

Aligning LLMs through Multi-perspective User Preference Ranking-based Feedback for Programming Question Answering

Hongyu Yang
School of Computer Science and
Technology, University of Science and
Technology of China
Hefei, China
hongyuyang@mail.ustc.edu.cn

Liyang He
School of Computer Science and
Technology, University of Science and
Technology of China
Hefei, China
heliyang@mail.ustc.edu.cn

Min Hou
Hefei University of Technology
Hefei, China
minho@mail.ustc.edu.cn

Shuanghong Shen
School of Artificial Intelligence and
Data Science, University of Science
and Technology of China
Hefei, China
closer@mail.ustc.edu.cn

Rui Li
School of Computer Science and
Technology, University of Science and
Technology of China
Hefei, China
ruili2000@mail.ustc.edu.cn

Jiahui Hou*
School of Computer Science and
Technology, University of Science and
Technology of China
Hefei, China
jhhou@ustc.edu.cn

Jianhui Ma
School of Computer Science and
Technology, University of Science and
Technology of China
Hefei, China
jianhui@ustc.edu.cn

Junda Zhao
University of Toronto
Toronto, Canada
junda.zhao@mail.utoronto.ca

ABSTRACT

Code Community Question Answering (CCQA) seeks to tackle programming-related issues, thereby boosting productivity in both software engineering and academic research. Recent advancements in Reinforcement Learning from Human Feedback (RLHF) have transformed the fine-tuning process of Large Language Models (LLMs) to produce responses that closely mimic human behavior. Leveraging LLMs with RLHF for practical CCQA applications has thus emerged as a promising area of study. Unlike standard code question-answering tasks, CCQA involves multiple possible answers, with varying user preferences for each response. Additionally, code communities often show a preference for new APIs. These challenges prevent LLMs from generating responses that cater to the diverse preferences of users in CCQA tasks. To address these issues, we propose a novel framework called **Aligning LLMs through Multi-perspective User Preference Ranking-based Feedback for Programming Question Answering (ALMupQA)** to create user-focused responses. Our approach starts with Multi-perspective Preference Ranking Alignment (MPRA), which synthesizes varied user preferences based on the characteristics of answers from code communities. We then introduce a Retrieval-augmented In-context

Learning (RIL) module to mitigate the problem of outdated answers by retrieving responses to similar questions from a question bank. Due to the limited availability of high-quality, multi-answer CCQA datasets, we also developed a dataset named StaCCQA from real code communities. Extensive experiments demonstrated the effectiveness of the ALMupQA framework in terms of accuracy and user preference. Compared to the base model, ALMupQA showed nearly an 11% improvement in BLEU, with increases of 20% and 17.5% in BERTScore and CodeBERTScore, respectively.

KEYWORDS

question answering, reinforcement learning, large language models, in-context learning

ACM Reference Format:

Hongyu Yang, Liyang He, Min Hou, Shuanghong Shen, Rui Li, Jiahui Hou*, Jianhui Ma, and Junda Zhao. 2024. Aligning LLMs through Multi-perspective User Preference Ranking-based Feedback for Programming Question Answering. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated their success in the field of open-domain Question Answering (QA) [3, 30, 33]. To align LLMs to users preference in domain-specific QA tasks, Reinforcement Learning from Human Feedback (RLHF) enables the alignment for human-like response generation [34]. For example, in code question answers [39], the LLM may produce redundant responses (e.g., a_i in Figure 1). Utilizing RLHF can effectively achieve precise generation behavior control. However, the application of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

© 2024 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

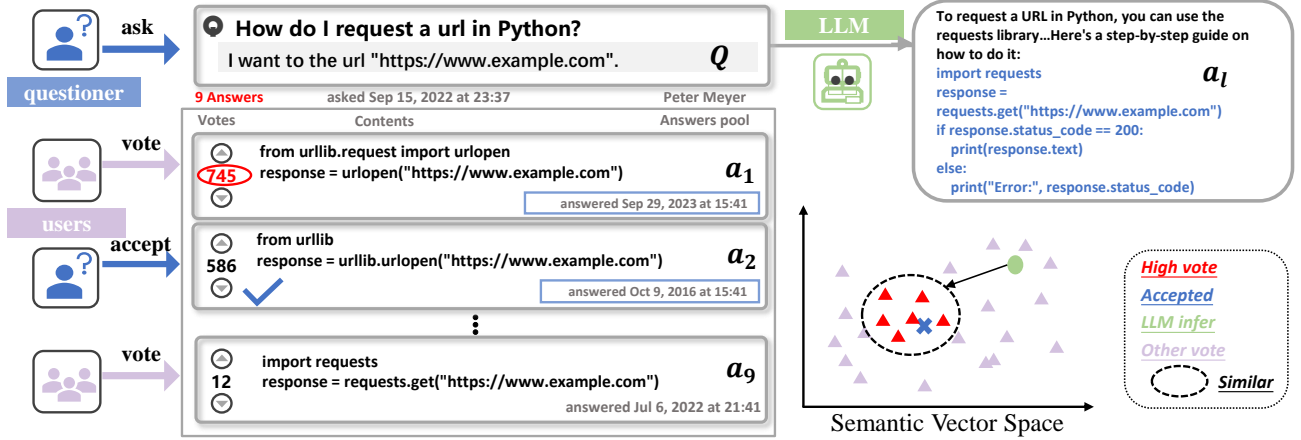


Figure 1: An example of a Code Community Question Answering. It encompasses key elements: a question Q , a pool of answers $\{a_1, \dots, a_9\}$. Each a_i contains its text of content, the number of votes, and a label indicating whether the answer has been accepted by the questioner. Additionally, in the semantic vector space, there exists a certain distance between the LLM-based answers a_l , the questioner-accepted answer a_2 , and the users-preferred answers a_1 .

LLMs to real-world Code Community Question Answering (CCQA) tasks and the related preference alignment research remains an underexplored domain.

In recent years, CCQA has gained increasing significance in both academia [9, 24, 31, 61] and industry [2, 18, 40?]. It focuses on the code question-answer interactions among users in code communities (e.g., Stack Overflow¹). Unlike conventional QA task [23, 44, 51], CCQA exhibits three distinct characteristics. First, a question typically does not have just one answer, and as indicated in Table 1, nearly 46% of questions receive more than two answers, with some boasting an answer pool as large as 30. Second, each answer encompasses not only the textual content but also additional interactive elements, such as votes from other users, which reflect rich user preferences. Third, different users exhibit varying preferences for different answers to a given question. For example, in Figure 1, a questioner posed a question Q and accepted answer a_2 from the pool of answers $\{a_1, \dots, a_9\}$, while some users favored answer a_1 with the highest votes.

Based on the characteristics mentioned above, to enable LLMs to be effectively applied in CCQA tasks and generate responses that satisfy diverse user preferences, several critical problems still need to be addressed. First, some works [41] have attempted to align LLM responses with human preferences by using the accepted answer (e.g., a_2 in Figure 1) as the alignment target. However, the accepted answer may not reflect the preferences of all users, as the answer chosen by the questioner may not be favored by other users. Second, although some studies [9, 11, 28, 61] have begun to focus on entire answers and have introduced content-based ranking methods, none have yet considered the inherent preferences of diverse users in CCQA and the feedback from LLMs. Third, it is worth noting that people’s preferences shift with API updates in code communities, as they tend to choose newer versions of APIs. However, the accepted answer may suffer from being outdated, as

in the field of programming, API updates occur rapidly. For example, in Figure 1, the “urllib” API in answer a_2 is applicable to Python 2 but deprecated in Python 3.

To overcome the above problems and limitations, we propose a novel multi-perspective preference ranking method for aligning LLMs on CCQA, which we call ALMupQA. ALMupQA primarily comprises two modules: Multi-perspective Preference Ranking Alignment (MPRA) and Retrieval-augmented In-context Learning (RIL). In MPRA, we first propose three scores as preference ratings for the answers, including a questioner-perspective bias score to assess the discrepancy between the accepted answer and other answers, a users-perspective vote score to reflect the collective preferences of other users, and a LLMs-perspective content score for evaluating the semantic quality of the answer content. Then, we introduce a preference ranking alignment method to utilize the three scores iteratively to identify the preference order of answers and optimize the alignment with user preferences using a list-wise contrastive loss. Besides, RIL aims to address the issue of outdated answers by retrieving answers to similar questions from the question bank and employing them as few-shot examples to enhance the effectiveness of the generated responses. Finally, due to the current lack of relevant datasets, we constructed a high-quality dataset, StaCCQA², from real-world code communities. Extensive experiments validated the effectiveness of the ALMupQA method in terms of accuracy and user preference. In summary, the paper makes three main contributions:

- We propose a novel method, ALMupQA, to achieve preference alignment in the multi-perspective Community Code Question Answering task, which is an industrial practice with practical applications.
- In ALMupQA, we introduce the MPRA method to align preferences from a ranking perspective, taking into account the unique

¹<https://stackoverflow.com>

²Our dataset is accessible on <https://anonymous.4open.science/r/PETCoQA-810A>.

characteristics of CCQA answers, and propose RIL to address the issue of potentially outdated code.

- We constructed a multi-user preference dataset, StaCCQA, from the real-world code community. Comprehensive experiments on this dataset have evaluated the performance of ALMupQA against other open-source and proprietary LLM baselines. The results demonstrate the superiority of ALMupQA, establishing it as a robust foundational model for CCQA research.

2 RELATED WORKS

2.1 Code Community Question Answering

In code communities, programmers can both seek and share expertise, exemplifying the trend of collaborative problem-solving and knowledge exchange in software development. Code Community Question Answering (CCQA) is a fundamental task in code communities, which involves programming issues generated from user-posted questions and relevant answers to these questions [25]. Given the substantial differences between structured code and text, CCQA systems must possess the ability to comprehend both programming and natural languages, rendering this task highly challenging [24].

Within CCQA, we can identify numerous research topics, such as predicting answerable questions [4], assessing answer quality [13, 40, 54], answer generation [61], and answer ranking [2, 10, 14, 24]. These answer ranking methods typically employ classical deep-learning models to utilize the answer text [61] and the fundamental characteristics of the user [14]. For instance, L2R [10] followed a learning to rank approach based on different groups of features like features referred to the users, stylistic or structural features. RCNN [61] employed Gated Recurrent Units (GRU) with thread-level features for ranking answers. Other research [2] utilized recency and quality as criteria for ranking responses. However, few study has considered the inherent preferences of diverse users and LLM feedback. Therefore, exploring the ranking of answers based on preferences through the utilization of LLMs for alignment is a worthy endeavor.

2.2 Preference Alignment for Question Answering

In recent years, large language models (LLMs) [3, 33, 46, 47, 53] have driven increasingly diverse applications, demonstrating notable expertise in question answering. By fine-tuning on extensive datasets across various programming domains, LLMs have also attained proficiency in synthesizing programs that are both syntactically correct and functionally accurate [8, 20, 32, 41, 49, 58]. This capability enables them to adeptly navigate the complexities of programming problems, including conceptual understanding, code generation, API utilization, and debugging [12, 16, 36].

Recently, reinforcement learning from human feedback (RLHF) [34, 41, 45] has emerged as a milestone method for aligning with human preferences. This approach typically employs the Bradley-Terry model to optimize the neural network’s reward function, followed by fine-tuning the language model using reinforcement learning algorithms, most commonly proximal policy optimization (PPO) [42], to maximize the given reward. Moreover, due to the

sensitivity of RL parameters and the complex three-stage process of RLHF, numerous preference alignment methods have been proposed. For instance, RRHF [52] introduced a boundary ranking loss function to optimize LLMs without requiring an additional reward model. DPO [39] introduced a direct preference optimization method, treating LLMs themselves as the reward model. PRO [43] optimizes complex preference data through a listwise ranking loss function. Crucially, LLMs exhibit their unique stylistic preferences in content generation, adeptly leveraging retrieved knowledge from prompts. Inspired by these insights, we propose aligning with human preferences through multi-perspective preference scoring by iteratively ranking the preference scores of all answers to a given question, rather than aligning preferences via a reward model.

3 PRELIMINARIES

3.1 Task Formulation

Our overall target is to design a multi-perspective preference alignment to guide a Large Language Model (LLM), denoted as \mathcal{M} , in generating answers that synthesize diverse user preferences with a real-world code community question answering dataset $\mathcal{D} = \{(q_i, \{a_1^i, a_2^i, \dots, a_{N_i}^i\}) \mid i \in \{1, 2, \dots, N\}\}$. Here, q_i represents the i^{th} question, and $\{a_1^i, a_2^i, \dots, a_{N_i}^i\}$ represents the pool of answers for q_i . We denote $a = (c, v, a_c)$, with c being the content of answer a ; v being the votes for answer a ; and $a_c \in \{0, 1\}$ representing whether the answer a is accepted by the questioner. Formally, any q or c is a sequence of tokens, denoted as $t = \{t_i \mid t_i \in \mathcal{C} \text{ or } t_i \in \mathcal{T}\}$, where t_i denotes the i -th token in the set t , \mathcal{C} represents the set of code, and \mathcal{T} represents the set of text.

3.2 Reinforcement Learning from Human Feedback

We begin with a brief introduction to Reinforcement Learning from Human Feedback (RLHF) [34], which primarily comprises three stages. The first stage is supervised fine-tuning on a LLM, denoted as \mathcal{M} , which is also a component of our framework and will be elaborated in Section 4.1. The second stage involves using the SFT model \mathcal{M}_1 to generate pairs of responses for a given prompt \mathcal{I} . These pairs have a preference order, as illustrated by p_i is preferred over p_j in Figure 3 (b). To predict these pairs, current works typically employ the Bradley-Terry (BT) model, which defines the preference probability as follows:

$$\mathcal{P}_{BT} = \frac{\exp(r_\phi(\mathcal{I}_1, p_i))}{\exp(r_\phi(\mathcal{I}_1, p_i)) + \exp(r_\phi(\mathcal{I}_1, p_j))} \quad (1)$$

Where r_ϕ is inherently a binary classification reward model, and \mathcal{I}_1 is a QA prompt containing the question q . The optimization objective of this stage is defined as a binary classification problem to train the reward model:

$$\mathcal{L}_{BT} = -\log \sigma(r_\phi(\mathcal{I}_1, p_i) - r_\phi(\mathcal{I}_1, p_j))$$

In the third stage, RLHF leverages the acquired r_ϕ to provide feedback to \mathcal{M}_1 and σ is the logistic function. Specifically, the optimization problem of RLHF is formulated the following :

$$\max_{\mathcal{M}_2} \mathbb{E}(r_\phi(\mathcal{I}_1, p) - \xi \log \frac{\mathcal{M}_2(p|\mathcal{I}_1)}{\mathcal{M}_1(p|\mathcal{I}_1)})$$

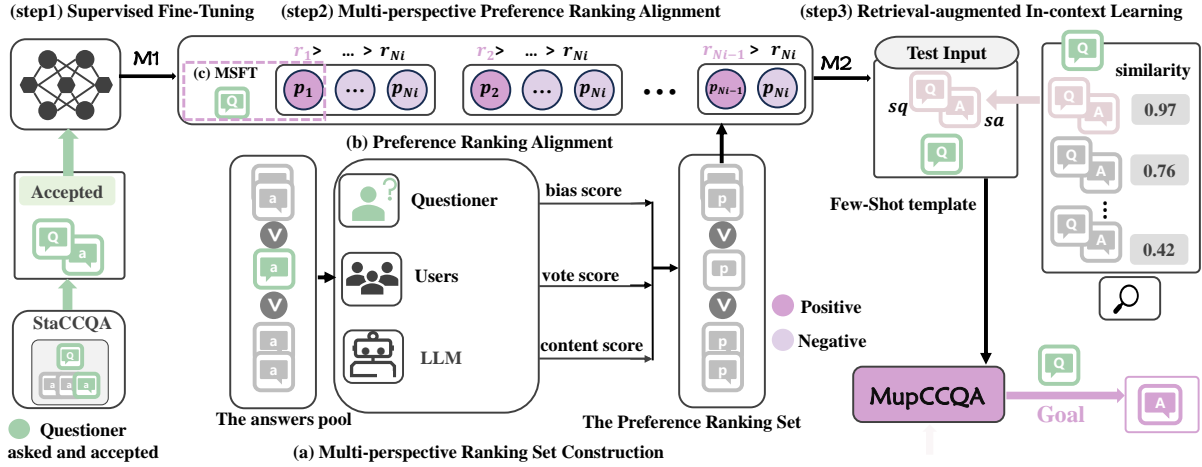


Figure 2: Overall architecture of the ALMupQA framework, including three stages: (step1) foundational Supervised Fine-Tuning (SFT) for acquiring programming-specific knowledge, (step2) Multi-perspective Preference Ranking Alignment (MPRA) for integrating diverse preferences, and (step3) Retrieval-augmented In-context Learning (RIL) to address the issue of outdated answers by retrieving the most similar post-solution pairs as prompts.

In this context, the role of ξ is to regulate the deviation from the baseline reference policy \mathcal{M}_1 , ensuring diversity in the generated outputs and preventing the production of high-reward yet non-sensical answers. It is worth noting that RLHF generates pairs of responses, which is not enough to questions with more than two answers. Therefore, we need to explore a new method to adapt.

4 METHODOLOGY

As shown in Figure 2, the ALMupQA framework encompasses three stages: (1) foundational Supervised Fine-Tuning to adapt to CCQA, (2) Multi-perspective Preference Ranking Alignment (MPRA) to integrate diverse preferences, and (3) Retrieval-augmented In-context Learning to address the issue of outdated answers.

4.1 Foundational Supervised Fine-Tuning

Foundational LLMs are typically trained on open-domain corpora. To adapt these universal LLMs to programming-specific code community corpora, we first employ a Supervised Fine-Tuning [34]. Specifically, we denote a^i with $a_c = 1$ as a_c^i and select pairs (q_i, a_c^i) from the dataset \mathcal{D} with votes v^i exceeding 100 to form the training and validation dataset. Then we optimize the LLM as follows:

$$\mathcal{L}_{SFT} = -\frac{1}{|a_c^i|} \sum_{j=1}^{|a_c^i|} \log \mathcal{P}_{\mathcal{M}}(a_c^{(i,j)} | \mathcal{I}, q_i, a_c^{(i,<j)}) \quad (2)$$

where $a_c^{(i,j)}$ is the j -th token of a_c^i , \mathcal{I} is the prompt template, and $\mathcal{P}_{\mathcal{M}}$ denotes the token probability predicted by the model \mathcal{M} . With this training objective, the QA data serves as the fundamental supervision information to fine-tune the model \mathcal{M} for the programming-specific QA scenario, resulting in a model denoted as \mathcal{M}_1 .

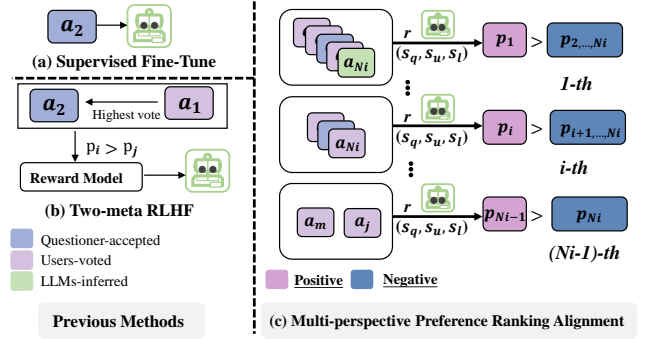


Figure 3: In CCQA, we compared previous human alignment methods with our approach. (a) SFT aligns only the answer accepted by the questioner a_2 , while (b) RLHF compares a_2 with the highest-voted users-preferred answer a_1 , sampling two-meta candidates $p_i > p_j$ from the entire ranking to train a reward model, and then relies on this reward model to fine-tune the base LLM. (c) Ours contrasts p_i with all members in the preference set $\{p_1, \dots, p_{N_i}\}$, based on the overall preference score r , which includes bias scores s_q , vote scores s_u , and content scores s_l .

4.2 Multi-perspective Preference Ranking Alignment

To align multi-perspective preferences from different users, we introduce the Multi-perspective Preference Ranking Alignment (MPRA) method. First, we propose three distinct metric scores from different aspects to build the ranking set. Then, these composite scores act as the reward value within the preference ranking alignment in a list-wise contrastive learning fashion.

4.2.1 Multi-perspective Ranking Set Construction. First, as the answer chosen by the questioner may not be favored by other users[18], we introduced questioner-perspective bias score to assess

the discrepancy between the accepted answer and the most users-preferred answer, denoted as s_q .

$$s_q = \frac{(v - v_a) - v_m}{v_\sigma} \quad (3)$$

Here, $v \in \{v_1, \dots, v_{N_i}\}$ represents the votes for each answer in question q . v_a denotes the votes for the answer accepted by the questioner, and v_m and v_σ are the mean and standard deviation of $\{v_1, \dots, v_{N_i}\}$, respectively.

Second, since high-quality text is usually accompanied by a high number of votes [15], interaction data from open communities not only reflect the preferences of community users but also serve as a dual filtering mechanism for the quality of the answer content. Unlike specific answers preferred by the questioner, the number of votes generally reflects the universality and generality of the answer, indicating its applicability and transferability to other contexts. This aspect is particularly significant in the field of software engineering, where similar issues may arise in different environments. The ability to generalize and apply solutions across diverse scenarios is a crucial capability in this field, underscoring the importance of community-driven feedback and the collective wisdom it represents. Therefore, to comprehensively consider the users' collective preferences and engagement with the answers, we introduce a users-perspective voting score, denoted as s_u , mathematically expressed as follows:

$$s_u = \frac{v - \min(V)}{\max(V) - \min(V)} \quad (4)$$

Here, v belongs to V , and $V = \{v_1, \dots, v_{N_i}\}$ is the set of votes for any given answer pool $\{a_1, a_2, \dots, a_{N_i}\}$. $\min(V)$ and $\max(V)$ represent the minimum and maximum values within V , respectively. This normalization ensures that the number of votes is adjusted to a common scale, facilitating fair comparison across different answers.

Third, since high semantic accuracy is a fundamental prerequisite for answering questions, we introduce a content score from the perspective of LLMs, denoted as s_l . This score aims to leverage LLMs, which excel at handling the nuanced semantic relationships between text and code, to evaluate the quality of text c in answer a . The LLMs employed can be general-purpose or specifically tailored for the code domain, typically possessing excellent comprehension and reasoning capabilities. In our study, we selected an easily accessible LLM, denoted as CM , to measure the semantic logical value of each question-answer pair (q_i, a_i) , where a_i is an element from the set $\{a_1^i, a_2^i, \dots, a_{N_i}^i\}$. The content score s_l is calculated by multiplying the probabilities of each token generated by CM , as detailed in Eq. (5). \mathcal{I}_1 represents a prompt template that integrates the question-answer pair (q_i, a_i) .

$$s_l = \prod_{t \in a_i} \sigma(CM(\mathcal{I}_1[q_i, t])) \quad (5)$$

In Eq. (5), the product is taken over all tokens t in the answer a_i , and σ is the logistic function. Ultimately, to comprehensively evaluate the pool of answers and construct a preference sequence, we introduced an overall preference score r , which consolidates various perspectives into a unified measure. Subsequently, we ranked the preference set based on the magnitude of r , with the highest-scoring answer a becoming p_1 .

$$r = \alpha_1 \cdot s_q + \alpha_2 \cdot s_u + \alpha_3 \cdot s_c \quad (6)$$

The weights α_1 , α_2 , and α_3 reflect the relative importance of each component within the multi-perspective preference set.

4.2.2 Preference Ranking Alignment. Assuming that each answer p_i in the preference ranking set $\{p_1, \dots, p_{N_i}\}$ has been trained to resemble human responses, the RLHF method based on the Bradley-Terry model for pairwise comparisons [34] may be insufficient, such as $p_i > p_j$ shown in Figure 3 (b). To facilitate the comparison of multiple responses, we extended the Bradley-Terry model through Multi-perspective Preference Ranking Alignment (MPRA), inspired by Proximal Policy Optimization (PPO) [42] and Preference Ranking Optimization [43]. MPRA shifts the focus from a reward model-centric approach to directly adjusting the probability ranking of N_i answers generated by LLMs to align with the overall preference score r . Here, N_i denotes the size of the answer pool $\{a_1^i, a_2^i, \dots, a_{N_i}^i\}$ for question q_i , which varies with the question q . The comprehensive process of MPRA is illustrated in Figure 3 (c). The initial preference ranking set $\{p_1, \dots, p_{N_i}\}$ is given by $p_1 > p_2 > \dots > p_{N_i}$, which can be divided into p_1 and $\{p_2, \dots, p_{N_i}\}$. The function parameterized of LLM \mathcal{M}_2 in MPRA is defined as $\mathcal{M}_2(\mathcal{I}_1, p_k)$. The extended Bradley-Terry objective is defined as follows:

$$\mathcal{P}(p_{1,2:N_i}|\mathcal{I}_1) = \frac{\exp(\mathcal{M}_2(\mathcal{I}_1, p_1))}{\sum_{k=1}^{N_i} \exp(\mathcal{M}_2(\mathcal{I}_1, p_k))} \quad (7)$$

As the i -th iteration unfolds, MPRA systematically eliminates the top i answers with higher r scores. This process is repeated until all answers are excluded. This iterative refinement continues sequentially until the entire set of potential solutions is exhausted, with the final target probability evolving into:

$$\begin{aligned} \mathcal{P}(p_{1,\dots,N_i}|\mathcal{I}_1) &= \prod_{i=1}^{N_i-1} \mathcal{P}(p_{i+1:N_i}|\mathcal{I}_1) \\ &= \prod_{i=1}^{N_i-1} \frac{\exp(\mathcal{M}_2(\mathcal{I}_1, p_i))}{\sum_{k=i}^{N_i} \exp(\mathcal{M}_2(\mathcal{I}_1, p_k))} \end{aligned} \quad (8)$$

This optimization objective is intricately aligned with the ultimate aim of human alignment, which is to select the desired response from the expansive response space of LLMs [39]. Essentially, when $N_i \rightarrow \infty$, Eq. (8) is capable of exhaustively exploring all potential responses generated by the LLM; when $N_i = 2$, MPRA transforms into the Bradley-Terry model as described in Eq. (1), yet still effectively optimizes the LLM. Unlike methods that compel the LLM to approximate a reward model, MPRA directly trains the LLM by optimizing the probability ranking of the preference set through list-wise contrastive learning objectives. Its optimization goal, represented by Eq. (9), aims to realize Eq. (8). This process iteratively designates the most favored response with the highest r score as positive, perfectly aligning with human preferences, while the remaining responses are treated as negative. MPRA not only needs to generate the most preferred responses but also to enhance text fluency and code structure. Consequently, MPRA incorporates the new SFT stage, as shown in Figure 3 step2 (c), with its loss identical to that of the foundational SFT phase in Eq. (2) described in Section

Table 1: Statistics on the size of the answers pool for each question.

Count Interval	[0,2)	[2,5)	[5,10)	[10,15)	[15,20)	[20,25)	[25,30]	Total
Count	325,780	245,793	21,986	2,057	572	203	222	596,613
Percentage(%)	54.60	41.20	3.68	0.35	0.10	0.03	0.04	100

4.1, denoted as \mathcal{L}_{MSFT} .

$$\mathcal{L}_{MPRA} = - \sum_{i=1}^{N_i-1} \log \frac{\exp(\mathcal{M}_2(\mathcal{I}_1, p_i))}{\sum_{k=i}^{N_i} \exp(\mathcal{M}_2(\mathcal{I}_1, p_k))} \quad (9)$$

The crucial difference is that during the foundational SFT phase, the alignment target is the questioner-accepted answer, whereas the LLM's alignment target now is the answer with the highest overall preference score r . Ultimately, the overall optimization objective can be summarized as follows:

$$\mathcal{L}(p_{1,\dots,N_i}|\mathcal{I}_1) = \mathcal{L}_{MPRA} + \alpha \mathcal{L}_{MSFT} \quad (10)$$

where \mathcal{L}_{MSFT} is the NLL loss for the top candidate, and α is a hyperparameter used to balance text quality and human preference.

4.3 Retrieval-augmented In-context Learning

As the accepted answer may become outdated with the rapid occurrence of API updates in the field of programming, for instance, in Figure 1, the "urllib" API in answer a_2 is applicable to Python 2 but deprecated in Python 3, we introduced Retrieval-augmented In-context Learning (RIL) to address the issue of outdated answers and to align with the user's preference for utilizing new API trends. By retrieving analogous questions from the question bank and employing them as few-shot examples, we enhance the efficacy of the generated responses. We utilize a dense retriever (\mathcal{R}_D), which excels at handling the transition from natural language to code generation, having been trained to extract documents from a comprehensive pool that includes a vast repository of code libraries, APIs, and functions. Due to the significant influence of the ordering of few-shot examples on the model's predictions [56], even though In-context Learning can still perform well when the orders or labels of prompts are exchanged [29], we select the most similar question-answer (sq, sa) pair from the question-answers bank \mathcal{D} to serve as the few-shot example in prompt \mathcal{I}_2 . The ultimate objective is formulated as follows:

$$\mathcal{P}(A_t) = \prod_{i=1}^T P_{\mathcal{M}_2}(A|\mathcal{I}_2, Q, (sq, sa), A_{<t}) \quad (11)$$

Here, \mathcal{M}_2 represents the LLMs following the MPRA stage, and Q is the question to be resolved

5 EXPERIMENTS

We examine the performance of our proposed ALMupQA for the CCQA task in this Section. In particular, we focus on answering the following research questions:

- **RQ1:** Can ALMupQA surpass the baseline methods in accuracy metrics within the CCQA task?
- **RQ2:** Can each component of ALMupQA make a contribution?

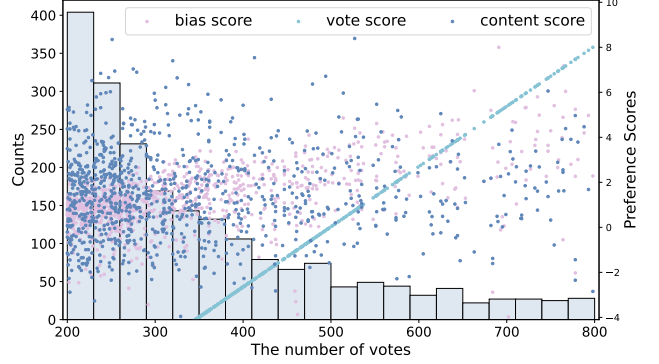


Figure 4: The statistic of the number of votes for a question and the mapping relationship among the bias score s_q , vote scores s_u and content scores s_l .

- **RQ3:** Can ALMupQA align with human preferences and explore the relationship between preference scores and accuracy metrics?
- **RQ4:** Can ALMupQA effectively generate user-centric responses that align with human preferences in the coding community?

5.1 StaCCQA Dataset Construction

Due to the lack of high-quality, authentic multi-answer preference datasets in code communities, there is an urgent need to construct a new dataset. To address this gap, we have turned to StackExchange, a platform whose forums are accompanied by rich question-answering metadata information. A publicly available dump of user-contributed content from Stack Overflow, provided by StackExchange under a cc-by-sa 4.0 license, has formed the foundation for the creation of our dataset StaCCQA.

The initial StaCCQA dataset was in XML format, comprising 757,702 (q_i, a_i) pairs, mainly featuring `<python>` tags, with 600,176 pairs containing code blocks. To obtain the latest answers, we systematically gathered all answers for each question q on Stack Overflow up to August 2023, resulting in a dataset totaling 596,613 pairs. Detailed statistics are presented in Table 1. We then proceeded with the following preprocessing steps and the resulting dataset \mathcal{D} contains 270,716 ($q_i, \{a_1, \dots, a_{N_i}\}$) pairs.

- To ensure that submission messages are descriptive, we removed pairs with titles that are shorter than three tokens (including three tokens). This decision follows CCT5 [21], which stipulates that code comments should contain more than three tokens.
- Pairs where the answer did not contain code block content were eliminated to ensure that ALMupQA's reference content includes both text and code, due to the nature of CCQA.
- Pairs with an answer pool size smaller than 2 were discarded.

Table 2: The zero-shot experimental results on the StaCCQA dataset. Open-source code baselines are above ALMupQA and closed-source baselines are below ALMupQA. The best result in each column is marked in bold. The second-best result in each column is underlined.

Model	Model size	BLEU ₄	ROUGE ₂	CHRF	BERTScore	CodeBERTScore-PR	CodeBERTScore-F
Godegen-mono	16B	6.72	9.24	32.94	77.53	54.42	50.20
GPT-NeoX	20B	8.40	11.26	33.46	78.06	53.79	49.87
StarCoder	15B	9.32	11.92	30.75	77.57	53.36	52.21
WizardCoder-python	13B	12.97	15.88	37.54	<u>79.34</u>	52.37	51.89
CodeT5+	-	3.86	5.16	25.58	75.96	53.48	46.19
Code Llama2	7B	11.86	16.32	35.08	70.10	46.46	47.05
Code Llama2	13B	13.56	18.32	38.68	78.13	51.79	52.91
ALMupQA(Ours)	7B	22.86	25.48	40.58	84.14	65.12	63.53
PaLM	-	13.15	18.68	<u>39.89</u>	77.89	52.81	51.98
ChatGLM	-	13.91	18.71	38.21	78.28	53.29	<u>53.77</u>
GPT-3.5	-	<u>15.29</u>	<u>19.24</u>	39.10	78.90	52.10	52.95
Claude2	-	14.69	19.12	38.78	78.45	51.58	52.63
GPT-4	-	13.04	17.74	35.43	78.23	<u>57.84</u>	46.82

- All HTML tags were cleaned and replaced with “[HTML]”, particularly `` and `<img=...>` tags, to ensure the model is not influenced by such exceedingly complex and meaningless content. This decision follows existing research that constructed datasets related to submissions [17, 26].

5.2 Multi-perspective Phenomenon Analysis

To validate the necessity of multi-perspective preference modeling, we randomly extracted approximately 2,000 entries from the constructed dataset StaCCQA. We calculated the bias score, vote score and content score for each answer a . These scores were then mapped onto a two-dimensional coordinate system, as illustrated in Figure 4. The horizontal axis represents the number of votes, the left vertical axis indicates the number of answers, and the right vertical axis denotes the preference score. Naturally, the vote score s_u shows a positive correlation with the number of votes. For question audience bias s_q , if the votes v for a user-favored answer are close to the votes for the answer chosen by the questioner, then s_q is near the X-axis. In Figure 4, most s_q values are distant from the X-axis, highlighting a significant divergence between user preferences and the questioner’s choices within the coding community. Analyzing the distribution of content scores s_l : if some answers to question q are semantically similar, content scores should cluster. The lack of clustering indicates that no single answer comprehensively covers all semantic aspects of the question.

In summary, the observed distinct distributions of content scores, vote scores, and bias scores underscore the existence of diverse preferences from different perspectives. This finding validates the necessity of accurately capturing and presenting user preferences in CCQA, necessitating the adoption of multi-perspective modeling approaches.

5.3 Experiment Settings

5.3.1 Baselines. The classification of LLMs can be determined by the openness of the technology and whether the code is available

for research or commercial use. Based on the unique characteristics of CCQA, we selected two types of baseline models. The first category consists of general-purpose, closed-source LLMs designed for text generation, including GPT-3.5-turbo, GPT-4 [33], PaLM [3], ChatGLM [53], and Claude2 [1]. The second category comprises open-source code LLMs that excel in program synthesis, such as StarCoder [20], WizardCoder-Python-13B [27], GPT-NeoX [5], CodeGen-mono-16B [?], and Code Llama 2 [41].

5.3.2 Evaluation Metrics. To comprehensively evaluate the experimental results, we employed various evaluation metrics from four perspectives: traditional text generation metrics (BLEU [35], Rouge [22], and CHRF [37]), model-based metrics (BERTScore [55]), code-related metrics (CodeBERTScore [59]), and preference metrics based on GPT-4 evaluations. Additionally, considering the similarity between Precision and Recall in CodeBERTScore, we unified these metrics as “CodeBERTScore-PR” (abbreviated as CB-PR). Similarly, the F1 and F-measure in CodeBERTScore were merged into “CodeBERTScore-F” (abbreviated as CB-F).

5.3.3 Implementation Details. In this study, we selected Code Llama-Instruct-7B [41] as the base model \mathcal{M} , which belongs to a series of large code language models based on Llama 2 [47]. Code Llama has demonstrated state-of-the-art performance across various code benchmarks. We utilized the ALMupQA framework for three-stage fine-tuning. During the SFT phase, we specified the following hyperparameters: *epoch*, *temperature*, *top_p*, *max_seq_len*, and *max_batch_size*, set to 4, 0.2, 0.95, 2048, and 28, respectively. We retained the remaining hyperparameter settings of Llama, which can be found at the following link³. In the MPRA phase, we selected an accessible LLM⁴ as $\mathcal{C}\mathcal{M}$. The hyperparameters were set as follows: *learning_rate*, *gradient_accumulation_steps*, *epochs*, *top_p*, *max_gen_len*, *temperature* and *max_batch_size*, set to 1e-4, 9, 4, 1.0, 0.95, 512, and 4, respectively. Given the excellent performance

³<https://github.com/facebookresearch/llama>

⁴<https://huggingface.co/OpenAssistant/oasst-rm-2-pythia-6.9b-epoch-1>

Table 3: The one-shot experimental results on the StaCCQA dataset. The best result in each column is marked in bold. The second-best result in each column is underlined.

Model	Model size	BLEU ₄	ROUGE ₂	CHRF	BERTScore	CodeBERTScore-PR	CodeBERTScore-F
Godegen-mono	16B	8.06	11.01	33.32	78.28	<u>54.67</u>	50.20
GPT-NeoX	20B	8.95	11.30	26.84	76.68	<u>52.64</u>	51.93
StarCoder	15B	10.59	14.40	33.71	78.20	53.43	52.96
WizardCoder-python	13B	13.35	15.97	37.56	<u>79.42</u>	52.70	52.11
CodeT5+	-	4.40	5.60	25.96	75.91	52.23	47.52
ALMupQA (Ours)	7B	22.86	25.48	40.58	84.14	65.12	63.53
PaLM	-	12.77	18.97	34.00	77.90	52.35	52.25
ChatGLM	-	13.47	17.50	37.06	78.20	53.51	<u>53.53</u>
GPT-3.5	-	14.50	18.43	<u>39.17</u>	78.92	52.64	52.52
Claude2	-	14.10	18.24	38.25	78.46	51.38	52.36
GPT-4	-	<u>14.73</u>	<u>18.87</u>	36.68	78.78	52.44	52.56

Table 4: The ablation study results. We evaluate various stripped-down versions of our model to compare the performance gains brought by different components. The full names of these abbreviations are as follows: SFT (Foundational Supervised Fine-Tuning); MPRA (Multi-perspective Preference Ranking Alignment); s_p (bias score); s_l (content scores); s_u (vote scores); and RIL (Retrieval-augmented In-context Learning). The components in bold have the most significant impact on performance.

Model	BLEU ₄	ROUGE ₂	CHRF	BS	CB-PR	CB-F
ALMupQA	22.86	25.48	40.58	84.14	65.12	63.53
w/o SFT	21.30	23.62	37.88	76.15	59.76	57.73
w/o MPRA	14.62	20.50	39.18	80.41	55.72	53.85
w/o s_q	22.16	25.18	39.38	83.34	64.72	62.83
w/o s_u	21.01	23.23	40.48	78.06	53.79	49.87
w/o s_l	21.56	24.58	38.78	82.54	64.22	62.13
w/o RIL	21.66	23.16	39.18	81.82	61.22	62.53

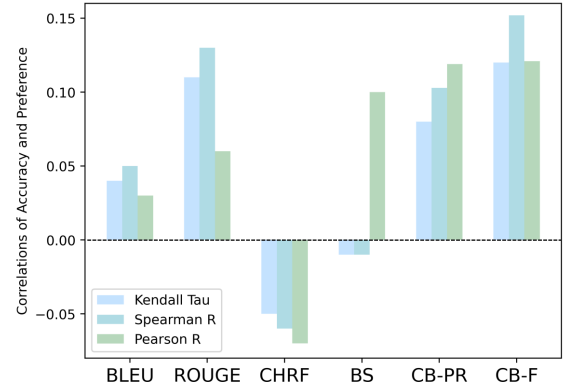
of this retrieval-generation approach in understanding diverse texts and code, during the Retrieval-augmented In-context Learning (RIL) phase, we chose the DocPrompting method based on SimCSE [60] as our retriever \mathcal{R}_D . This retriever \mathcal{R}_D includes 35,763 functions from all Python libraries on DevDocs⁵, encompassing the Python standard library and widely-used packages such as NumPy and Pandas, and was pre-trained on the re-split CoNaLa [50] benchmark.

5.4 RQ1: Main Results

In this experiment, we used text and code metrics to evaluate the performance of general baselines and code baselines on the StaCCQA dataset, and compared the results. The Table 2 below summarizes the experimental outcomes, presenting the following observations:

First, our ALMupQA significantly outperformed other baseline models across all metrics. Specifically, ALMupQA exceeded the next

⁵<https://devdocs.io>

**Figure 5: The consistency correlations between accuracy-based metrics (BLEU, ROUGE, CHRF, BERTScore, CB-PR, and CB-F) and preference-based metrics (GPT-4 evaluation scores). A positive correlation indicates that accuracy metrics improve as preference scores increase.**

best results in every metric, surpassing general LLMs in text generation and code LLMs in code generation. For instance, compared to the second-best GPT-3.5, it scored 5.2% higher on BERTScore, nearly 7% higher on CB-PR, and about 10% higher on CB-F than ChatGLM. This indicates that ALMupQA is a robust and versatile model, excelling in the quality, grammar, and semantics of generated answers. Notably, general large language models might have superior code generation capabilities in the CCQA task compared to code LLMs. Second, ALMupQA showed significant improvements compared to the baseline models, even doubling the BLEU score of the benchmark model (22.86% vs. 11.86%). Other n-gram-based metrics (ROUGE and CHRF) and semantic-grammar-based metrics also saw substantial enhancements, indicating that ALMupQA is an effective framework for applying large models to CCQA problems.

Due to the presence of similar question-answer pairs as few-shot examples in the Retrieval-Enhanced In-Context Learning (RIL) of ALMupQA, we also applied RIL to the remaining baselines in a new experiment, and the results are presented in Table 3. This

Question	The response of Code Llama	The response of MupCCQA
(1) Reloading submodules in IPython?	1.First, import the <code>autoreload</code> module in your IPython session: <code>import autoreload</code> . 2. Next, use the <code>autoreload.enable()</code> GPT-4 Score: 8	One solution: in your IPython console: <code>%autoreload</code> . This will enable autoreload for all modules, including submodules Other solution: if you want to reload only the, you can use the following command: <code>%autoreload main_mod.sub_mod</code> . GPT-4 Score: 9
(2) fatal error: Python.h: No such file or directory.	1.python -c import distutils.sysconfig; print(distutils.sysconfig.get_python_inc()). 2.export PYTHON_INCLUDE_DIR=/path/to/Python.h Replace /path/to/Python.h GPT-4 Score: 9	you can use the following command to build the shared library: <code>gcc -Wall -I/usr/include/python3.8 utilsmodule.c -o Util</code> . GPT-4 Score: 10
(3) Can I force pip to reinstall the current version?	Yes, you can force pip to reinstall the current version of a package by using the <code>--force-reinstall</code> option. Here's an example: <code>pip install --force-reinstall <package-name></code> This will reinstall the package even if it is already up-to-date. Alternatively, you can use the <code>--upgrade</code> option to upgrade the package to the latest version available. <code>pip install --upgrade <package-name></code> GPT-4 Score: 8	Yes, you can force pip to reinstall the current version of a package by using the <code>--upgrade</code> flag. <code>pip install --upgrade package_name</code> <code>pip install --upgrade-strategy=latest package_name</code> Note that the <code>--upgrade</code> and <code>--upgrade-strategy</code> flags are only available in pip version 19.0 and later. If you are using an earlier version of pip, you may need to use a different approach to force pip to reinstall the current version of a package. GPT-4 Score: 10
(4) How do I get a Cron-like scheduler in Python?	The <code>croniter</code> library can be used with the <code>sched</code> module from the standard library to create a Cron-like scheduler <code>import sched...import time...</code> <code>from croniter import croniter</code> . <code>scheduler = sched.scheduler(time.time, time.sleep)</code> GPT-4 Score: 7	The <code>APScheduler</code> library is a more modern and feature-rich option for scheduling tasks. <code>from apscheduler.schedulers.blocking import BlockingScheduler</code> <code>from apscheduler.triggers.cron import CronTrigger</code> . <code>scheduler = BlockingScheduler()</code> <code>scheduler.add_job(scheduled_task, CronTrigger.from_crontab(""))</code> GPT-4 Score: 9

Figure 6: A case study on the performance of Code Llama and ALMupQA in Code Community Question Answering. The black text signifies natural language. The blue text represents programming block. The red text indicates outdated content. The green indicates the preference score based on GPT-4 evaluation. The higher the score, the greater the alignment with user preferences. The scores ranges from [1,10].

aims to ensure fairness in comparing other zero-shot baselines with ALMupQA. In Table 3, each baseline shows improvement across various metrics compared to the zero-shot results in Table 2, with GPT-4 exhibiting significant enhancement in long-text performance, becoming the second-best baseline. However, they still cannot match our ALMupQA. Specifically, the BLEU score of ALMupQA remains far higher than the second-best GPT-4; the BERTScore of ALMupQA also surpasses the WizardCoder-python by nearly 4.7%. In terms of CB-PR and CB-F, ALMupQA exceeds the second-best baseline by nearly 10%.

5.5 RQ2: Ablation Study

To validate the enhancement in performance brought by MuCCQA’s preference scores across three stages and three different perspectives scores, we conducted ablation experiments. The results are shown in Table 4. Upon removing SFT, all metrics experienced a decline, with BERTScore showing the most significant drop (from 84.14% to 76.15%), underscoring the importance of this stage for understanding the semantics of programming domain knowledge. Eliminating MPRA resulted in notable decreases in model performance on metrics focused on complex phrase matching and code semantics, specifically CodeBERTScore, which dropped by 8.8% and 9.4%, respectively. This suggests that an unadapted LLM fails to account for the diversity of preferences within the coding community. If RIL is excluded, MuCCQA’s performance on semantic-focused metrics (BERTScore and CodeBERTScore) significantly declines, highlighting the critical role of similar examples in understanding problem semantics. Furthermore, all three preference scores in MuCCQA, including bias scores s_q , vote scores s_v , and content scores s_c , contribute to its performance. In summary, each component of our method plays a unique role, collectively enhancing overall performance.

5.6 RQ3: GPT-4 Evaluation

Given that GPT-4 [33] has demonstrated significant ability in effectively evaluating question-and-answer pairs and aligning with human preferences [48, 57], we utilize it to assess the preferences for responses generated by the open-source Code Llama, the closed-source GPT-3.5, and our proposed ALMupQA. To evaluate whether the responses align with human preferences, we designed evaluation criteria encompassing four dimensions: the usefulness, relevance, accuracy, and level of detail of each answer. Each solution is rated on a scale from 1 to 10, with comprehensive explanations required for each score. We need to provide GPT-4 with the question title and specific description, the standard answer, and the responses generated by the LLMs to be evaluated. The GPT-4 evaluation results indicate that ALMupQA is capable of generating responses that are highly aligned with human preferences, with an average score surpassing that of GPT-3.5 (7.54 to 7.51) and Code Llama (7.54 to 7.43).

To explore the consistency between accuracy-based metrics (including BLEU, ROUGE, CHRF, BERTScore, CB-P, and CB-F) and preference-based metrics (GPT-4 evaluated preference scores), we employed three key statistical correlation coefficients: Kendall’s Tau τ [19], Spearman’s R γ [38], and Pearson’s R ρ [7], as depicted in Figure 5. The Figure 5 primarily illustrates three points: First, the three correlation measures, τ , γ , and ρ , maintain a high degree of sign consistency. Second, text-based metrics (BLEU and ROUGE), semantic-based metrics (BERTScore), and code-based metrics (CB-P and CB-F) all exhibit a positive correlation with preference-based metrics, whereas CHRF shows a negative correlation in both τ and γ . Lastly, the correlation between code-based metrics and preference is the most pronounced, which aptly reflects the characteristics of our code community question-answering tasks. Overall, accuracy and preference are not contradictory, providing a valuable reference for evaluating CCQA tasks.

5.7 RQ4: Case Study

To validate the excellence of our ALMupQA, we selected four random questions for comparison, as shown in Figure 6. To analyze whether ALMupQA effectively generates user-centric responses, we take randomly the third question, "How to force pip to reinstall the current version?" as an example. ALMupQA scored 10, while Code Llama scored 8. Although both responses covered the core points and clearly explained how to use the "--force-reinstall" flag, ALMupQA excelled in the following aspects: First, in detail: ALMupQA provided a more thorough explanation, covering not only the "--upgrade" flag but also the "--upgrade-strategy" flag. This additional information helped users understand and manage package upgrades better. Second, in accuracy and relevance: ALMupQA accurately explained the usage of the "--upgrade" and "--upgrade-strategy" flags, making the response more informative and helpful for managing package versions and upgrades. Third, in user-friendliness: ALMupQA's response was well-structured and user-friendly, with clear instructions and examples that made it easier for users to follow and apply the information.

The fourth question in the Figure 6 aims to highlight the presence of outdated APIs in some responses generated by LLMs. Specifically, Code Llama employed the "sched" module, which is part of the Python standard library, but is no longer as commonly used. In contrast, the response of ALMupQA utilized a more contemporary library "APScheduler", a popular and feature-rich option for scheduling tasks.

6 CONCLUSION

In this paper, aiming to explore the application of LLMs with RLHF for human preference alignment in programming-domain Code Community Question Answering (CCQA), we propose ALMupQA. First, we introduce a multi-perspective preference ranking alignment framework to accommodate diverse user preferences. Second, to address users' inclination towards using new APIs, we implement a retrieval-augmented in-context learning (RIL) module to mitigate the issue of outdated information. We conducted extensive experiments to validate the accuracy of our ALMupQA-responses answers on our crafted dataset StaCCQA, demonstrating an improvement of nearly 11% in the BLEU compared to the foundation model, with increases of 20% and 17.5% in BERTScore and CodeBERTScore, respectively. Additionally, GPT-4 evaluations confirmed the increase in accuracy-based metrics, with preference scores also showing improvement, indicating the effectiveness of our approach in aligning preferences in the CCQA task. Overall, we emphasize a novel perspective that considers the diversity of users when aligning with human preferences.

REFERENCES

- [1] Anthropic. model card and evaluations for claude models, 2023. <https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf>.
- [2] Leandro Amancio, Carina F Dorneles, and Daniel H Dalip. Recency and quality-based ranking question in cqas: A stack overflow case study. *Information Processing & Management*, 58(4):102552, 2021.
- [3] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.
- [4] Muhammad Asaduzzaman, Ahmed Shah Mashiyat, Chanchal K Roy, and Kevin A Schneider. Answering questions about unanswered questions of stack overflow. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 97–100. IEEE, 2013.
- [5] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06374*, 2022.
- [6] Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. Gpt-neo: Large scale autoregressive language modeling with mesh-tensorflow. *If you use this software, please cite it using these metadata*, 58, 2021.
- [7] Auguste Bravais. *Analyse mathématique sur les probabilités des erreurs de situation d'un point*. Impr. Royale, 1844.
- [8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [9] Qin Chen, Qinmin Hu, Jimmy Xiangji Huang, Liang He, and Weijie An. Enhancing recurrent neural networks with positional attention for question answering. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 993–996, 2017.
- [10] Daniel Hasan Dalip, Marcos André Gonçalves, Marco Cristo, and Pavel Calado. Exploiting user feedback to learn to rank answers in q&a forums: a case study with stack overflow. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 543–552, 2013.
- [11] Jiangnan Du, Jun Chen, Suhong Wang, Jianfeng Li, and Zhifeng Xiao. Towards a two-stage method for answer selection and summarization in buddhism community question answering. In *Artificial Intelligence: First CAAI International Conference, CICA 2021, Hangzhou, China, June 5–6, 2021, Proceedings, Part II 1*, pages 251–260. Springer, 2021.
- [12] Lun Du, Xiaozhou Shi, Yanlin Wang, Ensheng Shi, Shi Han, and Dongmei Zhang. Is a single model enough? mucos: A multi-model ensemble learning approach for semantic code search. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2994–2998, 2021.
- [13] Zhipeng Gao, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. Generating question titles for stack overflow from mined code snippets. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(4):1–37, 2020.
- [14] Alexandru Lucian Ginsca and Adrian Popescu. User profiling for answer quality assessment in q&a communities. In *Proceedings of the 2013 workshop on Data-driven user behavioral modelling and mining from social media*, pages 25–28, 2013.
- [15] George Gkotsis, Karen Stepanyan, Carlos Pedrinaci, John Domingue, and Maria Liakata. It's all in the content: state of the art best answer prediction based on discretisation of shallow linguistic features. In *Proceedings of the 2014 ACM conference on Web science*, pages 202–210, 2014.
- [16] Muntasir Hoq, Sushanth Reddy Chilla, Melika Ahmadi Ranjbar, Peter Brusilovsky, and Bitu Akram. Sann: Programming code representation using attention neural network with optimized subtree extraction. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 783–792, 2023.
- [17] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.
- [18] Pranav Kasela, Gabriella Pasi, and Raffaele Perego. Se-pqa: Personalized community question answering. *arXiv preprint arXiv:2306.16261*, 2023.
- [19] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.
- [20] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.
- [21] Bo Lin, Shangwen Wang, Zhongxin Liu, Yepang Liu, Xin Xia, and Xiaoguang Mao. Cct5: A code-change-oriented pre-trained model. *arXiv preprint arXiv:2305.10785*, 2023.
- [22] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.
- [23] Zizheng Lin, Haowen Ke, Ngo-Yin Wong, Jiaxin Bai, Yangqiu Song, Huan Zhao, and Junpeng Ye. Multi-relational graph based heterogeneous multi-task learning in community question answering. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1038–1047, 2021.
- [24] Chenxiao Liu and Xiaojun Wan. Codeqa: A question answering dataset for source code comprehension. *arXiv preprint arXiv:2109.08365*, 2021.
- [25] Qiaoling Liu, Eugene Agichtein, Gideon Dror, Evgeniy Gabrilovich, Yoelle Maarek, Dan Pelleg, and Idan Szepietor. Predicting web searcher satisfaction with existing community-based answers. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 415–424, 2011.
- [26] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

- [27] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.
- [28] Macedo Maia, Siegfried Handschuh, and Markus Endres. A tag-based transformer community question answering learning-to-rank model in the home improvement domain. In *International Conference on Database and Expert Systems Applications*, pages 127–138. Springer, 2021.
- [29] Sewon Min, Xinxin Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.
- [30] Inderjeet Nair, Shwetha Somasundaram, Apoorv Saxena, and Koustava Goswami. Drilling down into the discourse structure with llms for long document question answering. *arXiv preprint arXiv:2311.13565*, 2023.
- [31] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. What makes a good code example?: A study of programming q&a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34. IEEE, 2012.
- [32] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.
- [33] OpenAI. Gpt-4 technical report, 2023.
- [34] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.
- [35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.
- [36] Hung Phan and Ali Jannesari. Evaluating and optimizing the effectiveness of neural machine translation in supporting code retrieval models: A study on the cat benchmark. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 2055–2064, 2023.
- [37] Maja Popović. chrF: character n-gram f-score for automatic mt evaluation. In *Proceedings of the tenth workshop on statistical machine translation*, pages 392–395, 2015.
- [38] A Franklin. *Introduction to the Theory of Statistics*. 1974.
- [39] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.
- [40] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering*, 47(3):560–581, 2019.
- [41] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.
- [42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [43] Feifan Song, Bowen Yu, Minghao Li, Haiyang Yu, Fei Huang, Yongbin Li, and Houfeng Wang. Preference ranking optimization for human alignment. *arXiv preprint arXiv:2306.17492*, 2023.
- [44] Avikalp Srivastava, Hsin-Wen Liu, and Sumio Fujita. Adapting visual question answering models for enhancing multimodal community q&a platforms. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1421–1430, 2019.
- [45] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.
- [46] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [47] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shriti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [48] Peiyi Wang, Lei Li, Liang Chen, Dawei Zhu, Binghui Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*, 2023.
- [49] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.
- [50] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th international conference on mining software repositories*, pages 476–486, 2018.
- [51] Pengcheng Yin, Nan Duan, Ben Kao, Junwei Bao, and Ming Zhou. Answering questions with complex semantic constraints on open knowledge bases. In *Proceedings of the 24th ACM international conference on information and knowledge management*, pages 1301–1310, 2015.
- [52] Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, Songfang Huang, and Fei Huang. Rrhf: Rank responses to align language models with human feedback without tears. *arXiv preprint arXiv:2304.05302*, 2023.
- [53] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.
- [54] Haoxiang Zhang, Shaowei Wang, Tse-Hsun Chen, Ying Zou, and Ahmed E Hassan. An empirical study of obsolete answers on stack overflow. *IEEE Transactions on Software Engineering*, 47(4):850–862, 2019.
- [55] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.
- [56] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*, pages 12697–12706. PMLR, 2021.
- [57] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanhào Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.
- [58] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*, 2023.
- [59] Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. Codebertscore: Evaluating code generation with pretrained models of code. *arXiv preprint arXiv:2302.05527*, 2023.
- [60] Shuyan Zhou, Uri Alon, Frank F Xu, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. In *The Eleventh International Conference on Learning Representations*, 2022.
- [61] Xiaoqiang Zhou, Baotian Hu, Qingcai Chen, and Xiaolong Wang. Recurrent convolutional neural network for answer selection in community question answering. *Neurocomputing*, 274:8–18, 2018.