \cup \{tg\}$.
- **$\mathbf{T^{c_i}_{cur}}$ (Child Node):** This implies that tg should be a child of the specific node $T^{c_i}_{cur}$. In this scenario, a recursive insertion is performed.

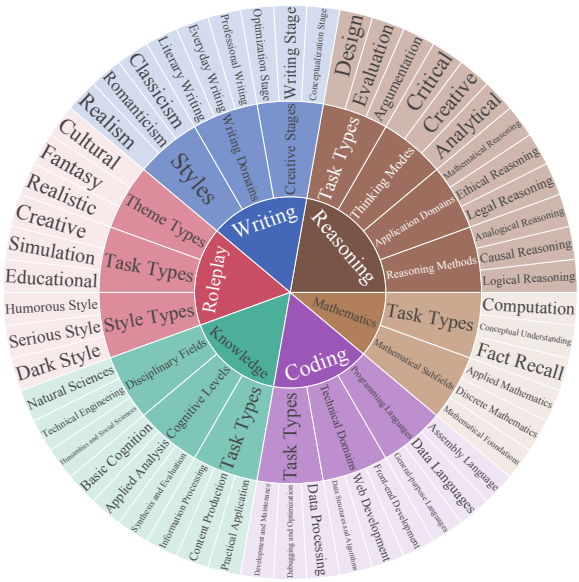

Figure 2: A subset of SCAN-T-V0. Please refer to project page for complete taxonomy.

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. 14), 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 4 layers. Nodes beyond the 4 layer are retained only if they have 5 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 SCAN-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

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 (Liu et al., 2024a, 2025b) 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 (Team, 2025b; Yang et al., 2024a) to annotate a set of real-user queries (lin Chiang et al., 2024) 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 SCAN-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 SCAN-T-V0 since some combinations do not exist in practice. To mitigate model bias, we employ multiple query generation models (Deepseek-V3 (DeepSeek-AI, 2024), gpt-4o, and doubao-1.5pro (ByteDance, 2024)). More details are provided in § C.2.2.

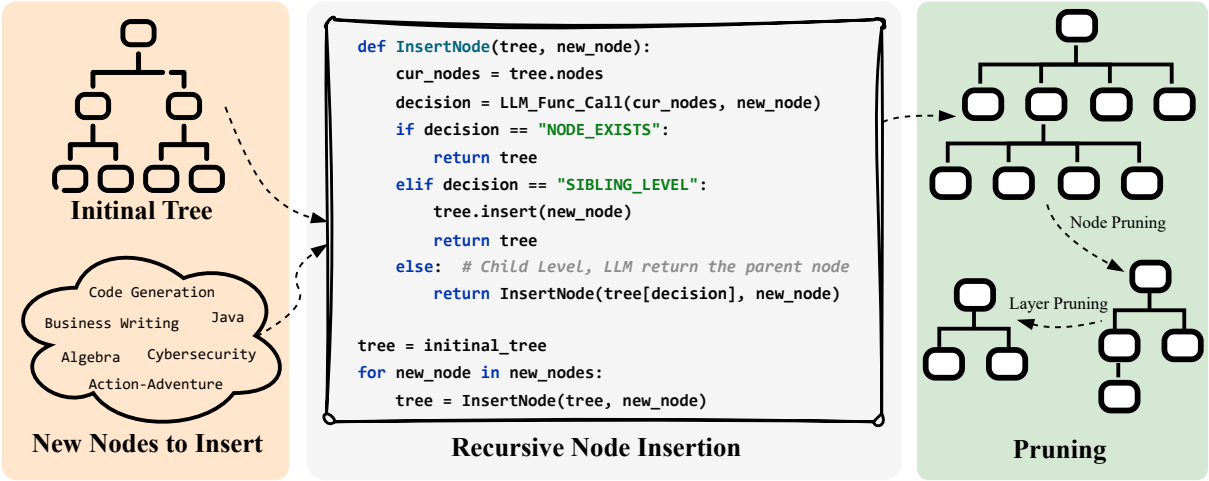

Figure 3: An overview of TaxBuilder, a tree-based automatic taxonomy generation.

Verification and Filtration. We validate both the tags and the quality of generated query using the same procedure applied to the seed data, and discard 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 (DeepSeek-AI, 2025), and o1-mini (OpenAI, 2024b)).

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 SCAN-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 SCAN-D-V0. We ensure that the sample size for each tag in SCAN-T-V0 is at least 19 (#MinNum) to guarantee the statistical representativeness of the evaluation results.

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.

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

Table 1: Statistics of SCAN-D-V0.

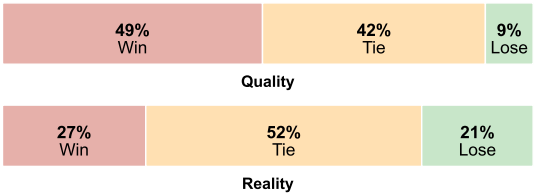

Figure 4: Comparison of RealMix and real user queries, judged by five human evaluators.

4 Pre-Comparison-derived Criteria-based LLM-as-a-Judge

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

Naive Pointwise Evaluation. Naive pointwise evaluation (Kim et al., 2023; Ye and Ng, 2024) refers to scoring each response independently. Judge model J is tasked to assign a score to a single response y with a naive pointwise evaluation prompt p_o.

$$J(y|x, p_o) \rightarrow s \in \mathbb{R}. \quad (2)$$

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 (Li et al., 2024c; Zheng et al., 2023).

Pairwise Evaluation. Pairwise evaluation (Cao et al., 2024; Hu et al., 2024) compares two responses to determine which is better. According to prior studies (Zheng et al., 2023; Li et al., 2024c), while this approach typically produces more reliable results than pointwise evaluation, it has a quadratic time complexity O(n^2) that becomes impractical as the number of tested models increases. Therefore, we do not adopt it in our work.

4.2 Methodology

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:

$$J(\{y^1, y^2, \ldots, y^n\} \mid x, p_c, p_w) \rightarrow$$
$$C = \{c_1, c_2, \ldots, c_m\},\ W = \{w_1, w_2, \ldots, w_m\} \quad (3)$$

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.

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

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

where p_c represents our criteria-weighted evaluation prompt and s is the final score. For the baseline answer y_b, we first evaluate it without using any baseline answer, then concatenate the evaluation results with p_c to assess other models. This operation helps improve the stability of multiple evaluations.

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 demonstrate the main experimental results. The experiment details are shown in § B.1. Results on more benchmark and ablation results are shown in § A.1.

As shown in Tab. 2, naive pointwise evaluation yields relatively low accuracy across all tested LLMs, confirming its limited reliability as a judging approach. Introducing metric decomposition markedly improves performance, with direct decomposition and single-model pre-comparison offering moderate gains, and diverse-model pre-comparison delivering more substantial improvements. Building upon this, our method further incorporates baseline answers, leading to consistent and significant accuracy gains over all baselines. These results demonstrate that the combination of criteria decomposition, diverse-model pre-comparison, and baseline guidance forms a robust and generalizable evaluation strategy.

Method / Model	Accuracy
Deepseek-R1	
naive	0.5694
direct metric decomposition	0.6134
metric decomposition (single model)	0.5974
metric decomposition (diverse model)	0.6466
ours	**0.6962**
Qwen3-32B	
naive	0.5181
ours	**0.6535**
claude-3.7-sonnet	
naive	0.5959
ours	**0.7453**
gpt-4.1	
naive	0.6116
ours	**0.7201**

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

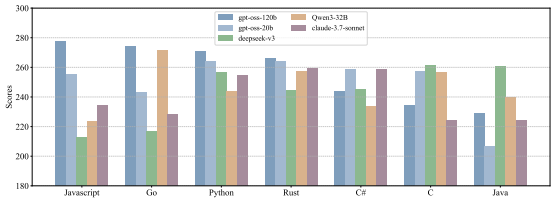

Figure 5: Fine-grained performance comparison on `coding.Programming Languages.General-purpose Languages`.

6 Results on SCAN

6.1 Overview of Evaluation Results

To utilize SCAN for analysis, we first employ SCAN-D-V0 to evaluate 21 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 SCAN: GPT-OSS as Examples

Upon the release of a new model, our SCAN framework facilitates a rapid, comprehensive evaluation of its capabilities across diverse domains. Here, we demonstrate SCAN's utility through an analysis of the recently released GPT-OSS family models.

Overview. The overall evaluation results summarized in Tab. 8 show that GPT-OSS-120B achieves the highest aggregate score, reflecting its strong overall performance. However, it ranks only 7th in roleplay domain, highlighting a notable weakness obscured by the composite metric. Crucially, our framework reveals that GPT-OSS-20B, despite a lower overall rank, performs exceptionally in coding domain (ranking 2nd), which demonstrates its specialized strength. This underscores the effectiveness of our approach in uncovering fine-grained capability profiles and identifying performance that holistic scores alone cannot capture.

Fine-grained Coding Performance Analysis. Our SCAN framework enables decomposition of the aggregate coding score by programming language, allowing evaluation of model performance across specific general-purpose languages such as C and Java. The results shown in Fig. 5 reveal substantial performance variation across languages. Notably, GPT-OSS-120B performs strongly in Python, JavaScript, Go, and Rust, but shows weaker performance in C and Java—areas where Deepseek-V3 achieves superior results. This level of granularity is essential for applications demanding language-specific usage. Furthermore, the analysis uncovers the "spiky" capability profile of GPT-OSS-20B: it ranks 2nd in Python and Rust, and outperforms its larger counterpart, GPT-OSS-120B, in both C and C#. This demonstrates that smaller models can excel in specific tasks, providing a more efficient option for domain-specific development. SCAN's fine granularity thus facilitates informed, use-case-driven model selection, advancing beyond holistic rankings toward nuanced capability profiling.

Diagnosing Knowledge Instability through Fine-Grained Analysis. Here, we demonstrate the use of our analytical tool, Failure Mode Explorer (see § C.3). In Fig. 6, we present the tool's automatically identified 4 parent nodes with the largest performance variance across sub-nodes for GPT-OSS-120B. Interestingly, all four are within the knowledge domain. Combined with the observation from Tab. 8 that GPT-OSS-120B's overall performance in knowledge is unexceptional, this motivates a deeper investigation into its capabilities within this domain.

While GPT-OSS-120B ranks third overall in the knowledge domain, a closer look reveals that its performance is far from uniformly strong. In GPT-OSS-120B's case, the internal performance distribution is highly uneven. For example, within the technical engineering field (parent node ranked #2), the model achieves top positions in computer science and aerospace engineering (#1), yet falls sharply to #11 in bioengineering. This demonstrates that, while aggregate scores facilitate macro-level comparisons, they may conceal important

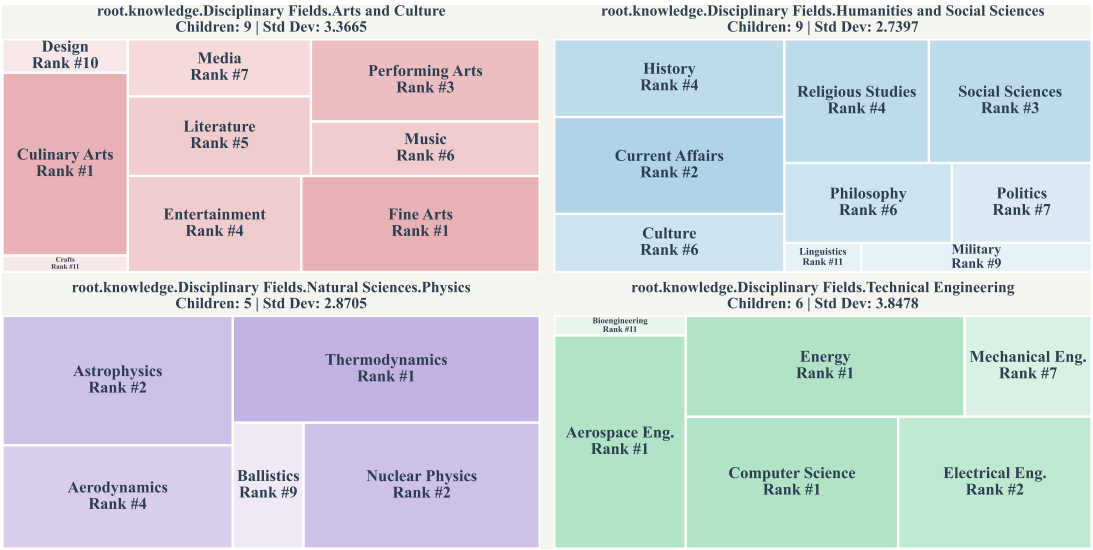

Figure 6: The top 4 parent nodes exhibiting the largest performance variance across child nodes, automatically identified by our Failure Mode Explorer tool. The symbol '#' denotes rankings.

weaknesses in a model's capabilities.

To tackle these hidden weaknesses, our SCAN framework converts coarse-grained aggregate metrics into fine-grained diagnostic signals that expose exactly where the gaps lie. In the case above, instead of settling for a general verdict like "the model's knowledge capability is weak," SCAN can pinpoint the specific sub-domains—such as bioengineering—that are underperforming relative to other strengths. It is worth noting that these results were automatically detected by our Failure Mode Explorer tool, further validating SCAN's effectiveness in systematically revealing hidden problems.

7 Related Work

Automatic Evaluation Benchmark. We review existing benchmarks here, focusing on their motivations, dataset designs, and evaluation methods. AlpacaEval-2.0 (Dubois et al., 2024) uses 805 curated general-domain data points to assess instruction-following with win rate metric. MT-Bench (Zheng et al., 2023) includes 80 high-quality queries across 8 domains. It adopts pointwise evaluation on a 1–10 scale. Arena-Hard (Li et al., 2024e) filters 500 challenging Chatbot Arena datasets and uses win rate metric for evaluation. LiveBench (White et al., 2024) manually updates queries in real-time with rule-based scoring to reduce LLM biases. Auto-Arena (Zhao et al., 2024) employs pairwise debates for Elo-based ranking, with a 40-query evaluation set. WildBench (Lin et al., 2024) 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

In this paper, we propose shifting the focus of evaluation from only rankings and leaderboards to providing valuable insights. To this end, we introduce SCAN, an extensible evaluation framework featuring comprehensive and fine-grained queries, along with tools for visualization and analysis. Additionally, our proposed PC^2-based LLM-as-a-Judge achieves superior accuracy compared to classic evaluation methods. Finally, through extensive experiments, we demonstrate that utilizing SCAN enables the convenient identification of a model's strengths and weaknesses, thereby providing valuable insights.

In the future, we believe SCAN will provide valuable insights to the community in the following aspects. (1) For model evaluation, SCAN helps prevent one-sided understanding of a model's capabilities, as its broad coverage across diverse categories. (2) For model training, SCAN 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. (3) For data synthesis, TaxBuilder offers a scalable taxonomy that empowers developers to generate comprehensive, diverse, and complex synthetic data for model training.

Limitations

Currently, our framework is designed primarily for language models, and does not yet provide support for multimodal models (Li et al., 2024d; Patraucean et al., 2023; Li et al., 2024a; Roberts et al., 2025; Liang et al., 2021). In future work, we aim to extend the framework to include multimodal capabilities through a dedicated module, tentatively named SCAN-Anything.

In addition, the current implementation does not encompass evaluation dimensions related to AI mechanisms such as safety, honesty, factuality, hallucination, and fairness (Ren et al., 2024; Röttger et al., 2025; Gu et al., 2024b; Jin et al., 2024; Wang et al., 2024; Liu et al., 2025a; Luo et al., 2024; Ravi et al., 2024; Ding et al., 2024; Huang et al., 2023). Nevertheless, the framework is inherently extensible and can be readily adapted to incorporate these aspects as additional evaluation domains in subsequent work.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, and 1 others. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, and 1 others. 2024. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*.

Anthropic. 2025. Claude 3.7 sonnet and claude code. Accessed: 2025-05-05.

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

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

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

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, and 1 others. 2024. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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.

Peng Ding, Jingyu Wu, Jun Kuang, Dan Ma, Xuezhi Cao, Xunliang Cai, Shi Chen, Jiajun Chen, and Shujian Huang. 2024. 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.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, and 1 others. 2024a. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Tianle Gu, Zeyang Zhou, Kexin Huang, Liang Dandan, Yixu Wang, Haiquan Zhao, Yuanqi Yao, Yujiu Yang, Yan Teng, Yu Qiao, and 1 others. 2024b. Mllmguard: A multi-dimensional safety evaluation suite for multimodal large language models. *Advances in Neural Information Processing Systems*, 37:7256–7295.

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

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

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

Kexin Huang, Xiangyang Liu, Qianyu Guo, Tianxiang Sun, Jiawei Sun, Yaru Wang, Zeyang Zhou, Yixu Wang, Yan Teng, Xipeng Qiu, and 1 others. 2023. Flames: Benchmarking value alignment of chinese large language models. *CoRR*.

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

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and 1 others. 2023. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*.

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, and 13 others. 2025. The BiGGen bench: A principled benchmark for fine-grained evaluation of language models with language models. In *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. Association for Computational Linguistics.

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

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

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, and 1 others. 2024b. From generation to judgment: Opportunities and challenges of llm-as-a-judge. *arXiv preprint arXiv:2411.16594*.

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

Jian Li, Weiheng Lu, Hao Fei, Meng Luo, Ming Dai, Min Xia, Yizhang Jin, Zhenye Gan, Ding Qi, Chaoyou Fu, and 1 others. 2024d. A survey on benchmarks of multimodal large language models. *arXiv preprint arXiv:2408.08632*.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. 2024e. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*.

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

Yanyang Li, Jianqiao Zhao, Duo Zheng, Zi-Yuan Hu, Zhi Chen, Xiaohui Su, Yongfeng Huang, Shijia Huang, Dahua Lin, Michael R Lyu, and 1 others. 2023. Cleva: Chinese language models evaluation platform. *arXiv preprint arXiv:2308.04813*.

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

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

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

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

Decheng Liu, Zongqi Wang, Chunlei Peng, Nannan Wang, Ruimin Hu, and Xinbo Gao. 2025a. 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.

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

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

Xiao Liu, Xuanyu Lei, Shengyuan Wang, Yue Huang, Andrew Feng, Bosi Wen, Jiale Cheng, Pei Ke, Yifan Xu, Weng Lam Tam, and 1 others. 2024b. 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.

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

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah A Smith, Hannaneh Hajishirzi, and Nathan Lambert. 2025. Rewardbench 2: Advancing reward model evaluation. *arXiv preprint arXiv:2506.01937*.

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

OpenAI. 2024a. Hello gpt-4o. https://openai.com/index/hello-gpt-4o/. Accessed: 2025-04-30.

OpenAI. 2024b. Openai o1 mini: Advancing cost-efficient reasoning. https://platform.openai.com/docs/models/o1-mini. Accessed: 2025-04-30.

OpenAI. 2025. gpt-oss-120b and gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Viorica Patraucean, Lucas Smaira, Ankush Gupta, Adria Recasens, Larisa Markeeva, Dylan Banarse, Skanda Koppula, Mateusz Malinowski, Yi Yang, Carl Doersch, and 1 others. 2023. Perception test: A diagnostic benchmark for multimodal video models. *Advances in Neural Information Processing Systems*, 36:42748–42761.

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

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

Richard Ren, Steven Basart, Adam Khoja, Alice Gatti, Long Phan, Xuwang Yin, Mantas Mazeika, Alexander Pan, Gabriel Mukobi, Ryan Kim, and 1 others. 2024. Safetywashing: Do ai safety benchmarks actually measure safety progress? *Advances in Neural Information Processing Systems*, 37:68559–68594.

Jonathan Roberts, Mohammad Reza Taesiri, Ansh Sharma, Akash Gupta, Samuel Roberts, Ioana Croitoru, Simion-Vlad Bogolin, Jialu Tang, Florian Langer, Vyas Raina, and 1 others. 2025. Zerobench: An impossible visual benchmark for contemporary large multimodal models. *arXiv preprint arXiv:2502.09696*.

Paul Röttger, Fabio Pernisi, Bertie Vidgen, and Dirk Hovy. 2025. 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.

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

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

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Qwen Team. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Qwen Team. 2025b. Qwq-32b: Embracing the power of reinforcement learning.

Tencent. 2025. Tencent hunyuan. Accessed: 2025-05-05.

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

Tempest A van Schaik and Brittany Pugh. 2024. 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.

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

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

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

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, and 1 others. 2024. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*.

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, and 23 others. 2024a. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024b. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

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

Jiaxuan You, Mingjie Liu, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. 2024. 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.

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

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

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. 2025. Rmb: Comprehensively benchmarking reward models in llm alignment. In *The Thirteenth International Conference on Learning Representations*.

Appendices Contents

A More Discussion **14**
A.1 Distinctions from Prior Works Enabled by Fine-Grained Dataset 14

B Experiments **14**
B.1 Experimental Setup for Evaluation Method 14
B.2 More Experiment on PC^2-Pointwise Evaluation 15
B.3 Experimental Setup for Evaluation and Leaderboard 16
B.4 Leaderboard 16

C Taxonomy and Dataset **19**
C.1 TaxBuilder 19
C.1.1 Manually Initialized Basic Taxonomy Tree 19
C.1.2 Initial Annotation for Query Type Tags 19
C.1.3 Algorithm for TaxBuilder 19
C.1.4 Prompts for TaxBuilder 20
C.2 RealMix 20
C.2.1 Seed Data 20
C.2.2 Synthesize New Queries 20
C.3 Detailed Description of Visualization and Analysis Toolkits 20
C.4 SCAN-D-V0 23
C.4.1 Query Quality: Human Evaluation 23
C.4.2 Query Quality: Data Contamination 23
C.4.3 Examples of SCAN-D-V0 24
C.4.4 SCAN-HPD 25

D Evaluation Methods **26**

A More Discussion

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

Several existing studies, such as SELF-INSTRUCT (Wang et al., 2022), Nemotron-4-340B (Adler et al., 2024), WildBench (Lin et al., 2024), CLEVA (Li et al., 2023), and C^2LEVA (Li et al., 2024f), provide fine-grained evaluation of LLM capabilities. However, their taxonomies are relatively coarse and lack extensibility. For instance, C^2LEVA 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 (Zeng et al., 2025), 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.

B Experiments

B.1 Experimental Setup for Evaluation Method

Models. The PC^2 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 (OpenAI, 2024a), doubao-1-5-pro-32k-250115 (ByteDance, 2024), and Deepseek-V3-250324 (DeepSeek-AI, 2024) to ensure the diversity of the responses. For judge model, we adopt Deepseek-R1 (DeepSeek-AI, 2025) due to its superior reasoning performance. Unless otherwise specified, we use the officially recommended decoding parameters.

Dataset. All our experiments are based on our pairwise human preference datasets SCAN-HPD, RewardBench-v2 (Malik et al., 2025) and RMB (Zhou et al., 2025). Pairwise human preference datasets usually contain a prompt and two responses, and the model's task is to determine which response is better. In pointwise evaluation, the judge model assigns scores to both responses, with the higher-scoring response being chosen as the winner.

SCAN-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 SCAN-HPD is provided in § C.4.4. RewardBench-v2 is a widely used benchmark for evaluating judge models, covering several domains: factuality, precise instruction following, math, safety, and focus. It is commonly used to measure and compare the performance of different language models. RMB is another 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, SCAN-HPD, which will be publicly released alongside SCAN-D-V0. Results on RewardBench-v2 and RMB are presented only in § B.2.

Baseline. We employ naive pointwise evaluation as main baseline. Additionally, we also tested the impact of various components on P^2C pointwise evaluation. Direct metric decomposition refers to generating query-specific criteria based only on the query, without the process of pre-comparison (Li et al., 2024g; Qin et al., 2024; Gupta et al., 2024). Metric decomposition (single model) refers to using only gpt-4o with a temperature of 1, generating three responses. Metric decomposition (diverse model) involves using gpt-4o, doubao-1-5-pro, and Deepseek-V3 as models for generating auxiliary responses. Metric decomposition (diverse model) does not adopt the baseline guidance mechanism. For the baseline answer, we use gpt-4o-2024-11-

Dataset	#Queries	Cont.-free	Fin.Tag	LongTail	Update
AlpacaEval2.0	805	×	×	×	×
MT-Bench	80	×	×	×	×
Arena-Hard	500	×	×	×	×
Auto-Arena	40	✓	×	×	✓
LiveBench(2024-11-25)	300	✓	×	×	✓
MixEval	4000	×	×	×	×
MixEval-Hard	1000	✓	×	×	×
WildBench	1024	×	×	×	×
C^2LEVA	16115	✓	✓	×	×
Ours	3343	✓	✓	✓	✓

Table 3: Comparison of LLM alignment benchmark datasets. "Real." denotes real-world queries, "Cont.-free" means contamination-free, "Fin.Tag" indicates fine-grained tagging.

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 naive 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. 7, the prompt for evaluation execution is shown in Fig. 8, the prompt for evaluation execution with baseline answer is shown in Fig. 9. For pairwise evaluation and naive pointwise evaluation, we directly use the prompts from MT-Bench (Zheng et al., 2023). For direct metric decomposition, the corresponding prompt is displayed in Fig. 10.

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

More Benchmarks. To further validate the effectiveness of our method, we conduct experiments on additional benchmarks, namely RewardBench-v2 (Malik et al., 2025) and RMB (Zhou et al., 2025). Both benchmarks adopt the Best-of-N evaluation paradigm, which requires judge model selecting the best answer among multiple candidates, making the task inherently more difficult. The experimental results are reported in Tab. 4. As can be seen, our method consistently improves performance across all benchmarks, thereby validating its broad effectiveness.

Ablation Study. To gain deeper insights into the factors that contribute to the performance of our method, we further conduct extensive ablation studies along three dimensions: (*i*) pre-com model combination, (*ii*) number of pre-com models, and (*iii*) order of pre-com models. The results are summarized in Tab. 5, 6, and 7, respectively.

	naive	ours
RMB	0.4068	0.5969
RewardBench-v2	0.5551	0.5869

Table 4: Evaluation accuracy results on RewardBench-v2 and RMB.

(1) Ablation on model combination. For model combination (Tab. 5), we observe that multi-model setups consistently outperform the single-model baseline (`gpt-4o`), confirming that a single model is insufficient for robust judging performance. However, the differences among various multi-model combinations are relatively small, indicating that the *diversity* of the incorporated models, rather than the different combinations of models, is the key factor that drives performance improvements.

(2) Ablation on number of models. As shown in Tab. 6, increasing the number of models does not always lead to better performance. The best result is obtained when using 3 models, whereas adding more models sometimes introduces redundancy or noise, resulting in a slight drop in accuracy.

(3) Ablation on order of models. Finally, we investigate the effect of model ordering (Tab. 7). The results show that different ordering permutations lead to only marginal differences in accuracy, suggesting that the impact of model order is relatively

PC^2 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 7: Prompt for pre-comparison-derived criteria extraction.

limited. This indicates that our approach is robust to changes in execution sequence, and the overall performance is primarily determined by the chosen set of models rather than their specific order.

These findings provide practical guidance for constructing and optimizing multi-model judging systems, highlighting the importance of not only choosing the right set of models but also carefully designing their arrangement.

B.3 Experimental Setup for Evaluation and Leaderboard

Models. We select 21 mainstream LLMs for evaluation. For closed-source models, we select claude3.7-sonnet-20250219 (Anthropic, 2025), gpt-4o-2024-11-20 (OpenAI, 2024a), qwen-max-2024-10-15 (), doubao-1-5-pro-32k-250115 (ByteDance, 2024), and hunyuan-standard-2025-02-10 (Tencent, 2025). For open-source models, we select Deepseek-V3-250324 (DeepSeek-AI, 2024), Deepseek-R1-250120 (DeepSeek-AI, 2025), QwQ-32B (Team, 2025b), Gemma-3-27B-IT (Team et al., 2025), Qwen2.5-72B-Instruct (Yang et al., 2024b), Gemma-3-4B-IT (Team et al., 2025), Qwen2.5-32B-Instruct (Yang et al., 2024b), Meta-Llama-3.1-70B-Instruct (Grattafiori et al., 2024), Qwen2.5-7B-Instruct (Yang et al., 2024b), Meta-Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Mistral-7B-Instruct-v0.3 (), and Phi-4-Mini-Instruct (Abdin et al., 2024). Additionally, we evaluate recent open-source models: GPT-OSS-20B (OpenAI, 2025), GPT-OSS-120B (OpenAI, 2025), Qwen3-8B (Team, 2025a), and Qwen3-32B (Team, 2025a). 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

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 8: Prompt for evaluation execution.

Model Combination	ACC
single model (`gpt-4o`)	0.5974
gpt-4o + doubao-pro-1.5-32k + Deepseek-V3	0.6962
gpt-4o + doubao-pro-1.5-32k + Qwen2.5-7B-Instruct	0.6904
gpt-4o + doubao-pro-1.5-32k + Meta-Llama-3.1-8B-Instruct	0.6747
gpt-4o + doubao-pro-1.5-32k + Phi-4-mini-instruct	**0.7077**
gpt-4o + Qwen2.5-7B-Instruct + Meta-Llama-3.1-8B-Instruct	0.6825
gpt-4o + Qwen2.5-7B-Instruct + Phi-4-mini-instruct	0.6574
Qwen2.5-7B-Instruct + Phi-4-mini-instruct + Meta-Llama-3.1-8B-Instruct	0.6699

Table 5: Ablation on model combination.

Number of Models	Model Combination	ACC
2	gpt-4o + doubao-pro-1.5-32k	0.6699
3	gpt-4o + Deepseek-V3 + doubao-pro-1.5-32k	**0.6962**
4	gpt-4o + doubao-pro-1.5-32k + Deepseek-V3 + Qwen2.5-7B-Instruct	0.6621
5	gpt-4o + doubao-pro-1.5-32k + Deepseek-V3 + Qwen2.5-7B-Instruct + Phi-4-mini-instruct	0.6448

Table 6: Ablation on number of models.

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. 8, 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.

The new overall leader, GPT-OSS-120B, exhibits a strong aptitude for technical tasks, securing top ranks in coding, reasoning, and mathematics. However, its performance is less dominant in creative domains like roleplay. A clear trade-off is visible in other models; for example, Deepseek-R1-

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 9: 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 10: Prompt for naive criteria decomposition.

Order of Models	ACC
gpt-4o → doubao-pro-1.5-32k → Deepseek-V3	0.6668
gpt-4o → Deepseek-V3 → doubao-pro-1.5-32k	**0.6962**
Deepseek-V3 → gpt-4o → doubao-pro-1.5-32k	0.6699
Deepseek-V3 → doubao-pro-1.5-32k → gpt-4o	0.6920
doubao-pro-1.5-32k → gpt-4o → Deepseek-V3	0.6778
doubao-pro-1.5-32k → Deepseek-V3 → gpt-4o	0.6605

Table 7: Ablation on order of models.

250120 excels more in writing and roleplay but is weaker in mathematics. Conversely, Gemma-3-27B-IT achieves the top score in knowledge and ranks highly in reasoning, yet its performance is notably lower in STEM-oriented subjects like coding and mathematics. Despite a lower overall ranking, claude-3-7-sonnet-20250219 shows competitive coding abilities, but its weaker reasoning skills hinder its general performance. Additionally, it is worth mentioning that doubao-1-5-pro-32k-250115 possesses a notable advantage in mathematical abilities relative to its overall rank.

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. 11 shows an example of a manually created taxonomy for the coding domain. Additional examples can be found in our open-source code repository.

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. 12.

C.1.3 Algorithm for TaxBuilder

The complete TaxBuilder algorithm is presented in Alg. 1.

Algorithm 1 TaxBuilder Algorithm

Input: Initial tree T_{init}, new nodes TG_{init}, LLM-as-Decision-Maker of node insertion $\texttt{DM}_{ins}$
Output: Final taxonomy tree T_{final}

function INSERTNODE(T, tg)
  $d \leftarrow \texttt{DM}_{ins}(tg, T^c)$
  if $d =$ "<E>" **then**
    return T
  else if $d =$ "<S>" **then**
    $T^c \leftarrow T^c \cup \{tg\}$
    return T
  else ▷ Child level: LLM returns parent node name
    $T^{c_i} \leftarrow d$
    return INSERTNODE(T^{c_i}, tg)
  end if
end function

function TAXBUILDER(T_{init}, TG_{init})
  $T_{cur} \leftarrow T_{init}$
  for all $tg \in TG_{init}$ **do**
    $T_{cur} \leftarrow$ INSERTNODE(T_{cur}, tg)
  end for
  $T_{cur} \leftarrow$ REFINEPRUNENODE(T_{cur}) ▷ Node refinement and pruning
  $T_{final} \leftarrow$ PRUNELAYER(T_{cur}) ▷ Layer pruing
  return T_{final}
end function

Model	Overall	Writing	Roleplay	Knowledge	Coding	Mathematics	Reasoning
Closed-source and Open-source (>100B) LLMs							
GPT-OSS-120B	242.24(1)	233.59(4)	231.33(7)	247.99(3)	263.35(1)	241.59(2)	235.70(1)
Deepseek-V3-250324	237.65(3)	233.04(5)	237.41(5)	250.91(2)	247.09(3)	220.01(5)	229.93(4)
Deepseek-R1-250120	235.25(4)	244.14(1)	248.29(2)	238.19(4)	231.94(6)	190.58(8)	234.91(2)
claude-3-7-sonnet-20250219 🔒	225.07(7)	232.74(6)	224.94(10)	229.79(8)	238.37(5)	187.36(10)	203.56(11)
gpt-4o-2024-11-20 🔒	214.80(9)	225.90(8)	234.42(6)	219.31(10)	202.03(10)	171.82(11)	210.50(9)
qwen-max-2024-10-15 🔒	182.59(13)	187.44(13)	174.14(14)	192.56(13)	183.33(13)	164.28(13)	178.68(14)
doubao-1-5-pro-32k-250115 🔒	182.23(14)	175.36(18)	173.19(15)	192.40(14)	187.05(12)	192.35(6)	181.46(13)
hunyuan-standard-2025-02-10 🔒	156.44(19)	163.21(20)	144.43(20)	156.85(18)	158.49(18)	149.47(15)	152.47(19)
Open-source (<100B) LLMs							
Qwen3-32B	241.13(2)	241.29(2)	246.03(3)	234.47(6)	246.87(4)	245.29(1)	224.93(6)
Gemma-3-27B-IT	229.88(5)	236.63(3)	243.59(4)	253.25(1)	207.01(9)	192.19(7)	233.82(3)
QwQ-32B	229.38(6)	232.26(7)	250.41(1)	237.71(5)	224.73(7)	187.67(9)	228.24(5)
Qwen3-8B	222.88(8)	224.63(9)	227.69(9)	210.07(11)	219.59(8)	241.40(3)	216.47(7)
GPT-OSS-20B	213.72(10)	196.23(12)	176.09(13)	230.79(7)	251.59(2)	225.26(4)	212.92(8)
Gemma-3-4B-IT	203.43(11)	224.30(10)	230.57(8)	219.62(9)	167.27(15)	137.90(17)	207.15(10)
Qwen2.5-72B-Instruct	190.42(12)	197.99(11)	179.91(11)	194.18(12)	190.53(11)	171.61(12)	194.16(12)
Qwen2.5-32B-Instruct	176.10(15)	185.19(14)	170.58(16)	174.20(15)	174.39(14)	162.31(14)	173.53(15)
Meta-Llama-3.1-70B-Instruct	170.51(16)	183.13(15)	178.49(12)	167.73(16)	165.25(16)	133.37(18)	170.07(16)
Qwen2.5-7B-Instruct	166.67(17)	179.10(16)	164.64(17)	162.77(17)	161.99(17)	146.67(16)	163.15(17)
Meta-Llama-3.1-8B-Instruct	159.01(18)	176.65(17)	159.26(18)	152.30(20)	154.93(19)	119.67(20)	159.35(18)
Mistral-7B-Instruct-v0.3	148.94(20)	163.44(19)	158.07(19)	152.55(19)	138.24(21)	114.74(21)	133.61(21)
Phi-4-mini-instruct	143.99(21)	150.48(21)	144.18(21)	142.36(21)	143.21(20)	123.93(19)	148.08(20)

Table 8: Overall and domain-level scores (fine-grained results on project page).

C.1.4 Prompts for TaxBuilder

The prompt for node insertion in the LLM-as-Decision-Maker framework is shown in Fig. 13. The prompt for node refinement and pruning in the LLM-as-Decision-Maker framework is shown in Fig. 14.

C.2 RealMix

C.2.1 Seed Data

The prompt for labeling the domain is shown in Fig.15.

The prompts for query tagging are shown in Fig.16, Fig.17, Fig.18, Fig.19, Fig.20, and Fig.21.

The prompts for quality labeling are shown in Fig.22, Fig.23, Fig.24, Fig.25, Fig.26, and Fig. 27.

The seed data is available in our dataset repository.

C.2.2 Synthesize New Queries

The prompt of RealMix is shown in Fig. 28

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 including Automatic Weakness Reporter, Automatic Weakness Analyzer, Failure Mode Explorer, and Automatic Difference Reporter.

(1) Automatic Weakness Reporter detects capability imbalances by comparing a model's rank-

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 11: Initial taxonomy for coding domain.

ing at a specific evaluation node with its overall ranking. If the local ranking differs from the global ranking by more than a predefined threshold (set to 5 in our experiments), the node is flagged as a weakness (negative deviation) or as a strength (positive deviation). This enables rapid localization of domains where the model exhibits pronounced strengths or weaknesses. For example, *Gemma-3-4B-IT* shows its largest deviation in *root.mathematics.Task Types.Visualization.Geometric Drawing*, where its overall ranking is 9 but its ranking in *Geometric Drawing* drops to 16, revealing a substantial weakness in spatial and geometric visualization tasks.

(2) Automatic Weakness Analyzer builds on the results of the Automatic Weakness Reporter and uses an advanced LLM (QwQ-32B in our study) to transform numerical deviations into human-readable summaries, grouping related deficiencies and strengths. This allows evaluators to understand the qualitative nature of the weaknesses detected. For *Gemma-3-4B-IT*, for instance, the analyzer output includes: "Coding & Tool Usage: Struggles with Swift, Batch, Lua programming languages; IDE configuration and test development." Such summaries enable model developers to target specific technical capabilities for improvement.

(3) Failure Mode Explorer investigates whether

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): - "Computation": Performing calculations. - "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 12: Prompt for initial annotation of tags.

underperformance in a hierarchical category is comprehensive or localized. For any underperforming parent node, it calculates the standard deviation (STD) of rankings across its child nodes. If the STD is within the top 20% of all nodes, the performance is considered *unstable*, meaning that low performance may be caused by a few sub-capabilities. If the STD is in the bottom 20%, the weakness is *comprehensive* and spread across all sub-capabilities. For example, *gemma* ranks 7 places worse in *root.mathematics.Mathematical Subfields.Algebra* than its overall ranking, with a low STD of 0.4899. This indicates broad underperformance across Algebra, suggesting the need for holistic improvement rather than fixes in isolated subfields such as *Linear Algebra*.

(4) Automatic Difference Reporter enables model-to-model comparison by measuring differences in performance variability across evaluation nodes. The larger the difference in standard deviation between two models, the more divergent their capabilities are in that domain. This is useful for identifying areas of specialization or distinct model behavior. For instance, a comparison between *claude-3.7-sonnet* and *gpt-4o* reveals substantial differences in *root.roleplay.Style Types.Experimental Style.Glitch Art*, *root.roleplay.Theme Types.Emotional.Erotic*,

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

and *root.reasoning.Application Domains.Personal Development Guidance*, pointing to contrasting strengths and specialization patterns between the two models.

Overall, this integrated framework transforms large-scale ranking data into actionable insights. By combining deviation detection, natural language summarization, hierarchical stability analysis, and comparative difference measurement, it supports interpretable evaluation and targeted optimization for LLM development.

C.4 SCAN-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. 29, present participants with 20 question pairs (one generated and one original) in randomized order to prevent positional bias.

Tab. 9 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.

C.4.2 Query Quality: Data Contamination

Following Auto-Arena (Zhao et al., 2024), we employ two methods to detect test data contamination. The first is the string match metric, as used in GPT-4 (Achiam et al., 2023), 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 (Lee et al., 2023), uses sentence embed-

Prompt of LLM-as-Decision-Maker (Node Refinement and Pruning)

Your task is to optimize a given tree structure classification system. I will provide all the next-level nodes under a certain category (referred to as "current layer nodes") and the leaf nodes among them (referred to as "leaf nodes in the current layer"). You need to determine the next steps and return results in the specified format. Before giving the final answer, please provide a brief analysis. Note: Do not modify the names of the original categories, and do not omit any categories from the "current layer nodes" when returning results. Each task analysis should analyze all categories without omission.

If you see that the "current layer nodes" are all classification criteria rather than specific categories, this layer absolutely must not be modified.

Task 1: Split
Examine all nodes in the "leaf nodes in the current layer" one by one, determine if there are any nodes that are not "single concepts" and split them. If the newly split category already exists, do not return that new category. A "single concept" refers to an independent concept, as opposed to a "combined concept" which is formed by combining multiple concepts. For example, "mythology novel" is a combined concept, consisting of "mythology" and "novel". Return format:
[(old_key1, [new_key1, new_key2]), (old_key2, [new_key1, new_key2])]

Task 2: Parent-Child Relationship
Analyze all nodes in the "current layer nodes" one by one, determine if there exist parent-child relationships, or if multiple nodes can be merged into a parent node, and return the results. If there are no qualifying situations, do not return anything. Return format:
[(parent_key1, [old_child_key1, old_child_key2]), (parent_key2, [old_child_key1, old_child_key2])]

Task 3: Delete
Examine all nodes in the "current layer nodes" one by one, identify meaningless categories (such as null values, blanks, etc.) or extremely rare categories in the real world, and delete them. Return format:
[old_key_to_delete1, old_key_to_delete2]

Task 4: Merge
Analyze all nodes in the "current layer nodes" one by one, identify categories with consistent concepts and merge them. Return format:
[(new_key1, [old_key_to_merge1, old_key_to_merge2]), (new_key2, [old_key_to_merge1, old_key_to_merge2])]

Task 5: Keep
Preserve the remaining nodes. Return format:
[old_key1, old_key2]

Output Format
Provide a brief analysis for each task and give the results after the task is executed.
After completing all tasks, format the final result as a Python dictionary wrapped in <decision></decision> tags, in the following format, where all keys with old_ must have appeared in the "current layer nodes":
<decision> {
"split": [(old_key1, [new_key1, new_key2]), (old_key2, [new_key1, new_key2])],
"reparent": [(parent_key1, [old_child_key1, old_child_key2]), (parent_key2, [old_child_key1, old_child_key2])],
"delete": [old_key_to_delete1, old_key_to_delete2],
"merge": [(new_key1, [old_key_to_merge1, old_key_to_merge2]), (new_key2, [old_key_to_merge1, old_key_to_merge2])],
"keep": [old_key1, old_key2]
} </decision>
If there is no operation for a certain task, do not display that item in the final output.

Current Layer Nodes
{current_keys}

Leaf Nodes in the Current Layer
{leaf_keys}

Figure 14: Prompt of LLM-as-Decision-Maker (node refinement and pruning).

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

Table 9: Query quality and reality evaluation results by human rater.

ding similarity: a question is considered contaminated if its BERT-Large (Devlin et al., 2019) 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 SCAN-D-V0

We present several samples of SCAN-D-V0 in Tab. 10.

Prompt of Domain Annotation

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 15: Prompt of domain annotation.

Prompt of Query Tags Annotation (writing)

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 16: Prompt of query tags annotation (writing).

C.4.4 SCAN-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 (Zheng et al., 2023) 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

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 17: 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 18: Prompt of query tags annotation (knowledge).

illustrated in Fig. 30.

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 SCAN-HPD is shown in Fig. 31. Similar to SCAN-D-V0, these queries span six domains: writing, roleplay, knowledge, coding, mathematics, and reasoning.

D Evaluation Methods

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 19: 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 20: 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 21: 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 22: 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 23: 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 24: 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 25: 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 26: Prompt of quality annotation (mathematics).

Prompt of Quality Annotation (reasoning)

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?
4. **Real-World Application**: Is the question something that would be encountered in real-world?
5. **Problem-Solving**: Does it require devising a solution or strategy?
6. **Deep Thinking**: Does it require deep reasoning and consideration of multiple factors?
7. **Novelty:** Does the question present a unique or unusual scenario that the LLM is unlikely to have encountered before?

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 (reasoning).

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 28: Prompt of RealMix.

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 29: Questionnaire protocol for comparative evaluation.

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., CO2 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("ï", "")‘ 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?

Table 10: Examples from different domains in SCAN-D-V0.

Your choice: Not selected yet

⚠️ Instructions

- Your task is to compare two answers and select the better one. Please click the "A is better", "B is better", or "Tie" button to make your choice.
- You can use keyboard shortcuts: 1 (Choose A is better), 2 (Choose B is better), 3 (Tie), ← (Previous page), → (Next page)
- You can exit at any time, but please make sure to click the "Save Results" button before exiting, otherwise your progress will be lost!
- Want to continue previous work? Please upload your previously saved file and click "Jump to first unlabeled data" to continue.
- **Important reminder:** Before closing the browser, you must click "Save Results", otherwise closing the page will cause all unsaved progress to be lost!

Choose File *human_preference_dataset_v0.1.jsonl*

Prompt (47/800):

← Previous

ehance my email: Alan, I hope all is well. I am circulating back to see if you progressed your deal.

PACE Equity and Builders Capital stand ready.

I look forward to hearing from you.

Kind regards

Next →

A is better [Key 1] | B is better [Key 2] | Tie [Key 3] | Jump to first unlabeled data | Save Results

Response A:

Here's an enhanced version of your email:

Subject: Follow-Up on Project Financing Options

Dear Alan,

I hope this message finds you well.

I am reaching out to follow up on our previous discussions regarding the financing of your project. Have there been any developments or decisions made concerning the deal?

As a reminder, **PACE Equity** and **Builders Capital** are fully prepared to assist and move forward with the necessary arrangements to support your project's financial needs.

Please let me know if there are any updates or if there is further information you require from our side to facilitate this process.

I look forward to your response.

Kind regards,

[Your Name] [Your Position] [Your Contact Information]

This version provides a more formal structure, includes a subject line for clarity, and ensures that the purpose of the email is straightforward while maintaining a polite and professional tone.

Response B:

Certainly! Here's an enhanced version of your email:

Subject: Follow-Up on Your Deal Progress

Hi Alan,

I hope this message finds you well. I wanted to circle back and check in to see if there have been any updates or progress on your deal.

Please know that both PACE Equity and Builders Capital are fully prepared to support you in any way needed to help move things forward.

Looking forward to hearing from you soon.

Kind regards, [Your Name]

This version is slightly more polished and professional, while maintaining the original intent. Let me know if you'd like any further adjustments!

Figure 30: The annotation interface used for labeling the human preference dataset.

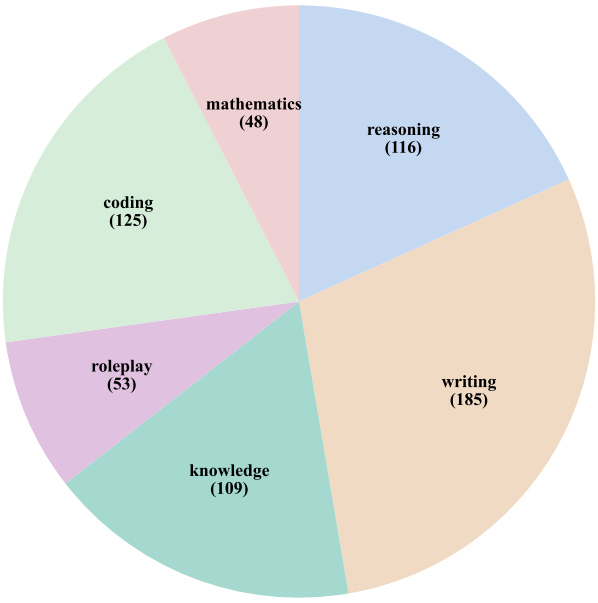

Figure 31: Domain distribution of SCAN-HPD.

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 32: 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 33: Example (2) of Pairwise Evaluation.