CritiQ: Mining Data Quality Criteria from Human Preferences

Anonymous ACL submission

Abstract

Language model heavily depends on high quality data for optimal performance. Existing approaches rely on manually designed heuristics, perplexity of existing models, training classifiers, or careful prompt engineering, which require significant expert experience and human annotation effort while introduce biases. We introduce CritiQ, a novel data selection method that automatically mines criteria from human preferences for data quality with only ∼30 human-annotated pairs and perform efficient data selection. The main component, CritiQ Flow, employs a manager agent to evolve quality criteria and worker agents to make pairwise judgments. We build a knowledge base that extracts quality criteria from previous work to boost CritiQ Flow. Compared to perplexity- and classifier- based methods, verbal criteria are more interpretable and possess reusable value. After deriving the criteria, we train the CritiQ Scorer to give quality scores and perform efficient data selection. We demonstrate the effectiveness of our method in the code, math, and logic domains, achieving high accuracy on human-annotated test sets. To validate the quality of the selected data, we continually train `Llama 3.1` models and observe improved performance on downstream tasks compared to uniform sampling. Ablation studies validate the benefits of the knowledge base and the reflection process. We analyze how criteria evolve and the effectiveness of majority voting.

1 Introduction

Large language models (LLMs) show significant performance in various downstream tasks (Brown et al., 2020; OpenAI et al., 2024; Dubey et al., 2024). Studies have found that training on high quality corpus improves the ability of LLMs to solve different problems such as writing code, doing math exercises, and answering logic questions (Cai et al., 2024; DeepSeek-AI et al., 2024; Qwen et al., 2024). Therefore, effectively selecting high-quality text data is an important subject for training LLM.

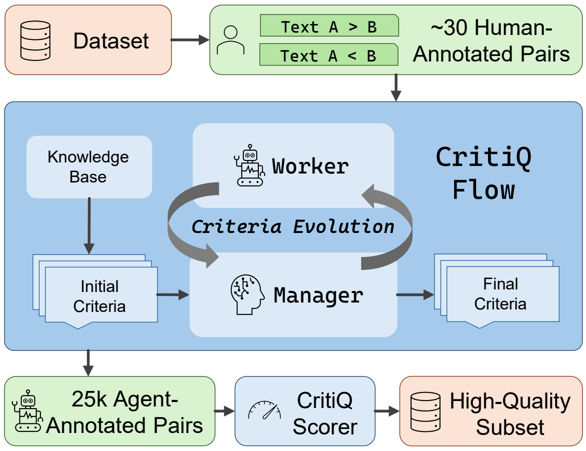


Figure 1: The overview of CritiQ. We (1) employ human annotators to annotate ∼30 pairwise quality comparisons, (2) use CritiQ Flow to mine quality criteria, (3) use the derived criteria to annotate 25k pairs, and (4) train the CritiQ Scorer to perform efficient data selection.

To select high-quality data from a large corpus, researchers manually design heuristics (Dubey et al., 2024; Rae et al., 2022), calculate perplexity using existing LLMs (Marion et al., 2023; Wenzek et al., 2019), train classifiers (Brown et al., 2020; Dubey et al., 2024; Xie et al., 2023) and query LLMs for text quality through careful prompt engineering (Gunasekar et al., 2023; Wettig et al., 2024; Sachdeva et al., 2024). Large-scale human annotation and prompt engineering require a lot of human effort. Giving a comprehensive description of what high-quality data is like is also challenging. As a result, manually designing heuristics lacks robustness and introduces biases to the data processing pipeline, potentially harming model performance and generalization. In addition, quality standards vary across different domains. These methods can not be directly applied to other domains without significant modifications.

To address these problems, we introduce CritiQ [1], a novel method to automatically and effectively capture human preferences for data quality and perform efficient data selection. Figure 1 gives an overview of CritiQ, comprising an agent workflow, CritiQ Flow, and a scoring model, CritiQ Scorer. Instead of manually describing how high quality is defined, we employ LLM-based agents to summarize quality criteria from only ∼30 human-annotated pairs.

CritiQ Flow starts from a knowledge base of data quality criteria. The worker agents are responsible to perform pairwise judgment under a given criterion. The manager agent generates new criteria and refines them through reflection on worker agents' performance. The final judgment is made by majority voting among all worker agents, which gives a multi-perspective view of data quality.

To perform efficient data selection, we employ the worker agents to annotate a randomly selected pairwise subset, which is 1000x larger than the human-annotated one. Following Korbak et al. (2023); Wettig et al. (2024), we train CritiQ Scorer, a lightweight Bradley-Terry model (Bradley and Terry, 1952) to convert pairwise preferences into numerical scores for each text. We use CritiQ Scorer to score the entire corpus and sample the high-quality subset.

For our experiments, we established human-annotated test sets to quantitatively evaluate the agreement rate with human annotators on data quality preferences. We implemented the manager agent by `GPT-4o` and the worker agent by `Qwen2.5-72B-Insruct`. We conducted experiments on different domains including code, math, and logic, in which CritiQ Flow shows a consistent improvement in the accuracies on the test sets, demonstrating the effectiveness of our method in capturing human preferences for data quality. To validate the quality of the selected dataset, we continually train `Llama 3.1` (Dubey et al., 2024) models and find that the models achieve better performance on downstream tasks compared to models trained on the uniformly sampled subsets.

We highlight our contributions as follows. We will release the code to facilitate future research.

- We introduce CritiQ, a method that captures human preferences for data quality and performs efficient data selection at little cost of human annotation effort.
- Continual pretraining experiments show improved model performance in code, math, and logic tasks trained on our selected high-quality subset compared to the raw dataset.
- Ablation studies demonstrate the effectiveness of the knowledge base and the the reflection process.

[1] **Criti**a of Data **Q**uality, pronounced as "critic".

2 Related Work

Heuristics for Data Selection. Using manually designed heuristics to identify data with specific characteristics is a basic approach for data selection. Common rules include keyword or stopword matching, length-based filtering, data source filtering, in-document duplication (Dubey et al., 2024; Cai et al., 2024), and training classifiers (noa, 2024; Xie et al., 2023; Dubey et al., 2024; Wei et al., 2024; Korbak et al., 2023; Lv et al., 2024). Designing these rules requires much experience and human effort.

Researchers also design specific rules to select high-quality domain data (Wang et al., 2023; Lozhkov et al., 2024; Huang et al., 2024), which requires much expert experience and lacks scalability and generalization.

Quality Signals from LLMs. The use of LLMs to assess data quality has become a prevalent approach. Researchers employ manual-designed prompts to query LLMs for quality assessment (Dubey et al., 2024; Sachdeva et al., 2024; Zhang et al., 2024), often using educational value as a proxy for data quality (Gunasekar et al., 2023; Wei et al., 2024). However, their focus remains limited to fixed aspects of data quality. Although these methods reduce the need for human annotation, they introduce inherent biases through predefined rules and standards.

Previous works like QuRating (Wettig et al., 2024) evaluate data quality using multiple manually defined criteria including writing style, factual accuracy, level of expertise, and educational value. These predefined criteria show varying effectiveness across different domains, suggesting that manually summarized criteria lack generalization and can not accurately describe data quality. In contrast, CritiQ Flow automatically discovers quality criteria by effectively capturing human preferences about data quality assessment from a few number of human-annotated pairs.

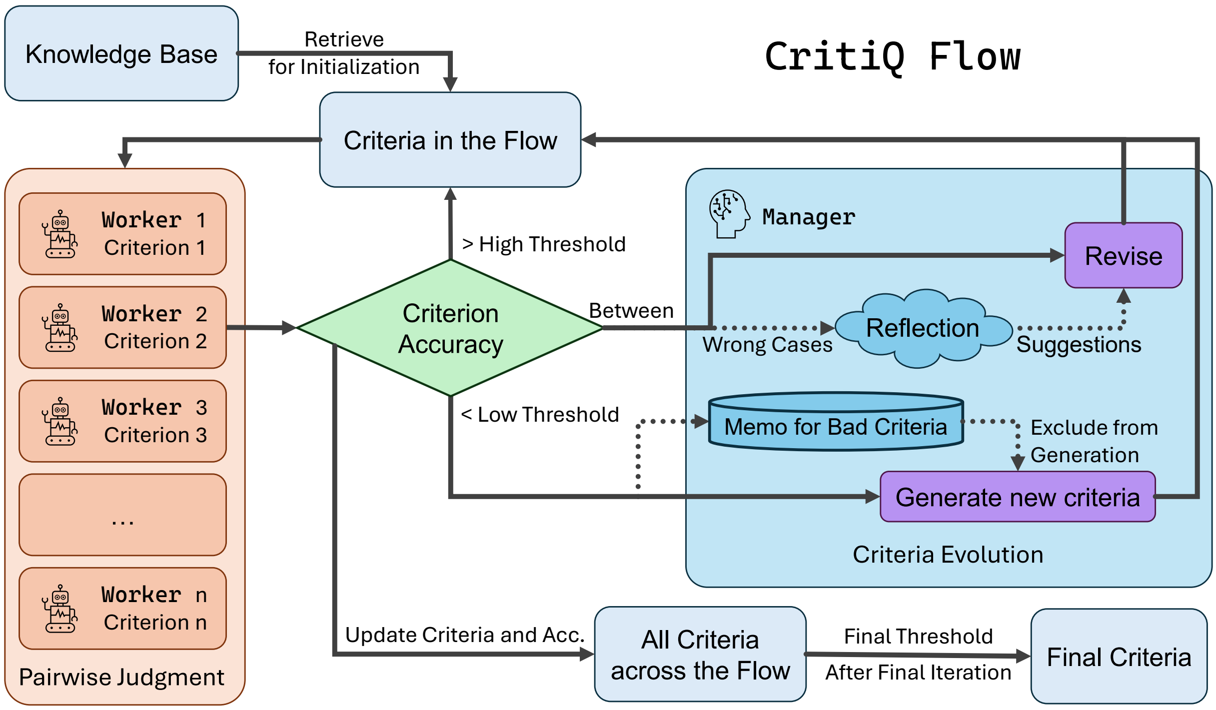


Figure 2: CritiQ Flow comprises two major components: multi-criteria pairwise judgment and the criteria evolution process. The multi-criteria pairwise judgment process employs a series of worker agents to make quality comparisons under a certain criterion. The criteria evolution process aims to obtain data quality criteria that highly align with human judgment through an iterative evolution. The initial criteria are retrieved from the knowledge base. After evolution, we select the final criteria to annotate the dataset for training CritiQ Scorer.

Thought and Reflection of LLMs. Prompting LLMs to reason before giving the final answer improves the model's performance on various tasks (Kojima et al., 2023; Yao et al., 2023). In our work, we also require the agents to think and analyze before making the quality comparison.

Reflection is a common technique to improve the performance of LLMs through iterative critiquing and refinement (Shinn et al., 2023; Madaan et al., 2023; Saunders et al., 2022; Xi et al., 2024). Existing frameworks have integrated the reflection mechanism to build LLM-based agents and do prompt engineering (Yuksekgonul et al., 2024; Asai et al., 2023; Wu et al., 2023). In CritiQ Flow, we alse prompt the agent to examine the wrong predictions and refine the quality criteria accordingly.

3 Method

3.1 Overview

In CritiQ, we first use an agent workflow, CritiQ Flow, to automatically extract quality criteria from human preferences for data quality with limited human annotation, and then use these criteria to train a scoring model, CritiQ Scorer, to efficiently perform large-scale data selection.

For a specific text dataset D, we sample $\sim$30 pairs of data points. Compared to works that the authors carefully design prompts (Dubey et al., 2024; Sachdeva et al., 2024; Zhang et al., 2024; Gunasekar et al., 2023; Wei et al., 2024), a small amount of data annotation requires less human effort. We employ human expert annotators to determine which data point in each pair is of higher quality, forming the training set D_{human} for CritiQ Flow. Details for annotation are shown in Appendix B. Figure 2 shows how CritiQ Flow mines quality criteria from D_{human}. Prompts we used are shown in Appendix E.

To perform large-scale data selection, we train CritiQ Scorer, a lightweight scoring model. Following Korbak et al. (2023); Wettig et al. (2024), we use a Bradley-Terry model (Bradley and Terry, 1952) to convert the pairwise comparison into a numerical score. We randomly sample a larger number of text pairs, forming the training dataset D_{agent} for CritiQ Scorer. The quality preference labels will be annotated by the worker agents through the pairwise judgment process under the obtained quality criteria. Finally, we use CritiQ Scorer to score all text data in D and select the high-quality subset according to the quality scores.

3.2 Knowledge Base

As an iterative agent workflow, the quality of the initial criteria is crucial for CritiQ Flow. Many research papers have shared valuable insights on quality standards and have succeeded in data selection. Therefore, we can leverage findings from these data selection studies to establish a criteria knowledge base. Drawing from well-validated methodologies, the knowledge base can enhance the initialization of CritiQ Flow, ensuring a strong foundation for subsequent refinements.

To construct the knowledge base, we first crawl the cited papers of the datasets published on the Hugging Face Hub [2]. We only use the arXiv papers available in HTML format, avoiding potential issues with PDF parsing. We employ `GPT-4o-mini` to identify papers that introduce datasets from the titles and abstracts. Subsequently, we use `GPT-4o-mini` to systematically extract quality criteria from these papers. After de-duplication, we establish a knowledge base $C_{\text{knowledge}}$ comprising 342 distinct quality criteria.

Algorithm 1 Retrieve Criteria from C_{domain}

Input: $C_{\text{domain}}, D_{\text{human}}, n$
Output: $C_{\text{retrieved}}$

```
Initialize C_retrieved[ ], Acc[ ]
for c_i ∈ C_domain do
    Acc_i ←acc over D_human
end for
Sort C_domain by Acc                ▷ Descending order
for c_i ∈ C_domain do
    if LENGTH(C_retrieved ≥ n) then
        break
    end if
    if Acc_i > 0.5 then
        APPEND(C_retrieved, c_i)
    end if
end for
return C_retrieved
```

We use this $C_{\text{knowledge}}$ to provide initial criteria for CritiQ Flow. We query a model with the domain description of the dataset to retrieve potentially useful criteria from $C_{\text{knowledge}}$, forming C_{domain}. As shown in Algorithm 1, we then retrieve n criteria from C_{domain}. If the criteria are not enough, we query the manager agent to propose new criteria.

[2]https://huggingface.co/datasets data collected before July 2024

3.3 Multi-Criteria Pairwise Judgment

Given a set of quality criteria C and a pair of data points $p = (\text{text}_\text{A}, \text{text}_\text{B}) \in D_h$, the pairwise judgment process gives a quality preference by a worker agent. Each criterion has a corresponding description to guide the comparison. In consideration of cost and efficiency, we do not use an expensive model as the worker agent. Instead, we use a model that can perform simple comparisons under a single criterion, which is not difficult for many open-source LLMs.

For each criterion $c_i \in C$, we query a distinct worker agent to determine which data point exhibits higher quality. The worker agent analyzes both data points with respect to c_i before making a judgment. If c_i is not applicable or if both text A and B of p demonstrate comparable quality, the worker agent can refuse to provide an answer, i.e., answer "null". The final judgment across all criteria is made through majority voting, i.e.,

$$\text{judge}(p, C) = \text{majority}_{c_i \in C}(\{\text{worker}_i(p, c_i)\}),$$

where $\text{worker}_i(p, c_i) \in \{\text{A}, \text{B}, \text{null}\}$ is the worker agent's judgment of p under c_i.

Because we only focus on whether the final judgment is consistent with the human annotation and do not require all criteria to be applicable to a certain pair, we do not take these situations into consideration when calculating the accuracy for this criterion. The criterion accuracy for c_i on dataset D_h is calculated as

$$\text{acc}(c_i|D_h) = \frac{|\{p \in D_h|\text{w}_i(p, c_i) = \text{h}(p)\}|}{|D_h| - |\{p \in D_h|\text{w}_i(p, c_i) = \text{null}\}|},$$

where $h(p) \in \{A, B\}$ is the human-annotated higher-quality one in p, and $\text{w}_i(p, c_i)$ is the worker agent's judgment of p according to c_i.

3.4 Criteria Evolution

After retrieving the initial criteria from the knowledge base, we perform an iterative criteria evolution to improve the accuracy on D_{human}. For each iteration, we first make pairwise judgments on D_{human}. Based on the accuracy acc_i of each criterion c_i, we then divide them into three groups by a high threshold t_{high} and a low threshold t_{low}. For c_i with $acc_i \geq t_{\text{high}}$, we keep them directly. For c_i with $acc_i \leq t_{\text{low}}$, we remove them and query the manager agent to generate new criteria. Simultaneously, they will be recorded to avoid being

	Method	Code	Δ	Math	Δ	Logic	Δ	Avg.	Δ
	Vanilla	82.02	-	72.86	-	72.99	-	75.96	-
	TextGrad	72.70	-9.32	78.57	+5.71	75.22	+2.23	75.50	-0.46
QuRating	Writing Style	73.03	-8.99	52.86	-20.00	59.70	-13.29	61.86	-14.09
QuRating	Facts & Trivia	76.40	-5.62	44.29	-28.57	84.33	+11.34	68.34	-7.62
QuRating	Educational Value	85.39	+3.37	68.57	-4.29	84.33	+11.34	79.43	+3.47
QuRating	Require Expertise	79.21	-2.81	52.86	-20.00	84.33	+11.34	72.13	-3.82
	CritiQ Flow	**89.33**	**+7.31**	**84.57**	**+11.71**	**88.06**	**+15.07**	**87.32**	**+11.36**
	w/o evo.	86.40	+4.38	78.00	+5.14	85.97	+12.98	83.46	+7.50
	w/o k.b.	87.19	+5.17	82.57	+9.71	81.64	+8.65	83.80	+7.84
	w/o evo. & k.b.	83.03	+1.01	76.29	+3.43	68.36	-4.63	75.89	-0.06
	CritiQ Scorer	**89.89**	**+7.87**	**90.00**	**+17.14**	**90.22**	**+17.23**	**90.04**	**+14.08**

Table 1: Accuracies on the human-annotated D_{test}. The best results and the best results without training a model are in bold. "Δ" is the delta value with the vanilla results. "evo." for iterative criteria evolution. "k.b." for retrieving initial critieria from the knowledge base instead of generating all initial critieria by the manager agent. The results are the average over 5 experiments with different random seeds.

generated again by the manager agent in subsequent iterations. For c_i with $t_{\text{low}} < acc_i < t_{\text{high}}$, we ask the manager agent to do reflection. For each incorrect judgment of $p \in \{p|worker(p, c_i) \notin \{h(p), null\}\}$, we provide the manager with the right answer $h(p)$ and worker agent's thought. The manager agent should analyze why the worker agent makes mistakes and provide a suggestion to itself on how to improve the criteria. Given all suggestions from the wrong cases, the manager agent should refine the description of c_i as c_i'. acc_i' will be calculated in the next iteration.

Unlike the gradient descent algorithm, text-based optimization does not guarantee that the loss will decrease within a neighborhood of the current state. Therefore, we need to introduce external constraints to ensure this. In CritiQ Flow, we save all criteria c_i throughout the evolution process with their accuracies acc_i. After getting the new accuracy acc_i' of a revised criterion c_i', we will only update the description of it when $acc_i' \geq acc_i$. This constraint ensures that the description revision will not make the criterion worse. The final criteria are those with the highest accuracy of all criteria across iterations.

3.5 Train the Scoring Model

After obtaining the quality criteria, we can use them to annotate a larger number of pairs from the dataset D to train CritiQ Scorer. To form the pairs, we randomly sample several data points and group them by the length of the text to remove the potential influences of length biases of the worker agent. We then use the pairwise judgment process to annotate the pairs according to the quality criteria mined by CritiQ Flow, forming D_{agent}. Only worker agents are employed in this process, which get rid of the high cost API calls to the manager agent.

Training the CritiQ Scorer s_θ is straightforward by minimizing the loss function,

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{p \in D_{\text{agent}}} \log \sigma(s_\theta(d_{\text{high}}) - s_\theta(d_{\text{low}}))$$

where σ is the sigmoid function, d_{high} and d_{low} are the relatively high and low quality data points in the pair p.

3.6 Selecting Data

In consideration of cost and efficiency, we use a lightweight base model as the scoring model, which increases the speed of scoring the entire dataset D. After getting a score $s_\theta(d_i)$ for each data point d_i in D, we normalize the scores to obtain the final quality score s_i. As QuRating (Wettig et al., 2024) suggests, sampling is better than naive top-k selection. We select each data point d_i with the probability $p_i \propto exp(\frac{s_i}{\tau})$, where τ is the temperature. This process is implicitly equivalent to reward-weighted regression (Wettig et al., 2024; Korbak et al., 2023; Peters and Schaal, 2007). We use the Gumble top-k trick (Wettig et al., 2024; Kool et al., 2019) to perform efficient sampling without replacement.

4 Experiments

We verify the effectiveness of CritiQ Flow in improving the accuracies on human-annotated test sets. Hyperparameters for CritiQ Flow are shown in Appendix A. We continually pretrain a `Llama-3.1-3B` model to show the improved quality of our selected subset compared to the original dataset.

4.1 Setup

Domain	$\#D_{\text{human}}$	$\#D_{\text{agent}}$	$\#D_{\text{test}}$
Code	25	25000	193
Math	30	25000	70
Logic	30	25000	134

Table 2: Number of pairs in each split.

Dataset. We focus on three domains: code, math and logic. We use the Python subset of the Stack v2 (Lozhkov et al., 2024), the non-code subset of OpenWebMath (Paster et al., 2023) and Zyda-2 (Tokpanov et al., 2024) datasets as the source dataset D. The numbers of pairs of D_{human} and D_{agent} are shown in Table 2.

Models. We employ `GPT-4o` [3] as the manager agent which is good at reflection but is costly, and `Qwen2.5-72B-Instruct` as the worker agent which can perform simple pairwise comparison while is relatively cheap. We initialize CritiQ Scorer by `Qwen2.5-1.5B` for efficiency considerations. Hyperparameters for CritiQ Scorer are shown in Appendix A.

Baselines. Directly prompting the worker LLM for data quality comparison serves a vanilla baseline. We use the same prompt as ours without specifying a criterion for vanilla baseline experiments. We compare the optimization algorithm in our workflow with TextGrad (Yuksekgonul et al., 2024). The initial prompt for TextGrad is the same as the vanilla baseline. We run TextGrad optimizations on the same training set D_{agent} as ours. We compare our criteria with those proposed by QuRating (Wettig et al., 2024). The prompts for QuRating are from their original work.

Evaluation. We evaluate CritiQ Flow by the accuracy on the human-annotated test set D_{test}. High accuracy indicates effectiveness in capturing human preferences for data quality. For each pair, three annotators will determine which data point exhibits higher quality independently under the same annotation guidelines with D_{human}. We only keep the pairs for which all three annotators give the same judgment. The final number of pairs in D_{test} is shown in Table 2. We emphasize that although we take human effort to annotate more pairs for validation purpose, and the workflow itself just need a tiny annotated dataset to work. We will show how well CritiQ Flow mines data quality criteria by only $\sim$30 human annotated pairs and get high accuracies on D_{test}.

4.2 Results

We report the accuracies of the baselines and CritiQ on the test set of all 3 domains in Table 1. In addition, we report the ablation results for the knowledge base and the criteria evolution process.

Vanilla method can be improved by TextGrad and CritiQ Flow. Although the vanilla method is not low in the agreement rate with human annotators, it can be further improved by TextGrad (Yuksekgonul et al., 2024) and CritiQ Flow. Detailed descriptions and instructions help the worker agent to perform better judgments.

CritiQ Flow outperforms TextGrad. Compared with TextGrad, CritiQ Flow achieves higher accuracies in all domains, indicating a higher effectiveness in capturing human preferences for data quality. Interestingly, we find that TextGrad is also trying to find quality criteria, but it is not as effective as CritiQ Flow. This suggests that the optimization algorithm in our workflow is more effective in the scenarios of mining quality criteria from human preferences. We show the prompts generated by TextGrad in Appendix C.

CritiQ Flow surpasses single criteria. Any single criterion proposed by QuRating (Wettig et al., 2024) fails to achieve a high accuracy. Although, as highlighted in many related studies (Zhang et al., 2024; Gunasekar et al., 2023; Wei et al., 2024), the Educational Value criterion shows relatively higher consistency with human judgment, it can not comprehensively describe data quality. This suggests that compared to single criterion, and CritiQ Flow which uses multiple criteria is better.

Evolution and knowledge base help CritiQ Flow improve the performance. Ablation shows that

[3] The specific version is `gpt-4o-2024-11-20`.

both the iterative evolution process and knowledge base in our workflow help improve the accuracies. This indicates that the criteria extracted from previous work are effective in judging data quality, while still have the potential to be optimized according to the specific domain and dataset; and that the optimization process is effective in improving the criteria with only $\sim$30 human annotations.

CritiQ Scorer shows increased accuracy. Notably, CritiQ Scorer achieves higher accuracies than the direct multi-criteria voting by worker agents across all domains, despite being trained on data annotated by them. This suggests that our method effectively extracts human's inner quality evaluation criteria, and these criteria demonstrate strong generalization capability.

4.3 Continual Pretraining

We choose `Llama-3.1-3B` as the base model for the continual pretraining experiments. We sample 10B tokens from the Stack v2 and Zyda-2, and 3B from OpenWebMath. We perform uniform sampling and sampling using CritiQ Scorer with temperature $\tau = 1$ for the code and math datasets and $\tau = 0.5$ for the logic dataset. We continually train the models on the six datasets separately. Hyperparameters are shown in Appendix A.

We evaluate the continually trained models on corresponding downstream tasks, including 4 code-writing tasks: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), HumanEval+, and MBPP+ (Liu et al., 2023); 3 math problem solving tasks: GSM8k (Cobbe et al., 2021), SAT-Math (Zhong et al., 2023), and MATH (Hendrycks et al., 2021); and 2 logic reasoning tasks ARC-Challenge (Clark et al., 2018) and LogiQA (Zhong et al., 2023). Coding tasks are evaluated using EvalPlus (Liu et al., 2023), while others are evaluated by OpenCompass (Contributors, 2023). The results are shown in Table 3. The models trained on our selected high-quality subsets show improved performance on downstream tasks compared to the models trained on the uniformly sampled subsets.

Code	HumanEval / +	MBPP / +	Avg. / +
Raw	28.66 / 25.61	48.94 / 39.15	38.80 / 32.38
Stack	31.71 / 27.44	56.61 / 46.30	44.16 / 36.87
Ours	**39.02 / 33.54**	**68.73 / 48.41**	**53.88 / 40.98**

Math	GSM8k	SAT-Math	MATH	Avg.
Raw	27.60	35.00	5.50	22.70
OWM	28.51	32.27	5.80	22.19
Ours	**32.22**	**39.55**	**6.34**	**26.04**

Logic	ARC-C	LogiQA	Avg.
Raw	37.97	27.34	32.66
Zyda-2	36.61	23.50	30.06
Ours	**38.31**	**30.41**	**34.36**

Table 3: Evaluation results on downstream tasks of the continually trained model. "Raw" is the original `Llama-3.1-3B` model without any continual pretraining. "+" for HumanEval+ or MBPP+ (Liu et al., 2023). "Stack" for the Python subset of the Stack v2 (Lozhkov et al., 2024). OWM for the non-code subset of OpenWebMath (Paster et al., 2023).

5 Analysis

5.1 Evolution of Criteria Distribution

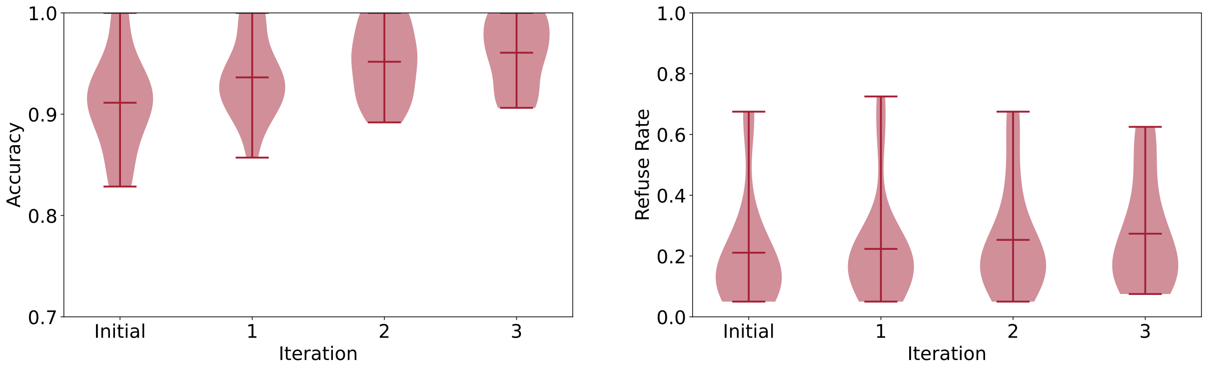
In this section, we analyze how the distribution of quality criteria evolves during the evolution process. Using the code domain as a representative example, Figure 3a shows the distribution of training accuracies for all criteria across optimization iterations. The plot reveals a clear upward trend, with the distribution progressively shifting and concentrating towards higher values as the optimization proceeds. This trend demonstrates the effectiveness of our iterative optimization process.

Notably, several criteria achieve 100% accuracy. As explained in Section 3.3, we exclude the cases where the worker agent explicitly declines to provide a judgment. Through the optimization process, the manager agent refines the criteria descriptions to be more precise about their applicability. These highly accurate criteria are particularly valuable as they effectively characterize code quality and guide the worker agent to make accurate assessments when applicable, even if they may not cover all possible scenarios.

In addition, we analyze the distribution of the refuse rate of the criteria. As shown in Figure 3b, the refuse rate falls predominantly in lower ranges, indicating that most criteria are widely applicable, while there are still a few criteria with refuse rates higher than 60% that are retained due to their high accuracy when applicable.

5.2 Criterion Refinement

The improvement in accuracy of CritiQ Flow is driven by two key processes: deprecating low-quality criteria and refining the mid-quality criteria by revising the descriptions. Deprecating the low-quality ones is something like reject sampling, which is straightforward in improving performance. In this section, we analyze how mid-quality criteria



(a) Distribution of accuracy.

(b) Distribution of refuse rate.

Figure 3: Evolution of distributions of the top-k Python code quality criteria through evolution iterations, where k is the number of the final criteria.

are refined by the manager agent.

We categorize the criteria refinement into 2 types: (1) refining the criteria retrieved from the knowledge base or generated by the manager agent, and (2) continually refining the already refined criteria. We show examples of criteria before and after refinement in Appendix F.

Refinement for Retrieved or Generated Criteria. The knowledge base is built on previous dataset research, so the criteria retrieved from the knowledge base are often too general. When the knowledge base can not provide enough criteria or some criteria are deprecated due to low accuracy, the manager agent proposes new criteria. In this case, the initial descriptions of these criteria are usually too vague, because they have not been evaluated by the worker agent, thus the manager agent does not have enough information to generate precise descriptions. As a result, the manager agent can refine those criteria by rewriting them to fit the current domain, adding detailed guidelines for the worker agent, and specifying the applicability.

Refinement for Refined Criteria. For previously refined criteria, the manager agent can further improve them by adding more detailed descriptions or examples. However, we also observe that despite the iterative optimization process, refinements do not always yield higher accuracy, especially for already well-refined criteria. Excessive refinement by the manager agent can lead to over-fitting, particularly with small training sets. To address this, we encourage the manager agent to keep the criteria simple and concise.

5.3 Majority Voting

We have demonstrated the majority voting mechanism in Section 3.3. In this section, we investigate the impact of the voting mechanism by evaluating the accuracy of combining all criteria into a single prompt. We use the same quality criteria derived by CritiQ Flow and query the worker agent for judgments. The accuracies are shown in Table 4. In all domains, the accuracy decreases without the majority voting mechanism, indicating that the majority voting mechanism is essential for the performance of CritiQ Flow.

	Code	Math	Logic	Avg.
Ours	**89.33**	**84.57**	**88.06**	**87.32**
w/o voting	84.16	81.14	85.22	83.51

Table 4: Accuracies with / without Majority Voting on the human-annotated D_{test} across 3 domains. The higher values are in bold.

6 Conclusion

We introduce CritiQ, a novel method that automatically and mine quality criteria from human preferences for data quality with limited human annotation and performs efficient data selection. It uses ann agent workflow, CritiQ Flow, to effectively summarize quality criteria from only ∼30 human-annotated test sets. pairwise comparisons. CritiQ Flow achieves high accuracies on human-annotated test sets. Efficient data selection is performed by lightweight CritiQ Scorer. We train models on our selected subset and observe increased performance on code, math and logic domains, compared to a uniformly sampled subset.

Limitations

Our work has several limitations. First, our experiments focus on three specific domains, leaving the question of general domain data selection unexplored. The challenge of guiding annotators to provide quality comparisons in general domains remains open. Furthermore, while deriving criteria directly from human-annotated pairwise comparisons reduces biases compared to handwritten criteria, human biases can not be completely eliminated from the annotation process, as defining high-quality data remains inherently subjective. Finally, due to computational constraints, we limited our approach to continual pretraining rather than pretraining from scratch, and used a relatively modest model with 3B parameters. Future work could explore scaling to larger models and more comprehensive training approaches.

References

2024. Improving LLM Pretraining by Filtering Out Advertisements.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. *arXiv preprint*. ArXiv:2310.11511 [cs] version: 1.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *arXiv preprint*. ArXiv:2108.07732 [cs].

Ralph Allan Bradley and Milton E. Terry. 1952. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*, 39(3/4):324–345. Publisher: [Oxford University Press, Biometrika Trust].

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, and Dahua Lin. 2024. InternLM2 Technical Report. *arXiv preprint*. ArXiv:2403.17297 [cs].

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint*. ArXiv:1803.05457 [cs].

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint*. ArXiv:2110.14168 [cs].

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. `https://github.com/open-compass/opencompass`.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2024. DeepSeek-V3 Technical Report. *arXiv preprint*. ArXiv:2412.19437 [cs] version: 1.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The Llama 3 Herd of Models. *arXiv preprint*. ArXiv:2407.21783 [cs].

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks Are All You Need. *arXiv preprint*. ArXiv:2306.11644 [cs].

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv preprint*. ArXiv:2103.03874 [cs].

Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. 2024. OpenCoder: The Open Cookbook for Top-Tier Code Large Language Models. *arXiv preprint*. ArXiv:2411.04905.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. *arXiv preprint*. ArXiv:2205.11916 [cs].

Wouter Kool, Herke van Hoof, and Max Welling. 2019. Stochastic Beams and Where to Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement. *arXiv preprint*. ArXiv:1903.06059 [cs].

Tomasz Korbak, Kejian Shi, Angelica Chen, Rasika Bhalerao, Christopher L. Buckley, Jason Phang, Samuel R. Bowman, and Ethan Perez. 2023. Pretraining Language Models with Human Preferences. *arXiv preprint*. ArXiv:2302.08582 [cs].

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. *arXiv preprint*. ArXiv:1711.05101 [cs].

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation. *arXiv preprint*. ArXiv:2402.19173 [cs].

Kai Lv, Xiaoran Liu, Qipeng Guo, Hang Yan, Conghui He, Xipeng Qiu, and Dahua Lin. 2024. LongWanjuan: Towards Systematic Measurement for Long Text Quality. *arXiv preprint*. ArXiv:2402.13583 [cs].

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. *arXiv preprint*. ArXiv:2303.17651 [cs].

Max Marion, Ahmet Üstün, Luiza Pozzobon, Alex Wang, Marzieh Fadaee, and Sara Hooker. 2023. When less is more: Investigating data pruning for pretraining llms at scale. *Preprint*, arXiv:2309.04564.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, C. J. Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 Technical Report. *arXiv preprint*. ArXiv:2303.08774 [cs].

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2023. OpenWebMath: An Open Dataset of High-Quality Mathematical Web Text. *arXiv preprint*. ArXiv:2310.06786 [cs].

Jan Peters and Stefan Schaal. 2007. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 745–750, New York, NY, USA. Association for Computing Machinery.

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

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2022. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. *arXiv preprint*. ArXiv:2112.11446 [cs].

Noveen Sachdeva, Benjamin Coleman, Wang-Cheng Kang, Jianmo Ni, Lichan Hong, Ed H. Chi, James Caverlee, Julian McAuley, and Derek Zhiyuan Cheng. 2024. How to Train Data-Efficient LLMs. *arXiv preprint*. ArXiv:2402.09668 [cs].

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022. Self-critiquing models for assisting human evaluators. *arXiv preprint*. ArXiv:2206.05802 [cs].

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Yury Tokpanov, Paolo Glorioso, Quentin Anthony, and Beren Millidge. 2024. Zyda-2: a 5 Trillion Token High-Quality Dataset. *arXiv preprint*. ArXiv:2411.06068 [cs].

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

Zengzhi Wang, Rui Xia, and Pengfei Liu. 2023. Generative AI for Math: Part I – MathPile: A Billion-Token-Scale Pretraining Corpus for Math.

Yuxiang Wei, Hojae Han, and Rajhans Samdani. 2024. Arctic-SnowCoder: Demystifying High-Quality Data in Code Pretraining.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2019. Ccnet: Extracting high quality monolingual datasets from web crawl data. *Preprint*, arXiv:1911.00359.

Alexander Wettig, Aatmik Gupta, Saumya Malik, and Danqi Chen. 2024. QuRating: Selecting High-Quality Data for Training Language Models. *arXiv preprint*. ArXiv:2402.09739 [cs].

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint*. ArXiv:2308.08155 [cs].

Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong, Shihan Do, Wenyu Zhan, Xiao Wang, Rui Zheng, Tao Ji, Xiaowei Shi, Yitao Zhai, Rongxiang Weng, Jingang Wang, Xunliang Cai, Tao Gui, Zuxuan Wu, Qi Zhang, Xipeng Qiu, Xuanjing Huang, and Yu-Gang Jiang. 2024. Enhancing llm reasoning via critique models with test-time and training-time supervision. *Preprint*, arXiv:2411.16579.

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy Liang. 2023. Data Selection for Language Models via Importance Resampling. *arXiv preprint*. ArXiv:2302.03169.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. TextGrad: Automatic "Differentiation" via Text. *arXiv preprint*. ArXiv:2406.07496 [cs].

Yifan Zhang, Yifan Luo, Yang Yuan, and Andrew Chi-Chih Yao. 2024. Autonomous Data Selection with Language Models for Mathematical Texts. *arXiv preprint*. ArXiv:2402.07625 [cs].

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models. *arXiv preprint*. ArXiv:2304.06364 [cs].

A Hyperparameters

A.1 Hyperparameters for CritiQ Flow

We have manually tried different sets of hyperparameters and the chosen hyperparameters for the final experiments are shown in Table 5.

	Code	Math	Logic
#Criteria	20	20	20
#Iterations	3	5	3
Retrieval Threshold	0.5	0.5	0.5
High Threshold	0.9	0.8	0.8
Low Threshold	0.8	0.7	0.7
Final Threshold	0.9	0.7	0.8

Table 5: Hyperparameters for CritiQ Flow.

A.2 Hyperparameters for CritiQ Scorer

We use the `trl` (von Werra et al., 2020) library to train CritiQ Scorers. On the 3 domains, we train each CritiQ Scorer using AdamW (Loshchilov and Hutter, 2019) optimizer with learning rate 2×10^{-5} and weight decay 0.01 for 4 epochs. The learning rate warmups in the first 20% training steps and cosine decay in the rest steps. We truncate the text longer than $32,768$ tokens. The global training batch size is 128. We randomly select 5% from the CritiQ Scorer training set D_{agent} as the validation set, and use the rest to train the scoring model. We save the model every 50 training steps and select the checkpoint with the best validation accuracy as the final CritiQ Scorer.

A.3 Hyperparameters for Continual Pretraining

We use AdamW (Loshchilov and Hutter, 2019) optimizer with the maximum learning rate 1×10^{-4}, the minimal learning rate 1×10^{-5}, and weight decay 0.01 for 4 epochs. The learning rate increases in the first 5% training steps, and cosine decays in the rest steps. The training sequence length is 8192 and global batch size is $4M$ tokens. Each model is trained on 32 NVIDIA H800 GPUs.

B Annotation

B.1 Annotators

Our annotation team consists of three annotators for each domain (code, math, and logic). The annotators are paper authors who meet the following qualifications:

- Hold bachelor's or master's degrees
- Have multiple years of professional programming experience
- Possess foundational mathematical knowledge
- Demonstrate competency in logical reasoning

The annotators volunteered their time without additional compensation. As authors of the paper, they had a vested interest in producing high-quality annotations, since the annotation results directly impacted the experimental outcomes and overall research quality.

B.2 Annotation Guidelines

B.2.1 Annotation Guidelines for Code

Please compare the two Python Code files and choose the one of higher quality.

Low-quality code often has the following characteristics:

- The code is badly formatted or has syntax errors.
- The code consists solely of comments or package imports, which is non-informative.
- The code only consists of simple class or function definitions, which is hard to understand without other files.
- The code just defines meaningless variables while do not perform any operations.
- The code is too simple.
- The code contains too much hard-coded data or is a configuration or an entrypoint file to a larger project, which is not helpful in learning programming.

High-quality code often has the following characteristics:

- The code is educational for code starters, which shows basic programming principles, design patterns, or data structures.
- The code is a solution to an algorithm problem, which is beneficial for learning algorithm.
- The code is well-structured with proper code comments, which leads to high readability and maintainability.
- The code shows clear purpose and can accurately solve certain kind of problems, while keeps extensible and flexible.
- The code has self-contained classes or functions that can be understood without other files, which shows high simplicity and reusability.

Choose the better one of A and B according to the above guidelines and your preferences for code quality. If the two files are of similar level, answer C.

B.2.2 Annotation Guidelines for Math

Please compare the two text data related to math and choose the one of higher quality.

High-quality math data show significant mathematical intelligence and is educational for math learners. Mathematical quality can be evaluated based on several key aspects:

(1) Logical Structure: Content should demonstrate clear reasoning with properly structured arguments, proofs and deductions, avoiding inconsistencies or unjustified assumptions;

(2) Mathematical Rigor: Expressions should use precise and consistent notation, terminology and symbols throughout, with all necessary steps clearly stated;

(3) Pedagogical Value: The content should be build systematically from fundamentals to advanced ideas, including instructive examples that reinforce understanding;

(4) Conceptual Depth: Material should go beyond elementary arithmetic to explore deeper mathematical concepts and problem-solving techniques, showing connections between different ideas;

(5) Technical Accuracy: Content should be free of mathematical errors, misconceptions, ambiguous notation, or incorrect terminology that could impede understanding.

High-quality mathematical content will excel in these areas while maintaining accessibility, whereas lower-quality content may be lacking in one or more of these essential aspects.

Choose the better one of A and B according to the above guidelines and your preferences for mathematical quality. If the two texts are of similar level, answer C.

B.2.3 Annotation Guidelines for Logic

Compare the following two texts, determine which one better requires and promotes logical thinking by evaluating these three essential criteria:

1. Does understanding later content require careful reasoning from previous information?

- Positive: Text that builds logical arguments progressively.

- Negative: Text that can be understood superficially without deeper thinking Contextual Integration.

2. Does comprehension require connecting multiple pieces of evidence or ideas?

- Positive: Text with interconnected logical elements.

- Negative: Simple chronological narratives or disconnected descriptions Structured Interpretation.

3. Can the content be understood through clear rational analysis?

- Positive: Text with well-defined logical relationships.

- Negative: Ambiguous literary expressions with multiple subjective interpretations.

Choose the better one of A and B according to the above guidelines and your preferences for logical quality. If the two texts are of similar level, answer C.

C Prompts Generated by TextGrad

We show the prompts generated by TextGrad for the three domains in Section E. The quality criteria are in bold.

Code

Task Instruction\nYou are tasked with performing a comprehensive comparison of the quality and structure of two Python code files. Evaluate them based on **readability, efficiency, adherence to Python coding standards (PEP 8), and maintainability**. Highlight strengths and weaknesses for each file and suggest specific improvements where necessary. \n\n## Code File A\n{A}\n\n## Code File B\n{B}

Math

Compare the Mathematical Quality of Two Solutions\nPlease evaluate the mathematical quality of the two provided solutions. Consider factors such as **correctness, clarity, logical reasoning, and mathematical rigor** in your assessment. Once you have thoroughly reviewed both solutions, choose "A" or "B" to identify the solution that exhibits superior mathematical quality.\n\n[A]\n{A}\n\n[B]\n{B$}

Logic

Assess the logical consistency between the two text pieces provided below. Identify which text is more **logically consistent**\n of A, B, or if they are equally consistent. Clearly explain your reasoning behind the evaluation.\n\n[A]\n{A}\n[/A]\n\n[B]\n{B}\n[/B]

D Responsible NLP Research Statements

We used generative AI to assist in this work. We used GitHub Copilot for short-form input assistance when writing the code. We used ChatGPT and Claude for paraphrasing and polishing the original content in the paper.

The datasets used in this work are publicly accessible. The usage of the Stack v2 is under Terms of Use for The Stack v2 [4]. The usage of OpenWebMath is under ODC-By 1.0 license [5] and the CommonCrawl ToU [6]. The usage of Zyda-2 is under the terms of Open Data Commons License [7].

We used `gpt-4o` for the experiments, which is under OpenAI's Terms of Use [8]. We used `Qwen2.5-72B-Instruct`, whose weight is distributed under Qwen LICENSE AGREEMENT [9]. We trained `Llama-3.1` respect to LLAMA 3.1 COMMUNITY LICENSE AGREEMENT [10].

E Prompts

E.1 Prompts for Knowledge Base

Judge if a paper releases a dataset.

There is a research paper about artificial intelligence.\n\nTitle: <TITLE>\nAbstract: <ABSTRACT>\n\nInstruction: Does this paper propose a dataset? Return your answer in the following format:\n\n"‘ json { "analysis": "Your analysis. For example, the main contribution of the paper.", "dataset": "The name of the dataset if it is proposed. Otherwise, answer 'N/A'.", "answer": "Yes/No/Unsure" } "‘

Extract quality criteria from papers.

There is a research paper about artificial intelligence which proposed a new dataset named <DATASET_NAME>.\n\n[BEGIN_OF_PAPER]\n <PAPER_CONTENT>\n[END_OF_PAPER]\n\nI

[4] https://huggingface.co/datasets/bigcode/the-stack-v2

[5] https://opendatacommons.org/licenses/by/1-0/

[6] https://commoncrawl.org/terms-of-use/

[7] https://opendatacommons.org/licenses/by/1-0/

[8] https://openai.com/policies/terms-of-use/

[9] https://huggingface.co/Qwen/Qwen2.5-72B-Instruct/blob/main/LICENSE

[10] https://www.llama.com/llama3_1/license/

want to learn how to distinguish between data of high and low quality from the process of constructing the <DATASET_NAME> dataset. Please conclude the criteria for determining data quality from the paper.\n\n- The criteria should be able to used to filter the data for the dataset.\n - The criteria should be general enough to be applied to other datasets.\n - If the paper proposed a data processing method, you should describe the criteria for the processed data which may be of higher quality.\n - You should not just copy the criteria from the paper, but summarize them in your own words.\n\n“‘json { "name_of_the_criterion": "description_of_the_criterion", "name_of_another_criterion": "description_of_another_criterion", ... } \n\nThe names of criteria should be a descriptive word. The descriptions should show what the criteria is about and how it can be used to determine if a data record should be included in the dataset. “‘

Retrieve Code Criteria

Instruction\nIs this criterion applicable for evaluating the quality of Python code?\n\n# Criterion\n<CRITERION>: <DESCRIPTION>\n\nYou should simply reply ’yes’ or ’no’.

Retrieve Math Criteria

Is the following criterion applicable to measure the mathematical quality of text data?\n\n### Criterion\n*<CRITERION>*: <DESCRIPTION>\n\nYou should simply reply ’yes’ or ’no’.

Retrieve Logic Criteria

Instruction\nIs the following criterion applicable to evaluate the logical quality of text data?\n\n# Criterion\n<CRITERION>: <DESCRIPTION>\n\nYou should simply reply ’yes’ or ’no’.

E.2 Domain Specific Prompts for Worker Agents

Pairwise Judgment for Code

Instruction\nGiven criterion **criterion**, compare two Python code files and determine which one human annotators will consider to be of higher quality.\n\n## A\n{A}\n\n## B\n{B}\n\n# Criterion\n**{criterion}**: {description}

Pairwise Judgment for Math

Instruction\nGiven criterion **{criterion}**, evaluate and determine which of the two text data is of higher quality in mathematics.\n\n[DATA_A]\n{A}\n [/DATA_A]\n\n[DATA_B] \n {B}\n [/DATA_B]\n\n# Criterion\n**{criterion}**: {description}

Pairwise Judgment for Logic

Which text piece of A and B is more logical based on **{criterion}**?\n\n{criterion}: {description}\n\n[A]\n{A}\n[/A]\n\n[B]\n{B}\n[/B]

E.3 Domain Specific Prompts for Manager Agents

Generate Initial Code Criteria

List and describe 20 criteria on how human compare the overall quality of two Python code files.

Generate Initial Math Criteria

List and describe 20 criteria on evaluating whether a text data is high quality math data.

Generate Initial Logic Criteria

List and describe 20 criteria to tell which is more logical of two text pieces.

E.4 General Prompts for CritiQ Flow

The full prompts of CritiQ Flow are complex. We simply list the source code here. Details can be checked in our released CritiQ software.

```
MANAGER_PROMPT_POSTFIX="""\n\nYour
    response should be in the following **json**
     format:
```json
{
" name_of_the_criterion ": " Detailed
     description for the criterion such as
    what it is , how it can be evaluated ,
    when it is applicable , and other
     relevant information . Be specific and
     detailed while keep concise .",
 ...
}
```"""
MANAGER_PROMPT_TEMPLATE="Give {
    n_criteria} criteria for evaluating data quality
     ."
ACCURACY_PROMPT="The worker agents
    had evaluated data pairs aginst these criteria
    . The accuracy of each criterion is as
    follows :"
GOOD_CRITERIA_PROMPT_TEMPLATE="
    Accuracies of criteria {criteria} are over {
```

```
    threshold}. They are good criteria."
MID_CRITERIA_PROMPT_TEMPLATE="The
    accuracy of {criterion_name} is over {
    threshold_0} but less than {threshold_1}. It
    can be improved. Here is the raw
    description of the criterion :"
MID_CRITIQUE_PROMPT="This is an
    incorrect case:"
MID_A_PROMPT_TEMPLATE="[
    BEGIN_OF_A]\n{}\n[/END_OF_A]"
MID_B_PROMPT_TEMPLATE="[
    BEGIN_OF_B]\n{}\n[/END_OF_B]"
MID_HOWEVER_PROMPT_TEMPLATE="
    Against this criterion, the worker agent chose
    {wrong} as better, but the correct answer is
    { correct }. Here is how the worker agent
    thinks :\n\n{thought}"
MID_REFLECTION_PROMPT="""Please
    analyze this incorrect case together with the
    worker agent's thought. Based on your
    anaylsis , ples provide your critique for
    how to write a better description of
    this critierion to guide the worker
    make correct judgment or properly
    indicate inapplicable situations for
    this criterion .

Your response should be in the following **
    json** format:
{
" analysis ": "Your analysis here .",
" critique ": "How this criterion can be
    improved. Please just point out the key
    points in a few sentences ."
}"""
MID_REFINE_PROMPT_TEMPLATE="There
    are the critiques for the wrong choices.\n\n
    {}\n\nBased on the above critiques, please
    improve the description for this
    criterion to make worker agents get
    higher accuracy. For exmaple, what it is
    , how it can be evaluated, when it is
    applicable , and other relevant
    information. Be specific and detailed
    while keep concise ."
MID_FORMAT_PROMPT_TEMPLATE='
    Return the improved description in the
    following **json** format:\n\n{{"{
    criterion_name}": "The improved description
    "}}'
LOW_PROMPT_TEMPLATE="Criteria {criteria
    } have an accuracy of less than {threshold_0
    }. They should be removed from the
    criteria list . Please provide {num} new
    criteria . The new ones should not be
    duplicated with the above ones."
LOW_FORMAT_PROMPT="""Return the new
    criteria in the following **json** format:\n\n
```json
{
" your_better_criterion_here ": "Detailed
    description for the criterion , including
    what it is, how it can be evaluated,
    when it is applicable , and other
    relevant information. Be specific and
    detailed while keep concise .",
...
}
```"""
PAIR_WORKER_PROMPT_POSTFIX="""
Your response should be in the following **
    JSON** format:
```json
{
" analysis_a ": "Analyze A based on the given
    criterion .",
" analysis_b ": "Analyze B based on the given
    criterion .",
"thought": "Compare A and B.",
"answer": "A / B / None"
}
```
Return None if any of the following
    conditions are met:
- The criterion is not applicable to this
    pair of data pieces.
- They are of the same quality .
- You are unsure.
"""
PAIR_WORKER_PROMPT="Which is better in
    the aspect of **{criterion}**?\n\n{criterion
    }: { description }\n\n[DATA_A]\n{A}\n[/
    DATA_A]\n\n[DATA_B]\n{B}\n[/DATA_B
    ]"
CRITERION_FORMAT_TEMPLAT="\n\n[
    CRITERION]\nCriterion: {name}\n\
    nDescription: {desc}\n[/CRITERION]"
```

F Examples for Criteria Refinement

F.1 Generated Criterion

Criterion algorithm_efficiency

Before refinement

This criterion assesses the efficiency of the algorithm implemented in the code. It measures time complexity, space complexity, and overall performance under different input sizes. By comparing how well the code scales and performs, evaluators can determine which implementation is superior in terms of resource optimization. It is particularly applicable to competitive programming or performance-critical applications.

Criterion algorithm_efficiency

After refinement

The 'algorithm_efficiency' criterion evaluates the computational performance of algorithms implemented within the code, focusing on measurable metrics such as time complexity, space complexity, and overall scalability. It assesses how well the code optimizes operations under varying input sizes and work complexity. This criterion is specifically applicable when the code implements clear and testable algorithms designed to solve computational or algorithmic tasks (e.g., sorting, searching, graph traversal). It is not concerned with the broader functionality, structure, or aesthetics of the code unless they directly impact efficiency.
How to evaluate: Workers should focus on examining the implementation of algorithms for efficiency improvements based on established computational principles. Consider aspects like: 1. Does the algorithm exhibit better performance by reducing computational steps (time complexity)? 2. Does it use fewer resources, such as memory, to achieve its output (space complexity)? 3. How does the algorithm perform under different input sizes (scalability)? 4. Are there opportunities to optimize the algorithm further (e.g., using better data structures or reducing redundant operations)?
When it is applicable: This criterion applies when both code files contain algorithms or computational content that can be directly analyzed for efficiency in terms of performance under realistic conditions. For example, files implementing tasks like sorting, searching, mathematical computations, or data processing should be evaluated under this criterion.
When it is inapplicable: This criterion should be marked 'Not Applicable' when: 1. Neither file contains algorithmic or computational content (e.g., configuration files, import statements, or static setup code). 2. The files are of vastly different purposes or functionalities, making a direct comparison of algorithmic efficiency irrelevant. 3. The code involves orchestration logic (e.g., API calls) rather than computational tasks. 4. The algorithms are incomplete, trivial, or lack sufficient context for meaningful evaluation (e.g., placeholders, stubs, or boilerplate code).
Additional information: Workers should avoid judging efficiency based on perceived complexity or verbosity. Simpler code that efficiently achieves the desired result should be favored over unnecessarily complex solutions. Ensure all evaluations consider the problem the algorithm is designed to solve in its context. To assist with clarity, workers should consider explicitly marking the criterion inapplicable when one or both files fail to meet the stated requirements for algorithm efficiency evaluation. Clear examples: - Applicable: Comparing two sorting algorithms for time and space complexity. - Inapplicable: Comparing a configuration file to a data processing algorithm. - Inapplicable: Comparing an I/O-bound script with CPU-intensive code.
This improved description ensures that workers consider the scope and context of 'algorithm_efficiency' before making decisions, reducing ambiguity and inaccuracies.

F.2 Retrieved Criterion

Criterion error_analysis

Before refinement

A thorough analysis of the reasoning flaws must be present. This includes understanding the causes of errors and what correct reasoning should have been applied at that point.

Criterion error_analysis

After refinement

Error analysis evaluates the quality and depth of reasoning related to identifying, diagnosing, handling, and mitigating potential or actual errors within the code. This includes examining error-handling mechanisms such as exception blocks, validation checks, logging, or any other explicit strategies to anticipate and address errors. Additionally, it considers the code's explanation or reasoning about errors, focusing on detail and thoroughness in addressing potential edge cases or failure points. To evaluate error analysis, workers should consider the following steps: (1) Identify the presence of error-handling logic or mechanisms in the code (e.g., try-except blocks, assertions, logging); (2) Assess whether the provided error-handling logic is appropriate for the scope and context of the code; (3) Pay attention to any accompanying comments or documentation explaining the approach to mitigating errors; and (4) Evaluate whether patterns of reasoning about errors are logical and well-structured, including how edge cases are anticipated.
This criterion is applicable only to code that contains logical processes, algorithms, or decision-making components where errors are likely to occur and need to be reasoned about or handled. It should be marked inapplicable for code that lacks relevant error-handling context, such as configuration files, boilerplate code, or import-only scripts. In cases where both pieces of code lack any mention or handling of errors, the criterion should also be deemed inapplicable, and no preference should be made.
Key aspects to avoid include judging the code based

on its overall complexity, functionality, or modularity unless they directly affect error analysis. Highlighting superficial error handling or assuming error-free code does not inherently satisfy this criterion. Workers should focus on explicit reasoning about errors and how the code mitigates or avoids potential failures. Concrete examples of good error analysis include thorough exception handling with explanations, detailed error logging, validations targeting specific failure scenarios, and robust test cases explicitly aimed at uncovering edge cases or logical flaws.

F.3 Refined Criterion

Criterion commented_context

Before refinement

The 'commented_context' criterion evaluates the presence, relevance, and quality of comments or documentation within a code file, ensuring they enhance understanding of the code's purpose, functionality, and any non-obvious logic. Comments should provide meaningful insights about the code's intent, clarify complex or non-intuitive sections, and offer context, such as explaining critical operations or unusual design decisions. This criterion does not favor the mere presence of comments or their verbosity but instead focuses on their necessity and utility in aiding comprehension.
Approach for evaluation: Workers should assess whether comments are directly relevant to specific parts of the code and whether they provide significant contextual value to understanding its intent and usage. For instance, comments explaining business logic, algorithmic choices, or intricate areas of code are highly valuable. Irrelevant, redundant, or excessively verbose comments that do not add clarity should not be positively weighted. Self-documenting code, where the use of clear variable/function names and logical structure makes it inherently understandable, should not be penalized for a lack of comments.
Applicability: This criterion is most relevant when comparing code that requires additional explanation due to complexity or specialized logic. It is less applicable or should be marked inapplicable when both files contain minimal or no comments, but their code is simple and self-explanatory. Examples include boilerplate files, import-only files, or scripts so straightforward that no additional context is needed.
Additional considerations: Workers should not rely on style or verbosity as sole indicators of quality. Comments that are overly generic (e.g., 'This is a for loop') or unrelated (e.g., boilerplate licensing information) should not factor into the evaluation. When both files feature sufficient documentation for their respective levels of complexity, preference should be given to concise, context-rich comments over verbose or unnecessary ones. If both files lack meaningful comments and are equally understandable without additional documentation, this criterion may not provide a basis for comparison.

Criterion commented_context

After refinement

The 'commented_context' criterion evaluates the presence, relevance, and necessity of comments or documentation within a code file. Comments should meaningfully enhance understanding by providing critical context, explaining complex logic, or clarifying non-obvious design decisions. The value of comments should be judged by their ability to aid comprehension, rather than their quantity or verbosity. High-quality comments are concise, appropriately placed, and directly related to the code's purpose and functionality. For example, comments explaining intricate algorithms, decision-making processes, or domain-specific details are valuable, whereas redundant, trivial, or boilerplate comments (e.g., licensing headers, generic statements like 'this is a loop') are not.
Evaluation Steps: 1. Assess whether the file contains comments, and if present, determine whether they address essential aspects of the code's logic, design, or purpose. 2. Focus on relevance: Identify whether the comments clarify concepts that are not immediately understandable from the code structure itself. 3. Consider necessity: Evaluate if the complexity of the code requires additional explanation, or if the code is inherently self-explanatory (e.g., simple utility scripts or well-named variables/functions). 4. Judge quality: Favor concise, meaningful comments over verbose, generic, or redundant ones. 5. Evaluate whether comments contribute to maintainability by providing future developers with clear insights into the code's intent or potential edge cases.
Applicability: - This criterion is applicable when the code includes non-obvious logic, intricate design, or contextual details that are essential for understanding. For example, it applies to files with algorithms, configuration settings, or any code where additional clarification adds significant value. - It is not applicable for files containing minimal or self-explanatory code, such as import statements, trivial scripts, or boilerplate content, where comments are unnecessary. - When comparing two files, if both lack comments but are sufficiently self-documenting, this criterion should be marked as inapplicable rather than favoring one file over the other based on the absence of comments.
Additional Notes: - Avoid penalizing files that are simple and naturally clear without requiring comments. Instead, prioritize whether the comments add actual value relative to the code's complexity. - Clear examples should be provided to illustrate appropriate use, such as comments that explain unexpected behavior or unconventional approaches, versus meaningless or excessive commentary that does not enhance comprehension. - Do not elevate files with verbose or irrelevant comments over those with concise, targeted, and effective comments. Focus on substance, not volume. - Metadata comments, like licensing information, may be required for compliance but should not be counted as contributing to 'commented context' unless they add value to the understanding of the code.
In summary, this criterion focuses on whether comments are necessary, relevant, and useful in providing additional context or understanding. It recognizes

that not all code requires extensive commenting and explicitly allows for marking the criterion as ’Not Applicable’ in cases of minimalistic, self-explanatory, or trivial files.