

Multi-perspective Preference Alignment of LLMs for Programming-Community Question Answering

Hongyu Yang¹, Jiahui Hou^{1*}, Liyang He¹, Rui Li¹,

¹University of Science and Technology of China,

jhou @ustc.edu.cn

hongyuyang, heliyang, ruili2000 @mail.ustc.edu.cn

Abstract

Programming-Community Question Answering (PCQA) aims to address challenges by generating functional code and guiding descriptions. It involves multiple candidates, each with varying user preferences. Additionally, some answers may contain outdated APIs, which further complicates the task of generating responses that meet user expectations. Recently, Reinforcement Learning from Human Feedback (RLHF) has proven effective in controlling the behavior of large language models (LLMs) to produce human-like responses. However, its application to domain-specific PCQA remains underexplored. To address this gap, we propose **Multi-perspective Preference Alignment for Programming-Community Question Answering** to generate user-centric responses, called **MupPCQA**. It consists of three stages: Preference Standardization to control content quality, Preference Integration to account for diverse user tendencies, and Preference Timeliness Mitigation to address outdated answers. Extensive experiments on a high-quality, real-world PCQA dataset validate its accuracy and preference. Compared to its base model, MupPCQA shows an improvement of nearly 11% in BLEU, with increases of 20% and 17.5% in BERTScore and CodeBERTScore¹.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable success in the field of open-domain Question Answering (QA) (OpenAI, 2023; Anil et al., 2023; Chen et al., 2024). Additionally, Reinforcement Learning from Human Feedback (RLHF) can precisely control the behavior of LLMs, enabling alignment with human-like responses. However, its application to domain-specific QA (Rafailov et al., 2023; Huzaifah et al.,

2024) remains underexplored. For instance, in real-world Programming-Community Question Answering (PCQA), the misaligned LLM might produce redundant responses a_l compared to the human-answered a_1 and a_2 , as shown in Figure 1.

PCQA seeks to yield user-preferred responses containing functional code and guiding descriptions and primarily focuses on the interactions among users in code communities (e.g., Stack Overflow²). Recently, it has gained increasing significance in both academia (Liu and Wan, 2021; Zhou et al., 2018; Chen et al., 2017) and industry (Amancio et al., 2021; Ragkhitwetsagul et al., 2019; Kasela et al., 2023). Unlike conventional QA (Chen et al., 2024; Sun et al., 2024), PCQA exhibits three distinct characteristics. First, a question typically does not have just one answer, and as indicated in Table 4, nearly 46% of questions receive more than two answers, with some boasting answers 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.

Regarding the above, Code Llama (Rozière et al., 2023) have attempted to treat the accepted answer (e.g., a_2 in Figure 1) as the alignment target. However, it may not reflect the preferences of all users, as the answer chosen by the questioner may not be favored by other users. Some studies (Zhou et al., 2018; Maia et al., 2021; Du et al., 2021) have begun to focus on entire candidates and have introduced content-based ranking methods, but none have yet considered the inherent preferences of diverse users in PCQA and the feedback from LLMs. Additionally,

¹Our codes are at <https://github.com/YHY0607/PETCoQA>.

²<https://stackoverflow.com>

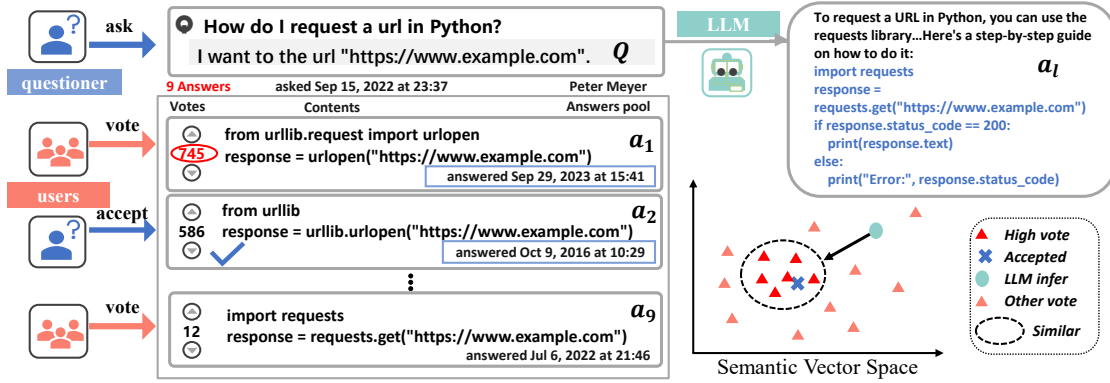


Figure 1: An example of a Programming-Community Question Answering. It encompasses key elements: a question Q , a pool of answers $\{a_1, \dots, a_9\}$. Each a_i contains its text of content, the votes, and a label indicating whether it has been accepted by the questioner. Additionally, in the semantic 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 .

user’s preferences change with API updates in these communities, as they tend to prefer newer versions of APIs. Consequently, the accepted answers may become outdated. For example, in Figure 1, the "urllib" API in answer a_2 applies to Python 2 but deprecated in Python 3.

To generate user-centric responses better, we propose **Multi-perspective Preference Alignment for Programming-Community Question Answering**, called MupPCQA. It consists of three stages: 1) **Preference Standardization** to transfer domain knowledge based on the questioner-accepted answer. 2) **Preference Integration** to consider diverse user tendencies. 3) **Preference Timeliness Mitigation** to alleviate outdated answers by retrieving similar question-answer pairs from the PCQA database as few-shots. Our contributions are:

- We are the first to propose applying LLMs’ alignment to programming-domain QA from the perspective of user diversity, called MupPCQA.
- Based on the questioner-perspective bias, user-perspective vote score, and LLM-perspective content score, MupPCQA realizes multi-perspective preference contrastive alignment through iteratively treating the answer with the highest score as the positive and the remaining as negative.
- Extensive zero-shot and few-shots experiments on a real-world and high-quality PCQA dataset validate MupPCQA accuracy and preference.

2 Related Work

2.1 Programming-Community QA

PCQA involves numerous research topics, such as predicting answerable questions (Asaduzzaman

et al., 2013), assessing answer quality (Raghitwet-sagul et al., 2019; Gao et al., 2020), answer generation (Zhou et al., 2018), and answer ranking (Amancio et al., 2021; Ginsca and Popescu, 2013; Liu and Wan, 2021; Dalip et al., 2013). These answer ranking methods typically employ classical deep-learning models to utilize both the answer text (Zhou et al., 2018) and the user’s fundamental characteristics (Ginsca and Popescu, 2013). For instance, L2R (Dalip et al., 2013) followed a learning to rank approach based on different groups of features like user-related features, stylistic or structural features. RCNN (Zhou et al., 2018) employed Gated Recurrent Units (GRU) with thread-level features to rank answers. Other research (Amancio et al., 2021) utilized recency and quality as criteria for ranking responses. However, few studies has considered the inherent preferences of diverse users and LLM feedback. Therefore, exploring how LLMs rank answers based on user preferences for alignment is a worthwhile endeavor.

3 Methodology

As shown in Figure 2, the MupPCQA encompasses three stages: (1) Preference Standardization, which is designed to quickly acquire programming-domain knowledge; (2) Preference Integration, which considers diverse user preferences; and (3) Preference Timeliness Mitigation, which addresses the issue of outdated answers.

3.1 Task Formulation

Our overall target is to instruct a LLM (M) to generate a user-centric response on a PCQA dataset $\mathcal{D} = \{(q^i, A^i) \mid i \in \{1, \dots, N\}\}$. Here, q^i represents the i -th question, and $A^i = \{a_1^i, a_2^i, \dots, a_{N_i}^i\}$

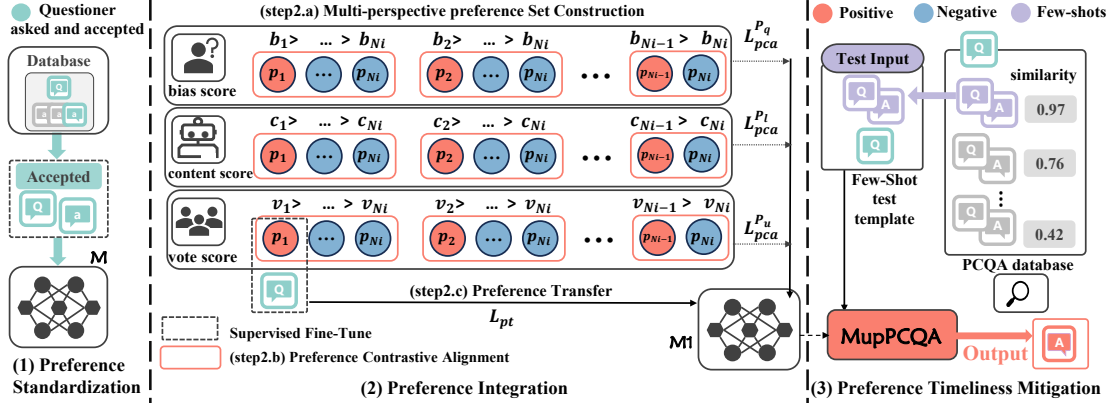


Figure 2: Overall of the MupPCQA framework, including three stages: (1) Preference Standardization to rapidly acquire the programming-domain knowledge, (2) Preference Integration to consider diverse preferences, and (3) Preference Timeliness Mitigation to address the issue of outdated answers. In Preference Integration, (step2.a) constructs three different perspective preference sets, (step2.b) performs preference contrastive alignment for each set. (step2.c) narrows the gap between the questioner-perspective goal and the viewer-perspective goal.

represents the candidate responses for q^i . We denote $a = (c, v, a_c)$, with c being the content; v being the votes; and $a_c \in \{0, 1\}$ representing whether the answer a is accepted by the questioner. Formally, any question q or content 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} and \mathcal{T} represents the set of code and text, respectively.

3.2 Preference Standardization

To adapt the foundational LLM M to the PCQA-specific corpus efficiently and control the quality of the response, we employ Supervised Fine-Tuning (Ouyang et al., 2022) and treat the questioner-accepted answer a^i with $a_c = 1$ as the alignment target, denoted as a_c^i . We optimize M as follows:

$$L_{ps} = -\frac{1}{|a_c^i|} \sum_{j=1}^{|a_c^i|} \log P_M(a_c^{(i,j)} | I, q^i, a_c^{(i,<j)}) \quad (1)$$

where $a_c^{(i,j)}$ is the j -th token of a_c^i , I is the QA prompt, and P_M denotes the token probability predicted by M , resulting in a model M_1 .

3.3 Preference Integration

After Preference Standardization ensures the content quality of the output, we proceed to align multi-perspective preferences from different users through Preference Integration. First, we propose distinct metric scores to build three preference sets: (1) a questioner-perspective bias score to assess the discrepancy between the accepted answer and other answers, (2) a user-perspective vote score to reflect the collective preferences of other users, and (3) a

LLM-perspective content score to evaluate the semantic quality of the answer content. These scores then serve as the foundation for Preference Contrastive Alignment, which differentiates between positive and negative. Finally, to bridge the gap between the questioner-perspective goal and the viewer-perspective goal, Preference Transfer applies SFT again.

3.3.1 Multi-perspective Preference Set Construction

First, as the answer chosen by the questioner may not be favored by other users (Kasela et al., 2023), we introduced the questioner-perspective bias score s_q to assess the discrepancy between the accepted answer and the answer most preferred by users.

$$s_q(a_i) = \frac{(v_i - v_a) - v_m}{v_\sigma} \quad (2)$$

Here, $V = \{v_1, \dots, v_{N_i}\}$ represents the set of votes v_i for all answers a_i in question q . The v_a denotes the votes for the answer accepted by the questioner. The v_m and v_σ represent the mean and standard deviation of the vote set V , respectively.

Second, since high-quality text is usually accompanied by a high number of votes (Gkotsis et al., 2014), interaction data from open communities reflect the preferences of users and act as a dual filtering mechanism for answer content quality. Therefore, to comprehensively consider the users' collective preferences and engagement with the answers, we introduce a user-perspective voting score, denoted as s_u , expressed as follows:

$$s_u(a_i) = \frac{v_i - \min(V)}{\max(V) - \min(V)} \quad (3)$$

Here, $\min(V)$ 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, given that high semantic accuracy is crucial for ensuring the quality of answers, we introduce an LLM-perspective content score s_l , calculated by general or code LLMs M_c . This score leverages LLMs, which excel at handling nuanced semantic relationships between text and code, to evaluate the quality of text c in answer a . The s_l is represented as follows:

$$s_l(a_i) = \prod_{t \in a_i} \sigma_c(I_1, q, t) \quad (4)$$

Here, σ_c is the logistic function derived from the product of probabilities assigned to each token by M_c . I_1 represents a QA prompt.

The sets of perspective scores S_q , S_u , and S_l can be derived by arranging the scores of candidates $\{a_1, \dots, a_{N_i}\}$ from different perspectives in descending order, and uniformly represented as:

$$\left\{ s(a_{i_1}), \dots, s(a_{i_{N_i}}) \mid s(a_{i_1}) \geq \dots \geq s(a_{i_{N_i}}) \right\} \quad (5)$$

where $s(\cdot)$ is the perspective score function, and the indices $\{i_1, \dots, i_{N_i}\}$ indicate the descending positions of the candidates. Specifically, $s_q(\cdot)$, $s_u(\cdot)$, and $s_l(\cdot)$ are the perspective score functions for each perspective.

If s_i , the score for answer a_j , is denoted as p_i , the preference response set can be represented as:

$$P = \{p_i \mid s_i \in S, i = 1, 2, \dots, N_i\} \quad (6)$$

Finally, the score sets S_q , S_u , and S_l are directly mapped to the preference sets P_q , P_u , and P_l .

3.3.2 Preference Contrastive Alignment

To comprehensively rank the preferences of each candidate in the preference sets P , we apply iterative contrastive learning over N_i rounds, utilizing the Plackett-Luce model (Luce, 1959; Plackett, 1975) as implemented in DPO (Rafailov et al., 2023). Details regarding RLHF and DPO are provided in Appendix 6.1.2. In each round, the candidate with the highest perspective score is treated as positive, while the others are treated as negative. Each example p_i is weighted based on its perspective score s_i . For a single preference set, the

objective for the i -th round is defined as follows:

$$O(i) = \frac{\exp(\sigma_{M_1}(p_i) \cdot s_i)}{\sum_{k=i}^{N_i} \exp(\sigma_{M_1}(p_k) \cdot s_k)} \quad (7)$$

Therefore, for all preference sets P_q , P_u , and P_l , the overall optimization objective for the i -th round is represented as follows:

$$O_t(i) = O^{P_q}(i) + O^{P_u}(i) + O^{P_l}(i) \quad (8)$$

As the i -th iteration progresses, the top $i - 1$ answers with the highest perspective scores are sequentially removed. This iterative process continues until all potentials are exhausted, yielding the final target probability expressed as follows:

$$L_{pca} = -\log \prod_{i=1}^{N_i-1} O_t(i) \quad (9)$$

3.3.3 Preference Transfer

At this stage, to narrow the gap between the questioner-perspective goal a_c in Preference Standardization and the viewer-perspective goal p_1^u in P_u , we further perform SFT by treating p_1^u as the alignment, similar to Eq. (1), as shown below.

$$L_{pt} = -\frac{1}{|p_1^u|} \sum_{i=1}^{|p_1^u|} \log P_M(p_1^{u(i)} | I, q, p_1^{u(<i)}) \quad (10)$$

Eventually, the overall optimization objective can be summarized as follows:

$$Loss = L_{pca} + \alpha L_{pt} \quad (11)$$

Here, α controls the extent to which the output response deviates towards the preferred response, ensuring its text fluency and code structure quality, resulting in a model M_2 .

3.4 Preference Timeliness Mitigation

Accepted answers may become outdated due to rapid API updates. For instance, in Figure 1, the "urllib" API in answer a_2 is applicable to Python 2 but deprecated in Python 3, we introduced Preference Timeliness Mitigation 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 employ a dense retriever (\mathcal{R}_D), designed to effectively map natural language questions to relevant code-generation tasks, enhancing

Table 1: The zero-shot experimental results on the PCQA dataset. Open-source code baselines are above MupPCQA and closed-source baselines are below MupPCQA. 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	79.34	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
MupPCQA(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

Table 2: The one-shot experimental results on the PCQA 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	52.64	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	<u>75.91</u>	52.23	47.52
MupPCQA (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

the retrieval of analogous question-answer pairs. The ordering of few-shot examples has a significant impact on model predictions (Zhao et al., 2021). Therefore, we select the most similar question-answer pair (f_q, f_a) from the PCQA databases \mathcal{D} to serve as the few-shot example in the QA prompt I_2 . The final inference objective is defined as follows:

$$\mathcal{P}(A_t) = \prod_{i=1}^T \sigma_{M_2}(A|I_2, Q, (f_q, f_a), A_{<t}) \quad (12)$$

Here, Q is the question to be resolved.

4 Experiment

4.1 Baselines and Dataset

Baselines. Based on the unique characteristics of PCQA, we selected two categories of baselines. The first consists of general-purpose, closed-source LLMs designed for text generation, including GPT-3.5-turbo, GPT-4 (OpenAI, 2023), PaLM (Anil et al., 2023), ChatGLM (Zeng et al., 2022), and

Claude2 (ant). The second comprises open-source code LLMs that excel in program synthesis, such as StarCoder (Li et al., 2023), WizardCoder-Python-13B (Luo et al., 2023), GPT-NeoX (Black et al., 2022), CodeGen-mono-16B (Nijkamp et al., 2022), and Code Llama 2 (Rozière et al., 2023).

Dataset. We collected real-world PCQA data from StackExchange³ and performed a series of data processing steps, as detailed in Appendix 6.2, resulting in 270,716 instances.

4.2 Evaluation Metrics

To comprehensively evaluate the experimental results, we employed various evaluation metrics from four perspectives: traditional text generation metrics (BLEU (Papineni et al., 2002), Rouge (Lin, 2004), and CHRF (Popović, 2015)), model-based metrics (BERTScore (Zhang et al., 2019)), code-related metrics (CodeBERTScore (Zhou et al., 2023)), and Preference based on GPT-4. Addition-

³<https://archive.org/details/stackexchange>

Table 3: The ablation study results. We evaluate performances brought by different components. The full names of these abbreviations are: PS (Preference Standardization); PI (Preference Integration); s_p (bias score); s_l (content scores); s_u (vote scores); and PTM (Preference Timeliness Mitigation). The components in **bold** have the most significant impact on performance.

Model	BLEU	ROUGE	CHRF	BS	CB-PR	CB-F
MupPCQA	22.86	25.48	40.58	84.14	65.12	63.53
-w/o PS	21.30	23.62	37.88	76.15	59.76	57.73
-w/o PI	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 PTM	21.66	23.16	39.18	81.82	61.22	62.53

ally, considering the similarity between Precision and Recall in CodeBERTScore, we unified these metrics into 'CodeBERTScore-PR' (CB-PR). Similarly, we merged the F1 and F-measure metrics into 'CodeBERTScore-F' (CB-F). The details of the experimental implementation are provided in the Appendix 6.3.

4.3 Analysis

4.3.1 Main Results

This experiment aims to explore whether MupCQA can outperform all baselines in terms of text and code generation accuracy within PCQA. Its results are shown in Table 1 and Table 2, and the key findings are summarized as follows:

MupPCQA excels. First, MupPCQA significantly outperformed all other baselines in every metric. Specifically, it surpassed the next-best baseline in every metric, outperforming 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. Second, MupPCQA achieved a BLEU score twice that of the benchmark model (22.86% vs. 11.86%). Other n-gram-based metrics (ROUGE and CHRF) and semantic-grammar-based metrics also showed substantial improvements, indicating that MupPCQA is an effective framework for enhancing the grammar and semantics of generated answers.

Significant benefit from retrieval-augmented strategy. Due to the presence of similar QA pairs

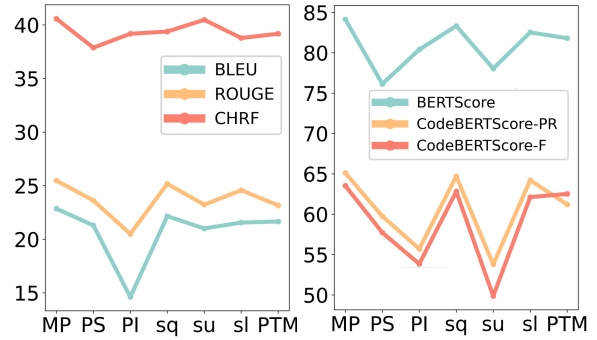


Figure 3: Performance of Each Component: Text-Based (Left) and Code-Based (Right). MP represents the overall performance of MupPCQA.

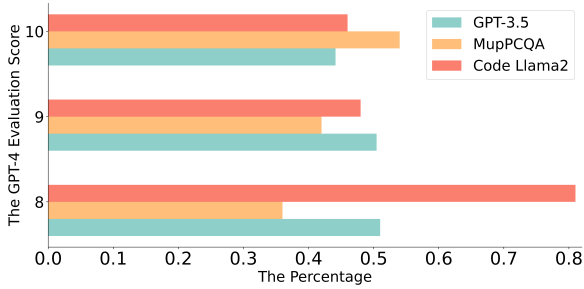
as few-shots in the Retrieval-Enhanced Preference Timeliness Mitigation (REPTM) of MupPCQA, we also applied few-shots to the remaining baselines in a new experiment. The results are shown in Table 2. This ensures a fair comparison of other zero-shot baselines with MupPCQA. In Table 2, each baseline shows improvement across various metrics compared to the zero-shot results in Table 1, with GPT-4 exhibiting significant enhancement in long-text performance, becoming the second-best baseline. However, MupPCQA's BLEU score remains much higher than that of the second-best GPT-4, and its BERTScore surpasses WizardCoder-Python by nearly 4.7%. In terms of CB-PR and CB-F, MupPCQA exceeds the second-best by nearly 10%. Although they still cannot match our MupPCQA, REPTM has played a significant role in performance improvement.

4.3.2 Ablation Study

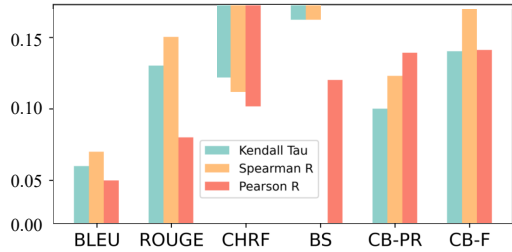
To validate the impact of each component on MupPCQA's performance, we conducted ablation experiments. The results are shown in Table 3.

Preference Standardization: A Prerequisite for Domain Knowledge Transfer. Upon removing it, 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 knowledge.

Preference integration affects mostly. Eliminating it resulted in notable decreases in performance on BLEU focused on complex phrase matching in Figure 3 (Left) and code semantics in Figure 3 (Right). Specifically, CodeBERTScore dropped by 8.8% and 9.4%, respectively. Additionally, the s_u positive impact on CodeBERTScore is much higher than on text-based metrics, and the effects of s_q and



(a) The percentage statistics of GPT-4 evaluation scores for different baselines.



(b) 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.

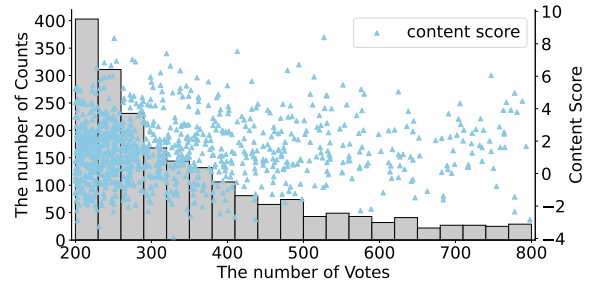
Figure 4: Analysis of GPT-4 evaluations for different baselines on the test set.

s_l are much more moderated. This suggests that an unadapted LLM fails to capture the diversity of preferences within the programming community.

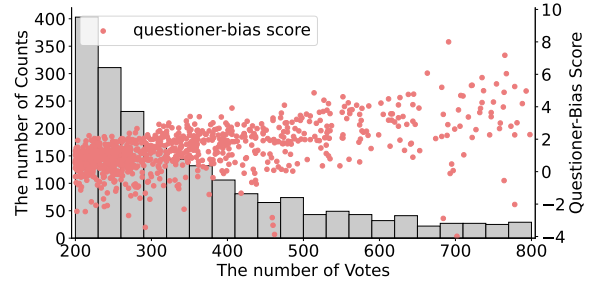
Similar few-shots promote semantics. If PTM is excluded, the performance on BERTScore and CodeBERTScore significantly declines, highlighting the critical role of similar examples in understanding problem semantics.

4.3.3 GPT-4 Evaluation

Given that GPT-4 (OpenAI, 2023) has demonstrated significant ability in effectively evaluating question-and-answer pairs and aligning with human preferences (Wang et al., 2023a; Zheng et al., 2023a), we utilize it to assess the preferences for responses generated by the Code Llama, the GPT-3.5, and our MupPCQA. 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, shown in Figure 8b. Each solution is rated on a scale from 1 to 10, with comprehensive explanations required for each score. As shown in Figure 4a, MupPCQA achieves a significantly higher percentage of 10-point scores compared to the other baselines.



(a) The distribution of Content Scores and their relationship with the number of Votes



(b) The distribution of questioner-perspective bias Scores and their relationship with the number of Votes

Figure 5: Multi-perspective Phenomenon Analysis.

To explore the consistency between accuracy-based metrics and preference-based metrics, we employed three key statistical correlation coefficients: Kendall’s Tau τ (Kendall, 1938), Spearman’s R γ (Franklin, 1974), and Pearson’s R ρ (Bravais, 1844), as depicted in Figure 4b. It primarily illustrates three points: First, the three correlation measures, τ , γ , and ρ , maintain a high degree of sign consistency. Second, text-based BLEU and ROUGE, semantic-based BERTScore, and code-based 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 the PCQA. Overall, accuracy and preference are not contradictory, providing a valuable reference for evaluating PCQA.

4.3.4 Multi-perspective Phenomenon Analysis

To validate the necessity of multi-perspective preference modeling, we randomly extracted 2,000 entries from the PCQA dataset. We calculated the s_q , s_u and s_l for each answer a . These scores were then mapped onto a two-dimensional coordinate system. 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

Question	The response of Code Llama	The response of MupCCQA
(1) Reloading submodules in IPython?	1.First, import the autoreload 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	<code>1.python -c import distutils.sysconfig; print(distutils.sysconfig.get_python_inc()); 2.export PYTHON_INCLUDE_DIR=/path/to/Python.h</code> <code>Replace /path/to/Python.h</code> 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 Uutilc.</code> GPT-4 Score: 10
(3) Can I force pip to re-install 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.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, 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 croniter library can be used with the sched module from the standard library to create a Cron-like scheduler. <code>import sched...import time...from croniter import croniter.scheduler = sched.scheduler(time.time, time.sleep)</code> GPT-4 Score: 7	The APScheduler library is a more modern and feature-rich option for scheduling tasks. <code>from apscheduler.schedulers.blocking import BlockingScheduler from apscheduler.triggers.cron import CronTrigger.scheduler = BlockingScheduler() scheduler.add_job(scheduled_task, CronTrigger.from_crontab("))</code> GPT-4 Score: 9

Figure 6: A case study about Code Llama and MupPCQA in PCQA. The black signifies natural language. The blue represents code. The red 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].

answer are close to the votes for the answer chosen by the questioner, then s_q is near the X-axis. In Figure 5b, 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 in Figure 5a: 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 PCQA, necessitating the adoption of multi-perspective modeling approaches.

4.3.5 Case Study

To validate the excellence of our MupPCQA, we selected four random questions for comparison, as shown in Figure 6, and we take randomly the third question as an example. MupPCQA scored 10, while Code Llama scored 8. Although both responses covered the core points and clearly explained how to use the "force-reinstall" flag, MupPCQA excelled in the following aspects: First, in detail: MupPCQA 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: MupPCQA accurately explained the usage of the "upgrade" and "upgrade-strategy" flags, making the response more informative and help-

ful for managing package versions and upgrades. Third, in user-friendliness: MupPCQA'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 MupPCQA utilized a more contemporary library "APScheduler", a popular and feature-rich option for scheduling tasks.

4.4 Success and failure analysis

To discuss under what conditions MupPCQA performs optimally, we selected some successful cases as shown in Figure 9a. Their characteristics are analyzed and summarized as follows: (1) Clear problem statements: Successful cases usually have clear, specific problem statements, making the user's intent and requirements evident. (2) Complete context: They provide sufficient context for the model, enabling better understanding of the problem and generating responses for better alignment.

To explore the scenarios where MupPCQA's performance is limited, we also pick some failure cases as shown in Figure 9a. These factors are outlined: (1) Vague problem statements: Problem statements in failure cases may be ambiguous, leading to difficulty in understanding. (2) Insufficient context: Failure cases lack adequate context, causing the LLMs to lack understanding of the problem's background, resulting in impractical responses. (3) Excessively long and redundant context.

5 Conclusion

In this paper, to explore the application of LLMs in PCQA for generating user-centric responses through RLHF, we propose MupPCQA. It measures and synthesizes responses' preference levels from different perspectives to accommodate diverse and evolving user preferences based on Multi-perspective Preference Contrastive Alignment. Extensive experiments validate the superior accuracy and preference of the MupPCQA's responses. Overall, this work highlights considering the diversity of user preferences to generate human-like responses while aligning with human inclinations.

6 Ethical Statement

It should be acknowledged that there is a tiny possibility that sensitive and offensive content may exist within the dataset. It is important to emphasize that this content does not reflect our views or beliefs, but is solely intended for research purposes.

Acknowledgement

The work was supported in part by the National Natural Science Foundation of China under Grant U23A20308, and by the Fundamental Research Funds for the Central Universities.

References

- Anthropic. model card and evaluations for claude models, 2023. <https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf>.
- Utkarsh Agarwal, Kumar Tanmay, Aditi Khandelwal, and Monojit Choudhury. 2024. Ethical reasoning and moral value alignment of llms depend on the language we prompt them in. *arXiv preprint arXiv:2404.18460*.
- Leandro Amancio, Carina F Dorneles, and Daniel H Dalip. 2021. Recency and quality-based ranking question in cqas: A stack overflow case study. *Information Processing & Management*, 58(4):102552.
- Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.
- Muhammad Asaduzzaman, Ahmed Shah Mashiyat, Chanchal K Roy, and Kevin A Schneider. 2013. Answering questions about unanswered questions of stack overflow. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 97–100. IEEE.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.
- Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. 2022. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*.
- 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.
- Auguste Bravais. 1844. *Analyse mathématique sur les probabilités des erreurs de situation d'un point*. Impr. Royale.
- Lei Chen, Bobo Li, Li Zheng, Haining Wang, Zixiang Meng, Runfeng Shi, Hao Fei, Jun Zhou, Fei Li, Chong Teng, et al. 2024. What factors influence llms' judgments? a case study on question answering. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 17473–17485.
- Qin Chen, Qinmin Hu, Jimmy Xiangji Huang, Liang He, and Weijie An. 2017. 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.
- Daniel Hasan Dalip, Marcos André Gonçalves, Marco Cristo, and Pavel Calado. 2013. 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.
- Jiangnan Du, Jun Chen, Suhong Wang, Jianfeng Li, and Zhifeng Xiao. 2021. 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.
- Zhipeng Gao, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. 2020. Generating question titles for stack overflow from mined code snippets. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(4):1–37.
- Alexandru Lucian Ginsca and Adrian Popescu. 2013. 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.

- George Gkotsis, Karen Stepanyan, Carlos Pedrinaci, John Domingue, and Maria Liakata. 2014. 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.
- Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Code-searchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.
- Muhammad Huzaifah, Weihua Zheng, Nattapol Chanpaisit, and Kui Wu. 2024. Evaluating code-switching translation with large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 6381–6394.
- Pranav Kasela, Gabriella Pasi, and Raffaele Perego. 2023. Se-pqa: Personalized community question answering. *arXiv preprint arXiv:2306.16261*.
- Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93.
- Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.
- Taiji Li, Zhi Li, and Yin Zhang. 2024. Improving faithfulness of large language models in summarization via sliding generation and self-consistency. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 8804–8817.
- Bo Lin, Shangwen Wang, Zhongxin Liu, Yepang Liu, Xin Xia, and Xiaoguang Mao. 2023. Cct5: A code-change-oriented pre-trained model. *arXiv preprint arXiv:2305.10785*.
- Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.
- Chenxiao Liu and Xiaojun Wan. 2021. Codeqa: A question answering dataset for source code comprehension. *arXiv preprint arXiv:2109.08365*.
- Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*.
- R Duncan Luce. 1959. *Individual choice behavior*, volume 4. Wiley New York.
- Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.
- Macedo Maia, Siegfried Handschuh, and Markus Endres. 2021. 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.
- Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.
- OpenAI. 2023. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. 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.
- Robin L Plackett. 1975. The analysis of permutations. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2):193–202.
- Maja Popović. 2015. chrF: character n-gram f-score for automatic mt evaluation. In *Proceedings of the tenth workshop on statistical machine translation*, pages 392–395.
- A Franklin. 1974. *Introduction to the Theory of Statistics*.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*.
- Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. 2019. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering*, 47(3):560–581.
- Mengjie Ren, Boxi Cao, Hongyu Lin, Cao Liu, Xi-anpei Han, Ke Zeng, Guanglu Wan, Xunliang Cai, and Le Sun. 2024. Learning or self-aligning? rethinking instruction fine-tuning. *arXiv preprint arXiv:2402.18243*.

- Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Feifan Song, Bowen Yu, Hao Lang, Haiyang Yu, Fei Huang, Houfeng Wang, and Yongbin Li. 2024. Scaling data diversity for fine-tuning language models in human alignment. *arXiv preprint arXiv:2403.11124*.
- Feifan Song, Bowen Yu, Minghao Li, Haiyang Yu, Fei Huang, Yongbin Li, and Houfeng Wang. 2023. Preference ranking optimization for human alignment. *arXiv preprint arXiv:2306.17492*.
- Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021.
- Kexuan Sun, Nicolaas Paul Jedema, Karishma Sharma, Ruben Janssen, Jay Pujara, Pedro Szekely, and Alessandro Moschitti. 2024. Efficient and accurate contextual re-ranking for knowledge graph question answering. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 5585–5595.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shrutu Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- Peiyi Wang, Lei Li, Liang Chen, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023a. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*.
- Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. 2023b. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*.
- Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. 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.
- Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, Songfang Huang, and Fei Huang. 2023. Rrhf: Rank responses to align language models with human feedback without tears. *arXiv preprint arXiv:2304.05302*.
- Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.
- Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.
- Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*, pages 12697–12706. PMLR.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023a. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.
- Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. 2023b. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*.
- Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. 2023. Codebertscore: Evaluating code generation with pretrained models of code. *arXiv preprint arXiv:2302.05527*.
- Shuyan Zhou, Uri Alon, Frank F Xu, Zhengbao Jiang, and Graham Neubig. 2022. Docprompting: Generating code by retrieving the docs. In *The Eleventh International Conference on Learning Representations*.
- Xiaoqiang Zhou, Baotian Hu, Qingcai Chen, and Xiaolong Wang. 2018. Recurrent convolutional neural network for answer selection in community question answering. *Neurocomputing*, 274:8–18.

Appendix

6.1 Preliminary

6.1.1 Reinforcement Learning from Human Feedback

We introduce Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022), which primarily comprises three stages. The first stage is supervised fine-tuning on a LLM, denoted as M , which is also a component of our framework and will be elaborated in Section 3.2. The second stage involves using the SFT model M_1 to generate pairs of responses for a given prompt I . These pairs have a preference order, as illustrated by p_i is preferred over p_j in Figure 7 (b). To predict these pairs, current works typically employ the Bradley-Terry (BT) model (Bradley and Terry, 1952), which defines the preference probability as follows:

$$\mathcal{P}_{BT} = \frac{\exp(r_\phi(I_1, p_i))}{\exp(r_\phi(I_1, p_i)) + \exp(r_\phi(I_1, p_j))} \quad (13)$$

Where r_ϕ is inherently a binary classification reward model, and 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:

$$L_{BT} = -\log \sigma(r_\Phi(I_1, p_i) - r_\Phi(I_1, p_j))$$

In the third stage, RLHF leverages the acquired r_ϕ to provide feedback to M_1 and σ is the logistic function. Specifically, the optimization problem of RLHF is formulated the following :

$$\max_{M_2} \mathbb{E}(r_\Phi(I_1, p) - \xi \log \frac{M_2(p|I_1)}{M_1(p|I_1)})$$

In this context, the role of ξ is to regulate the deviation from the baseline reference policy M_1 , ensuring diversity in the generated outputs and preventing the production of high-reward yet nonsensical 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.

6.1.2 the Plackett-Luce Model

The Plackett-Luce model (Plackett, 1975) is a generalization of the Bradley-Terry (Bradley and Terry, 1952) model to rankings (rather than just pairwise comparisons). Similar to the Bradley-Terry model, it stipulates that when faced with a set of possible choices, individuals prefer a choice with a

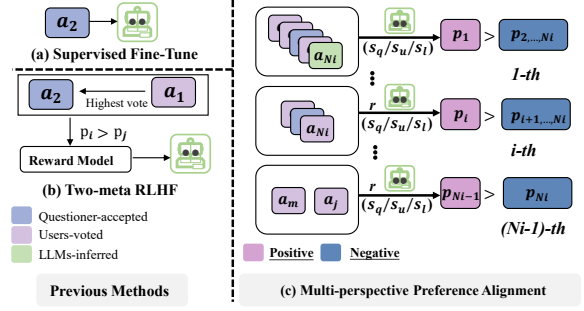


Figure 7: In PCQA, we compared previous human alignment methods with our approach. (a) SFT aligns only the answer accepted by the questioner a_2 , while (b) RLHF compares a_2 with the highest-voted user-preferred answer a_1 , sampling two-meta candidates $p_i \succ p_j$ from the entire ranking to train 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 preference score, which includes bias scores s_q , vote scores s_u , and content scores s_i .

probability proportional to the value of some latent reward function for that choice. In our context, given a question q and a set of candidate responses a_1, a_2, \dots, a_K , a user outputs a permutation $\tau : [K] \rightarrow [K]$ that represents their ranking of the answers. The Plackett-Luce model specifies as follows:

$$p^*(\tau | a_1, \dots, a_K, q) = \frac{\exp(r^*(q, a_{\tau(k)}))}{\sum_{j=k}^K \exp(r^*(q, a_{\tau(j)}))} \quad (14)$$

Please note that when $K = 2$, Equation 14 simplifies to the Bradley-Terry model. However, for the general Plackett-Luce model, we can still utilize the logistic probability to replace the reward function similar with the DPO (Rafailov et al., 2023).

$$r(q, a) = \beta \log \frac{\pi_{\text{ref}}(a | q)}{\pi_\tau(a | q)} + \beta \log Z(q). \quad (15)$$

This Equation 15 represents the reward function in terms of its corresponding optimal policy π_τ , reference policy π_{ref} , and the unknown partition function $Z(\cdot)$. When the normalization constant $Z(x)$ cancels out and we're left with:

$$p^*(\tau | a_1, \dots, a_K, q) = \frac{\exp\left(\beta \log \frac{\pi^*(a_{\tau(k)}|q)}{\pi_{\text{ref}}(a_{\tau(k)}|q)}\right)}{\sum_{j=k}^K \exp\left(\beta \log \frac{\pi^*(a_{\tau(j)}|q)}{\pi_{\text{ref}}(a_{\tau(j)}|q)}\right)} \quad (16)$$

For the PCQA dataset $D = \{(q_i, \{a_1, \dots, a_{N_i}\})\}$, which contains prompts and user-specified rank-

ings, we can use a parameterized model and optimize this objective using maximum likelihood:

$$L(\pi_\theta, \pi_{\text{ref}}) = -\mathbb{E} \log \frac{\exp\left(\beta \log \frac{\pi_\theta(a_{\tau(k)}|q)}{\pi_{\text{ref}}(a_{\tau(k)}|q)}\right)}{\sum_{j=k}^K \exp\left(\beta \log \frac{\pi_\theta(a_{\tau(j)}|q)}{\pi_{\text{ref}}(a_{\tau(j)}|q)}\right)} \quad (17)$$

6.2 Dataset

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

The initial dataset consists of 757,702 (q_i, a_i) pairs, primarily featuring `<python>` tags, with 600,176 pairs containing code blocks. To obtain the latest answers, we systematically collected all answers for each question q on Stack Overflow up until August 2023, resulting in a dataset totaling 596,613 pairs. Detailed statistics are presented in Table 4. We then performed 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 (Lin et al., 2023), which stipulates that code comments should contain more than three tokens.
- Pairs where answer does not contain `<code> ... </code>` content were eliminated to ensure that MupPCQA’s reference content includes both text and code, due to the nature of PCQA.
- Pairs with an answer pool size smaller than 2 were discarded.
- All HTML tags were cleaned and replaced with "[HTML]", particularly `<a href...>` 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 (Husain et al., 2019; Lu et al., 2021).

6.3 Implementation Details

Code Llama has demonstrated state-of-the-art performance across various code benchmarks. Therefore, we utilized the MupPCQA framework based on it for preference alignment in PCQA.

In Preference Standardization, 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. And 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 link⁴.

In Preference Integration, we selected an accessible LLM⁵ as M_c . The hyperparameters were set: *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 of this retrieval-generation approach in understanding diverse texts and code, during the Retrieval-augmented Preference Timeliness Mitigation phase, we chose the DocPrompting method based on SimCSE (Zhou et al., 2022) as our retriever \mathcal{R}_D . This retriever \mathcal{R}_D includes 35,763 functions from all Python libraries on DevDocs⁶, encompassing the Python standard library and widely-used packages such as NumPy and Pandas, and was pre-trained on the re-split CoNaLa (Yin et al., 2018) benchmark.

6.4 Related Work

6.4.1 Alignment of LLMs.

The language modeling objective of LLMs (e.g., predicting the next word) differs from the ultimate goals in LLM applications, such as following instructions and being helpful, factual, and harmless (Ouyang et al., 2022). Equivalently, the behavior of pre-trained LLMs may not necessarily align with the principles of their intended use cases. Therefore, alignment of LLMs (Song et al., 2024; Agarwal et al., 2024) aims to adjust the outputs of general pre-trained language models to better align with human preferences, significantly improving the performance of LLMs in various downstream applications, such as summarization (Li et al., 2024),

⁴<https://github.com/facebookresearch/llama>

⁵OpenAssistant/reward-model-deberta-v3-large

⁶<https://devdocs.io>

Table 4: Statistics on the size of the answers pool for each question about the PCQA dataset.

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

translation (Huzafah et al., 2024), and question-answering (Chen et al., 2024). Currently, the two most common alignment techniques are instruction tuning (Song et al., 2024; Agarwal et al., 2024) and RLHF (Ouyang et al., 2022). Additionally, emerging alignment techniques such as Constitutional AI (Bai et al., 2022) and self-alignment (Ren et al., 2024) are also gaining attention. These primarily focus on embedding alignment rules into pre-trained models to constrain harmful behavior during inference. However, they have not explored how to align in the presence of diverse user preferences. Our study demonstrates that different users have varying tendencies.

6.4.2 Preference Alignment for Question Answering

In recent years, the LLMs (OpenAI, 2023; Anil et al., 2023; Zeng et al., 2022; Touvron et al., 2023a,b) 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 (Nijkamp et al., 2022; Zheng et al., 2023b; Li et al., 2023; Wang et al., 2023b; Rozière et al., 2023). This capability enables them to adeptly navigate the complexities of programming problems, including conceptual understanding, code generation, API utilization, and debugging.

Recently, reinforcement learning from human feedback (RLHF) (Stiennon et al., 2020; Ouyang et al., 2022; Rozière et al., 2023) 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) (Schulman et al., 2017), 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

(Yuan et al., 2023) introduced a boundary ranking loss function to optimize LLMs without requiring an additional reward model. DPO (Rafailov et al., 2023) introduced a direct preference optimization method, treating LLMs themselves as the reward model. PRO (Song et al., 2023) 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.

The zero-shot prompt for baseline LLMs

Please play the role of a senior programmer.
The following is the title and question of a post in a programming community.
Please help me answer the question.
Title: {title}
Problem Description: {query}

(a) The QA prompt of the closed general-purpose LLMs

Preference-Based GPT-4 Evaluation Prompt

[Question] [Q]
[The Start Answer1] [R1] [The End of Assistant 1’s Answer]
[The Start of Assistant 2’s Answer] [R2] [The End of Assistant 2’s Answer]
[System] We would like to request your feedback on the performance of two AI assistants in response to the user question displayed above. Please rate the **helpfulness, relevance, accuracy, level of details of their responses**. Each assistant receives an overall score on a scale of 1 to 10, where a higher score indicates better overall performance. Please first output a single line containing only two values indicating the scores for Assistant 1 and 2, respectively. The two scores are separated by a space. In the subsequent line, please provide a comprehensive explanation of your evaluation, avoiding any potential bias and ensuring that the order in which the responses were presented does not affect your judgment.

(b) The prompt of GPT-4 Evaluation

Retrieval-augmented In-context Learning Prompt

Instruction=[

```
{
  "role": "user",
  "content": "### Q1: {The title of the most similar query sq} "
},
{
  "role": "assistant",
  "content": "### A1: {The answer of the most similar query sa} "
},
{
  "role": "user",
  "content": "f"""" # Based on the above similar questions, write a response that appropriately completes the request
### Instruction: {question} """" }
```

(c) The QA prompt of Preference Timeliness Mitigation

Figure 8: The prompt sets

Successful case

(1) Reloading submodules in IPython?

Currently I am working on a python project that contains sub modules and uses numpy/scipy. IPython is used as interactive console. Unfortunately I am not very happy with workflow that I am using right now, I would appreciate some advice. In IPython, the framework is loaded by a simple `import` command. However, it is often necessary to change code in one of the submodules of the framework. At this point a model is already loaded and I use IPython to interact with it. Now, the framework contains many modules that depend on each other, i.e. when the framework is initially loaded the main module is importing and configuring the submodules. The changes to the code are only executed if the module is reloaded using `reload(main_module.sub_mod)`. This is cumbersome as I need to reload all changed modules individually using the full path. It would be very convenient if `reload(main_module)` would also reload all sub modules, but without reloading numpy/scipy.

(2) Is there a portable way to get the current username in Python?

What is a portable way (e.g. for Linux and Windows) to get the current user's username? Something similar to `os.getuid()` would be nice: `>>> os.getuid()` # Does not currently exist in Python `>>> os.getusername()'slartibartfast'` The `pwd` module works for Unix only. Some people suggest that getting the username under Windows can be complicated in certain circumstances (e.g., running as a Windows service).

(3) How do I get a Cron like scheduler in Python?

I'm looking for a library in Python which will provide `at` and `cron` like functionality. I'd quite like have a pure Python solution, rather than relying on tools installed on the box; this way I run on machines with no cron. For those unfamiliar with `cron`: you can schedule tasks based upon an expression like: `0 2 * * 7 /usr/bin/run-backup` # run the backups at 0200 on Every Sunday 0 9-17/2 * * 1-5 /usr/bin/purge-temps # run the purge temps command, every 2 hours between 9am and 5pm on Mondays to Fridays. The cron time expression syntax is less important, but I would like to have something with this sort of flexibility. If there isn't something that does this for me out-the-box, any suggestions for the building blocks to make something like this would be gratefully received. Edit I'm not interested in launching processes, just `'jobs'` also written in Python - python functions. By necessity I think this would be a different thread, but not in a different process. To this end, I'm looking for the expressivity of the cron time expression, but in Python. Cron has been around for years, but I'm trying to be as portable as possible. I cannot rely on its presence.

(4) Can I force pip to reinstall the current version?

I've come across situations where a current version of a package seems not to be working and requires reinstallation. But `pip install -U` won't touch a package that is already up-to-date. I see how to force a reinstallation by first uninstalling (with `pip uninstall`) and then installing, but is there a way to simply force an `'update'` to a nominally current version in a single step?

(5) How can I write a `'try'/'except'` block that catches all exceptions?

How can I write a `try` `except` block that catches all exceptions?

(a) Examples where MupPCQA performs well.

Failure cases

(1) Checking whether a string starts with XXXX?

I would like to know how to check whether a string starts with `'hello'` in Python. In Bash I usually do: `if [["$string" =~ ^hello]]; then do something herefi` How do I achieve the same in Python?

(2) Find which version of package is installed with pip?

Using pip, is it possible to figure out which version of a package is currently installed? I know about `pip install XYZ --upgrade` but I am wondering if there is anything like `pip info XYZ`. If not what would be the best way to tell what version I am currently using.

(3) Python datetime to string without microsecond component?

I'm adding UTC time strings to Bitbucket API responses that currently only contain Amsterdam (!) time strings. For consistency with the UTC time strings returned elsewhere, the desired format is `2011-11-03 11:07:04` (followed by `+00:00`, but that's not germane). What's the best way to create such a string (without a microsecond component) from a `datetime` instance with a microsecond component? `>>> import datetime>>> print unicode(datetime.datetime.now())` 2011-11-03 11:13:39.278026 I'll add the best option that's occurred to me as a possible answer, but there may well be a more elegant solution. Edit: I should mention that I'm not actually printing the current time `\u2013` I used `datetime.now` to provide a quick example. So the solution should not assume that any `datetime` instances it receives will include microsecond components.

(b) Examples where MupPCQA performs poorly.

Figure 9: The Successful and failure cases of the MupPCQA

Q Suppress InsecureRequestWarning: Unverified HTTPS request is being made in Python2.6

Description: I am writing scripts in Python2.6 with use of pyVmomi and while using one of the connection methods:

```
service_instance = connect.SmartConnect(host=args.ip,user=args.user,pwd=args.password)
```

I get the following warning:

```
usr/lib/python2.6/site-packages/requests/packages/urllib3/connectionpool.py:734: InsecureRequestWarning: Unverified HTTPS request is being made. Adding
```

certificate verification is strongly advised. See: <https://urllib3.readthedocs.org/en/latest/security.html> InsecureRequestWarning

What's interesting is that I do not have urllib3 installed with pip (but it's there in /usr/lib/python2.6/site-packages/requests/packages/urllib3/). I have tried as suggested here `import urllib3...urllib3.disable_warnings()` but that didn't change anything.

Accepted Solution: I'm using Angular2 routing and I need to subscribe to an event when route is changed. (i.e. user clicked on one of the route links). The important thing is that the event should be when the new view HTML is inserted to DOM. Are there any events like onNavigating and onNavigated, so I can subscribe to?

I've found a couple of examples on stackoverflow and tried to use them (see constructor below), but that didn't work. Any ideas? `<reference path= "../typings/tsd.d.ts" />import {Component, View} from 'angular2/angular2';import {RouteConfig, Router, RouterOutlet, RouterLink} from 'angular2/router';@RouteConfig([...])@Component({selector: 'app'})@View({directives: [RouterOutlet, RouterLink],`

```
template: `<div class="mdl-layout mdl-js-layout mdl-layout--fixed-header"><header class="mdl-layout__header"><div class="mdl-layout__header-row">
```

```
<span class="mdl-layout-title">Test</span><div class="mdl-layout-spacer"></div><nav class="mdl-navigation mdl-layout--large-screen-only">
```

```
<a class="mdl-navigation__link" [router-link]="['/routerlink1']">routerlink1</a></nav></div></header><main class="mdl-layout__content">
```

```
style="padding: 20px;"><router-outlet></router-outlet></main></div>`}
```

```
export class App {constructor(private router: Router){router.subscribe((val) => function(){...}); //here is I need to process HTML}}
```

ChatGPT: It seems like you are facing an issue with a warning that occurs while executing Python2.6 scripts with pyVmomi. It warns you of an unverified HTTPS request being made. To suppress the warning, you can disable the requests warnings by adding the following code to your script:

```
import requests
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
```

This will disable the warnings caused by an unverified HTTPS request.

ChatGLM: As a senior programmer, I would recommend the following solution to suppress the InsecureRequestWarning: First, ensure that you have the latest version of the requests library installed. You can upgrade the requests library using pip by running:

```
""pip install --upgrade requests ""
```

If the warning still persists, you can use the following code to suppress the warning:

```
""python import requests.packages.urllib3 requests.packages.urllib3.disable_warnings() ""
```

Make sure to add this code before making the HTTPS request. However, it's important to note that disabling SSL verification is not a recommended practice, as it can expose your application to security risks. Instead, you should verify the SSL certificate and ensure that you are using a secure connection. If you are using a custom CA certificate or self-signed certificate, you can use the 'verify' parameter in your HTTPS request to specify the certificate file:

```
import requests
url = 'https://your_host' \ncert_file = 'path/to/your/certificate.crt'
response = requests.get(url, verify=cert_file)
```

If you still face issues, please provide more information about your environment, such as the exact version of Python, pyVmomi, and requests libraries, as well as any other relevant information that could help diagnose the problem.

GPT-4: The warning message you are seeing is related to making insecure HTTPS connections without verifying the server's certificate. This could potentially lead to security issues, as this is how man-in-the-middle attacks often start. Therefore, the best solution would be to use a verified, secure connection.

If you insist on silencing the warnings though, you need to disable the warnings for urllib3 that's built into 'requests', not a separate urllib3 you might have installed. Here is how you could do it:

```
import requests
requests.packages.urllib3.disable_warnings()
```

Be aware that this will disable all warnings about insecure HTTPS connections, which might not be what you want if you're working with different hosts, some of which might actually have improperly configured HTTPS servers.

MupPCQA: The warning you're seeing is caused by the 'urllib3' library, which is used by the 'requests' library that 'pyVmomi' uses under the hood. The warning is telling you that the connection you're making is not using SSL/TLS, which is not secure. The reason you're not seeing the warning when you run the script with 'python3' is because 'python3' uses a different version of 'urllib3' that doesn't have this warning.

To fix the warning, you can disable the warning by adding the following line at the beginning of your script:

```
import urllib3\nurllib3.disable_warnings()
```

This will disable the warning for all connections made by 'pyVmomi'. Alternatively, you can also add the following line to your script to disable the warning for a specific connection:

```
import urllib3\nurllib3.disable_warnings(urllib3.InsecureRequestWarning)
```

This will disable the warning for the specific connection you're making. Note that disabling the warning is not a solution to the problem, it's just a way to hide the warning. You should make sure that your connection is secure by using a valid SSL/TLS certificate.

MupPCQA+Preference Timeliness Mitigation: The warning you're seeing is caused by the 'urllib3' library being used by the 'requests' library, which is a dependency of 'pyVmomi'. The warning is telling you that the HTTPS connection you're making is not verified, which means that the SSL certificate presented by the server is not trusted.

To suppress the warning, you can try the following:

1. Install 'urllib3' using 'pip': `pip install urllib3`
2. Disable the warning by adding the following line at the beginning of your script: `import urllib3; urllib3.disable_warnings()`
3. Use the 'verify' parameter of the 'connect.SmartConnect' method to specify a path to a CA certificate bundle that you trust. For example: `service_instance = connect.SmartConnect(host=args.ip, user=args.user, pwd=args.password, verify='/path/to/ca_certs.pem')`

By using one of these methods, you should be able to suppress the warning and make your HTTPS connection secure

Figure 10: one real-world post on the programming community and their answers of LLMs.