Improving Socratic Question Generation using Data Augmentation and Preference Optimization

Nischal Ashok Kumar and **Andrew Lan**
University of Massachusetts Amherst
{nashokkumar, andrewlan}@cs.umass.edu

Abstract

The Socratic method is a way of guiding students toward solving a problem independently without directly revealing the solution to the problem by asking incremental questions. Although this method has been shown to significantly improve student learning outcomes, it remains a complex labor-intensive task for instructors. Large language models (LLMs) can be used to augment human effort by automatically generating Socratic questions for students. However, existing methods that involve prompting these LLMs sometimes produce invalid outputs, e.g., those that directly reveal the solution to the problem or provide irrelevant or premature questions. To alleviate this problem, inspired by reinforcement learning with AI feedback (RLAIF), we first propose a data augmentation method to enrich existing Socratic questioning datasets with questions that are invalid in specific ways. Also, we propose a method to optimize open-source LLMs such as LLama 2 to prefer ground-truth questions over generated invalid ones, using direct preference optimization (DPO). Our experiments on a Socratic questions dataset for student code debugging show that a DPO-optimized LLama 2-7B model can effectively avoid generating invalid questions, and as a result, outperforms existing state-of-the-art prompting methods[1].

1 Introduction

Learning based on a conversation that consists of questions and answers, where the student responds to questions posed by a more knowledgeable instructor, has been proven to be effective in teaching students about a particular concept (Wood et al., 1976). In particular, *Socratic questioning*, which refers to a way for the instructor to guide a student to solve a problem (within their zone of proximal development) by asking them questions that pro-

mote thinking while not directly revealing the solution (Quintana et al., 2018), is a very effective pedagogical method in conversation-based learning and tutoring.

Recent advances in large language models (LLMs) (Bubeck et al., 2023) have led to the rapid development of chatbots that promote student learning by automatically generating the instructor's utterances (Dan et al., 2023; Kazemitabaar et al., 2024; Tanwar et al., 2024). One key area of interest in the development of such chatbots is question generation, which can help students solve logical problems in the mathematics and programming domains (Al-Hossami et al., 2023; Shridhar et al., 2022). Typically, question generation in educational applications has focused on generating practice or assessment questions, in biology exams (Wang et al., 2018), reading comprehension (Ashok Kumar et al., 2023), math practice (Wang et al., 2021), and programming exercises (Sarsa et al., 2022). As a specific form of question generation, Socratic question generation has gained attention, owing to its effectiveness in improving student learning outcomes by eliciting critical thinking and self-discovery during problem-solving (Paul and Elder, 2007).

Socratic questions generation is a complex task because it involves mapping out the step-by-step thought process of students during problem-solving, locating the cause of their error, and providing effective questions without revealing the solution. Manually generating Socratic questions can be a cognitively demanding and time-consuming task for instructors. Several recent works proposed to automatically generate Socratic questions using LLMs: In math education, (Shridhar et al., 2022) shows that generating a sequence of Socratic sub-questions and prompting students to answer helps them solve math word problems more successfully. In computer science education, (Al-Hossami et al., 2024, 2023) releases a dataset on Socratic questions for student code debugging and provides baselines

---

based on LLM prompting and finetuning. In particular, the authors prompt GPT-3.5-turbo and GPT-4 (Bubeck et al., 2023) in a chain-of-thought manner (Wei et al., 2022) to generate Socratic questions. A human study by the authors shows that the generated questions can sometimes be invalid in several different ways, including being irrelevant to the problem, repetitive of earlier dialogue turns, or too direct and revealing the solution prematurely, which may hamper students' learning processes. Since GPT models are proprietary and expensive, the authors also attempt to fine-tune the open-source Flan-T5 model (Chung et al., 2022); however, doing so proves to be ineffective due to its insufficient scale and the pretraining procedure used.

In this paper, we propose a method to improve the validity of automatically generated Socratic questions using open-source LLMs. Our method is inspired by recent developments in reinforcement learning with AI feedback (RLAIF) (Lee et al., 2023); our method consists of two phases, data augmentation and preference optimization. Specifically, our contributions are as follows:

- To the best of our knowledge, this work is the first to introduce a data augmentation method to create negative samples, i.e., invalid questions, to help us train LLM-based Socratic question generation methods.

- We use the preference information in the dataset, i.e., pairs of valid and invalid Socratic questions, to optimize Llama 2 (Touvron et al., 2023), an open-source LLM, using direct preference optimization (DPO). (Rafailov et al., 2023).

- We show that our method using the Llama 2-7B model outperforms existing state-of-the-art methods that rely on larger, proprietary models such as GPT-3.5 and GPT-4 on the Rouge-L metric and are comparable in terms of BERTScore. We also use a series of case studies to illustrate the quality of Socratic questions we generate and that DPO consistently outperforms supervised fine-tuning (SFT).

2 Related Work

2.1 Question Generation in Education

In education, question-generation systems are used to create learning materials and problem sets for quizzes and exams. (Wang et al., 2021) introduces a framework for generating math word problems that incorporates a module for checking the consistency of the word problem generated in terms of the underlying equations that it solves. Our idea of checking the consistency of the synthetically generated samples in data augmentation is inspired by theirs. (Ashok Kumar et al., 2023) proposes a data augmentation and an over-generate and rank method to fine-tune a language model Flan-T5 (Chung et al., 2022) to generate questions for reading comprehension. Their data augmentation method prompts a larger LLM to augment the dataset with valid questions (positive examples) corresponding to a passage in the reading comprehension and then uses this augmented dataset for standard fine-tuning of a smaller open-source LLM. Unlike their work, our data augmentation method involves prompting a larger LLM to generate invalid questions (negative examples) to create a preference dataset that we use for performing preference optimization on a smaller open-source LLM. In computer science education, recent works show the effectiveness of LLMs like OpenAI Codex and GPT-4 (Sarsa et al., 2022; Kumar and Lan, 2024) on generating programming exercise questions, code explanations, and test cases. (Al-Hossami et al., 2024, 2023) introduce a Socratic code debugging dataset, to help a student debug their code along with maximizing the students' learning outcomes. Their experiments with prompting models like GPT-3.5-turbo, and GPT-4 show that these models tend to hallucinate and produce invalid questions. To address this issue, our work builds upon theirs to fine-tune language models to align the generated questions towards ground-truth human preferences and discourage the models from generating invalid questions.

2.2 Reward/ Preference Optimization

Fine-tuning language models to align with human preferences has proven to be beneficial in various natural language processing tasks (Kreutzer et al., 2018; Stiennon et al., 2020; Ziegler et al., 2019; Ouyang et al., 2022). Traditional methods first learn a reward model using a dataset of human preferences and optimize the language model for the downstream task using the rewards obtained from the reward model with reinforcement learning (RL) algorithms such as PPO (Schulman et al., 2017). There are two drawbacks to this method. First, it is hard to obtain a dataset of human preferences

as it is an expensive and sometimes cognitively demanding task. To address this issue, RLAIF procures rewards from an AI system, such as an LLM, and has become a scalable and cheaper alternative (Lee et al., 2023). Second, although preference optimization of LLMs using RL algorithms like PPO is effective, it is significantly more challenging and time-consuming than traditional supervised learning as it involves tuning multiple LLMs and sampling rewards in real time. To address this issue, the DPO method (Rafailov et al., 2023) optimizes a language model to a preference dataset in an RL-free manner by formulating the problem as a binary classification task.

In the domain of education, (Shridhar et al., 2022) proposes a reward-based method to generate Socratic sub-questions to solve math word problems. Similar to our method they define reward characteristics like fluency, granularity, and answerability to prefer sub-questions that have these desired characteristics. They use REINFORCE (Williams, 1992) a popular RL algorithm to optimize their model by sampling rewards from external systems in real time. Our method is different from theirs as we first prompt an LLM to generate invalid Socratic questions (negative examples) to construct a preference dataset. We then use this fixed dataset to tune an open-source LLM in an RL-free method, i.e., using DPO which makes the training more stable and less complex. (Hicke et al., 2023) proposes a DPO-based method for fine-tuning LLama 2 (Touvron et al., 2023) for question-answering on a dataset of Piazza posts for an introductory programming course. They create a proxy preference dataset by using the edit history of Piazza posts by preferring the final versions of answers as opposed to the earlier versions. However, the setting of their work is different from ours as we focus on Socratic question generation and propose a method to create the preference dataset using data augmentation. (Scarlatos et al., 2024) propose a method to perform DPO on LLama 2 for the task of feedback generation to help students solve mathematics word problems. To create preference pairs they prompt LLMs like Codex (Chen et al., 2021) and GPT-3.5 turbo to generate bad feedback and rate the feedback based on a pre-defined rubric using GPT-4. Our problem setting is different from theirs as we focus on the programming education domain and for our task the LLM needs to provide a series of step-by-step feedback in the form of a dialogue-based interaction through Socratic questions instead of just providing the feedback once for a given problem.

3 Problem Definition and Dataset

We study the problem of Socratic question generation in conversations between a *Student* and an *Instructor*, where the Instructor's goal is to guide the Student through the process of solving a problem. Concretely, our goal is to generate Socratic questions at a particular dialogue turn for the instructor during the conversation, given the dialogue history and contextual information about the problem the Student is trying to solve and their solution.

In this work, we use the dataset for code debugging introduced in (Al-Hossami et al., 2024, 2023). The dataset is based on didactic conversations between a Student and an Instructor, where the Student is a novice programmer tasked with writing a program for a given problem. The dataset consists of the Student's buggy code submissions along with a dialogue between the Instructor and the Student, where the Instructor asks Socratic questions in the form of a conversation to help the Student debug their code. The conversation consists of dialogue turns with each Instructor utterance being a collection of several possible "ground-truth" Socratic questions at that dialogue turn. The dataset also contains metadata including the problem statement, the test cases, the bug description, and code fixes to resolve the bug. In total, there are 38 problems with more than 50 different bugs in student solutions, and conversations centered around these buggy codes containing more than 1900 dialogue turns. The dataset is split into two subsets, a train set and a test set which contain 135 and 16 dialogues, respectively, spread across different problems.

4 Proposed method

In this section, we describe our method for the task of Socratic question generation. Our method involves two phases: First, data augmentation, and second, preference optimization, as shown in Figure 1.

4.1 Data Augmentation

Inspired by methods in RLAIF (Lee et al., 2023), we augment the dataset with invalid Socratic questions constructed by prompting GPT-4 (Bubeck et al., 2023), which provides realistic negative samples for LLM-based question generation meth-
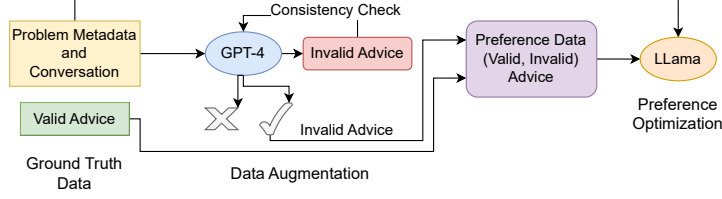
Figure 1: Illustration of our method for LLM-based Socratic question generation, which consists of two phases, data augmentation, and preference optimization.

ods to train on. We follow the method described in (Ashok Kumar et al., 2023) to prompt an LLM to generate synthetic data and employ another instance of the LLM for checking the quality/consistency of the generated synthetic data. Following the definition mentioned in (Al-Hossami et al., 2024), invalid Socratic questions fall into the four following categories:

- **Irrelevant** questions that are not useful for the student, as they shift focus from the actual bug, which may confuse the student.

- **Repeated** questions that have already been asked in previous dialogue turns, which are meaningless to the student.

- **Direct** questions that directly reveal the bug to the student, which do not prompt students to think and may hinder their learning process.

- **Premature** questions which prompt the student to make code fixes before identifying the bug, which may confuse the student.

To generate invalid questions via an LLM, we construct a few-shot prompt that consists of 1) the definition of the categories as mentioned above and 2) an in-context example for each of the invalid question categories detailed above. Our prompt encourages the model to reason using a chain-of-thought method, by first generating the "reasoning process/logic" behind an invalid question, followed by the question (Wei et al., 2022). We generate invalid questions corresponding to all four categories at every dialogue turn where the ground truth is provided.

Following (Ashok Kumar et al., 2023; Wang et al., 2021), we use a consistency checking step where we prompt GPT-4 to check the consistency of the generated questions to filter out inconsistent questions from the augmented dataset. Inconsistent questions are those that do not belong to any of the

invalid categories listed above. We pose the consistency checking step as a classification task where GPT-4 predicts a label for each generated question over six categories, including the four invalid categories and two additional categories: "good" and "incorrect". Good questions are acceptable Socratic questions at that particular dialogue turn and cannot be used as negative samples. Incorrect questions are unrelated to the problem and the dialogue itself and are often erroneous due to LLM hallucination, which provides little value as easy-to-tell negative samples. To maintain high data quality of our preference dataset, we discard all samples that are predicted as "good" or "incorrect", to get the final set of synthetically generated invalid questions.

Finally, we construct a preference dataset consisting of 2500 tuples of valid and invalid Socratic questions. In the preference pairs, valid questions are taken from the ground truth questions in the original dataset, while the invalid questions are generated synthetically as described above. Each valid question from the original dataset is paired with every synthetically generated invalid question of all categories to form the augmented dataset.

4.2 Preference Optimization

In this step, we fine-tune an open-source LLM, Llama 2 (Touvron et al., 2023) for Socratic question generation using DPO (Rafailov et al., 2023). The first step is to perform SFT, i.e., we use the original dataset, D, as is to fine-tune LLama 2 for Socratic question generation. For a given conversation in the train set, we first split the dialogue into constituent dialogue turns. The input to LLama 2 is a prompt (p) that consists of a systems message that instructs the LLM to generate a Socratic question, the problem metadata, and the current dialogue history (between the Student and the Instructor). The output is the valid Socratic question (q_v) corresponding to that dialogue turn in the dataset. In the cases where multiple Socratic questions were

given for a dialogue turn, we treat each one as a different output associated with the same input for fine-tuning LLama 2. As shown in Equation 1, the simple SFT step learns a reference policy π_{ref} by minimizing the loss $\mathcal{L}_{SFT}$, which serves as the starting point for preference optimization.

The second step is to perform preference optimization where we fine-tune Llama 2 on the preference dataset, D_P, that we obtain from the data augmentation phase, using the same prompt, p, as input that was used for SFT, but with two outputs: the valid question q_{v} and the invalid question q_{iv}, for that dialogue turn. As shown in Equation 2, this preference optimization step learns a human preference-aligned policy π_θ, given the reference policy π_{ref} obtained from Equation 1, by formulating the task as a binary classification task, minimizing the negative log-likelihood loss $\mathcal{L}_{DPO}$, where σ is the Sigmoid function. This minimization leads to learning π_θ, by increasing the likelihood of the valid question and decreasing the likelihood of the invalid question while remaining close to the reference policy π_{ref} which is governed by the hyperparameter β. Here θ is the parameters of the preference-aligned policy which is simply the parameters of the neural network, in our case LLama 2.

$$\mathcal{L}_{\text{SFT}}(\pi_{\text{ref}}) = -\mathbb{E}_{(q_{\text{v}},p)\sim D}[\log \pi_{\text{ref}}(q_{\text{v}}|p)] \quad (1)$$

$$\mathcal{L}_{DPO}(\pi_\theta; \pi_{\text{ref}}) =$$
$$- \mathbb{E}_{(q_{\text{v}},q_{\text{iv}},p)\sim D_P}\left[\log \sigma(\beta \log \frac{\pi_\theta(q_{\text{v}}|p)}{\pi_{\text{ref}}(q_{\text{v}}|p)} \right.$$
$$\left. - \beta \log \frac{\pi_\theta(q_{\text{iv}}|p)}{\pi_{\text{ref}}(q_{\text{iv}}|p)}) \right] \quad (2)$$

5 Experimental Settings

In this section, we detail the implementation setup, methods compared, and metrics used to evaluate our Socratic question generation method.

Implementation details. In the data augmentation phase, we query OpenAI's[2] GPT-4 using a rate-based API. We set the temperature of the GPT-4 model to 0.5 to encourage moderate randomness in the outputs. For the consistency checking GPT-4 model, we use a temperate of 0 to maintain determinism. In the preference optimization phase, we

use Code-Llama (7B) (Roziere et al., 2023) pre-trained for instruction following tasks, particularly on code data[3]. We load our Code-Llama model in an 8-bit configuration and train using QLora (Dettmers et al., 2023) with the *peft*[4] HuggingFace library to facilitate efficient fine-tuning. For the SFT step, we fine-tune the model for 5 epochs with a learning rate of 3e-5, and a batch size of 2 by accumulating gradients for creating a virtual batch size of 64 which takes about 10 hours to train on a single Nvidia A6000 GPU. For the DPO step, we fine-tune the model for 1 epoch with a learning rate of 3e-5 and a β (which denotes the KL-loss (Joyce, 2011) between the preference policy learned and the reference SFT policy) of 0.1, with a batch size of 2, which takes about 6 hours to train. For the DPO experiments, we carry out a grid search using hyperparameters learning rate as 1e-5, and 3e-5, β of 0.1, and 0.5 and number of epochs as 1 and 2 to arrive at the best-performing hyperparameters as mentioned above.

Methods. As baselines, we perform zero-shot prompting of the LLama 2 Chat model[5] (Touvron et al., 2023), denoted by **LLama**, to generate all possible Socratic questions for the current conversation turn. We also prompt LLama 2 in a chain-of-thought (Wei et al., 2022) manner to first generate the current student misconceptions and then generate the Socratic questions, denoted by **LLama (CoT)**.

To decode our trained (SFT and DPO) LLM, we use two decoding techniques, greedy and nucleus sampling, with a p value of 0.9 temperature of 1, and a number of return sequences of 5. We refer to these methods coupled with the trained SFT method as **SFT Greedy**, **SFT Sample-5**, and similarly for the DPO methods. Greedy decoding takes 30 minutes to complete, whereas Sample-5 takes an hour.

Metrics. To measure the similarity between the generated Socratic questions and the ground truth questions, we use two commonly used evaluation metrics in natural language generation tasks: **BERTScore** (Zhang* et al., 2020) based on the DeBERTa language model (He et al., 2021), which measures the semantic similarity, and **Rouge-L** (Lin, 2004), which measures n-gram overlap based

Table 1: Performance comparison across different methods. All GPT baseline results are reported in (Al-Hossami et al., 2024). Boldface represents the highest value/s for that column.

Method	Rouge-L			BERTScore		
	P	R	F1	P	R	F1
GPT-3.5	21.0	14.3	17.0	56.0	43.5	**48.9**
GPT-3.5 (CoT)	20.3	9.7	12.0	61.7	35.8	41.6
GPT-4	14.1	23.3	17.6	35.4	62.6	45.2
GPT-4 (CoT)	5.2	26.6	8.1	12.6	**64.8**	19.5
LLama	12.8	18.6	13.2	36.0	48.3	35.9
LLama (CoT)	13.7	15.5	13.2	42.3	49.0	41.0
SFT Greedy	29.7	13.4	17.2	61.8	29.3	36.8
DPO Greedy	**30.6**	13.3	17.1	**65.9**	32.7	40.3
SFT Sample-5	14.1	26.0	17.1	32.1	62.9	41.1
DPO Sample-5	15.1	**27.9**	**18.3**	34.8	**64.3**	42.0

on the longest common subsequence (LCS). In addition, the dataset we use (Al-Hossami et al., 2024, 2023) provides multiple ground truth Socratic questions at each dialogue turn. To measure the similarity between a set of m LLM-generated questions with a set of n ground truth questions, we adopt the process used in (Al-Hossami et al., 2024), which uses Edmond Blossom algorithm (Galil, 1986) to find the maximum matching in a complete bipartite graph between the two sets with a total of mn edges, where the weight of each edge is computed using one of the metrics mentioned above. This step guarantees that every ground-truth question corresponds to, at most, one LLM-generated question, inhibiting semantically equivalent LLM generations from artificially inflating the metric scores. The number of True Positives (TP) is the total sum of the weights of all edges in the optimal matching. False Positives (FP) are calculated by summing the difference between every weight of an edge in the matching with 1. Any unmatched LLM-generated question counts 1 towards False Positive. Similarly, any unmatched ground truth question counts 1 towards False Negative (FN). The TP, FP, and FN values are used to compute the precision, recall, and F1 score for a particular metric. The metric penalizes over-generated LLM questions that do not match with any ground truth questions by classifying them as an FP, thus decreasing the precision.

6 Results and Discussions

In the consistency checking step of the data augmentation phase, we see that 72% of the generated questions are considered for the preference dataset creation as 27% of the generated questions are classified as "good" and 1% as "incorrect". This result shows that GPT-4 is more prone to generate "good"

questions for particular dialogue turns than incorrect questions that do not relate to the problem and the dialogue.

For the task of Socratic question generation, Table 1 shows the comparison between different methods on the metrics defined for our task. All the GPT-3.5 and GPT-4 results are taken from prior work (Al-Hossami et al., 2024). We observe that GPT-4 (CoT) has the highest recall and yet the lowest F1 score. This observation is because, GPT-4 generates a large number of Socratic questions a few of which are similar to the ground truth questions, however, a significant fraction of the generated questions do not correspond to any ground truth questions, hence being labeled as false positive, thus decreasing the precision. (Al-Hossami et al., 2024) also carry out manual analysis to show that GPT (CoT) outputs are the best despite having low F1 scores. This observation can be attributed to the fact that GPT (CoT) has the highest recall among all other GPT methods and hence better corresponds to the ground truth questions.

For the baseline methods that use zero-shot LLama prompting, we observe that LLama (CoT) is the best, which shows that chain-of-thought prompting to first generate the students' current misconceptions followed by the Socratic questions is effective. Among the preference optimization experiments, we see that DPO consistently outperforms SFT. We also observe that the LLama (CoT) performs as well as DPO Greedy in terms of BERTScore F1 as LLama (CoT) generates a higher number of Socratic questions whereas the DPO Greedy method just generates one. Hence, the recall of the DPO Greedy method is lower than that of LLama (CoT). Among decoding variants, we see that the Sample-5 method is better than the Greedy method highlighting the importance of sampling multiple possible Socratic questions instead of just one.

Overall, we see that our preference-optimized models with DPO give the best Rouge-L scores for all precision, recall, and F1 scores with DPO Greedy having the highest precision and DPO Sample-5 having the highest recall and F1 score among all the methods. DPO Greedy has the highest BERTScore precision, whereas DPO Sample-5 has a recall comparable to the best GPT method, GPT-4 (CoT). These results suggest that the DPO-optimized LLama 2-7B model is better than (or as effective as) much larger models like GPT-4 (25 times larger) for Socratic question generation.

Table 2: An example of invalid Socratic questions generated from GPT-4 for a given conversation, which we use to augment the dataset.

Problem	Write a function "top_k(lst: List[int], k: int) -> List[int]" that returns the top k largest elements in the list. You can assume that k is always smaller than the length of the list. Example Case: top_k([1, 2, 3, 4, 5], 3) => [5, 4, 3]; top_k([-1, -2, -3, -4, -5], 3) => [-1, -2, -3]
Bug Description	The function removes the element at index 'max(lst)' instead of removing an element equal to 'max(lst)'. Consequently, the function throws an IndexError on line 5 when a removed value in 'lst' is greater than the length of 'lst'.
Bug Fixes	On line 5, replace 'lst.pop(max(lst))' with 'lst.remove(max(lst))'
Conversation	**Student**: Hi. I am confused. My code doesn't seem to work. Can you help? **Instructor**: Hello. Sure, let's see. Do you know what might be the issue? **Student**: I think the problem is with the '.pop()' method. It seems to have issues with indexing.
Ground Truth	1. Ok, no worries. Let's review your code line by line. Could you please explain it to me? 2. Let's start with a simple example. What is the output of the following code snippet: 'top_k([1, 2, 3, 4, 5], 3)'? 3. Could you please explain what line 5 in your code does? 4. Let's look into the Python documentation. Can you describe what the '.pop()' method does?
Invalid Generated Questions	**Irrelevant**: What happens if you enter an empty list as the input? **Repeated**: Do you know what might be the issue? **Direct**: Are you sure you should be using the pop() method to remove the maximum element from the list? **Premature**: Have you considered using the remove() method instead of pop()?

7 Case Study

We now use a case study to illustrate why our method leads to better Socratic question generation. First, we show an example of invalid Socratic questions generated by our data augmentation phase. Second, we compare different methods for Socratic question generation.

Table 2 shows an example of the augmented data, i.e., invalid questions generated by GPT-4 for an example problem, which asks students to write code to return the largest k elements in a list. The student's code (Table 4 Code 1) incorrectly removes elements at index `max(lst)` as opposed to removing elements equal to `max(lst)`, thereby causing an `IndexError`. The potential fix to the code is to replace the `.pop()` function with `.remove()`. In the conversation, we see that the student knows the problem lies in their use of `.pop()`. The ground truth Socratic questions for this dialogue turn are highly generic, asking the student to review the code line by line, apply an example test case, or do further reading on Python documentation. We see that the four types of invalid questions generated by GPT-4 are: the *irrelevant* question is out of context and does not help the student understand the bug in their code. The *repeated* question has already been mentioned by the instructor. The *direct* questions reveal the problematic function `.pop()` and do not give room for the students to discover the problem themselves. The *premature* question directly

suggests a code change to replace the `.pop()` with `.remove()` function even before the student has realized the actual bug. These diverse examples of invalid questions serve as good training data to let an LLM know what kinds of invalid questions it should avoid generating.

Table 3 shows the questions generated by different methods for an example problem that asks students to write code that returns the factorial of a number. The student's code (Table 4 Code 2) has some indexing errors with the `range` function, resulting in the `fact` variable being multiplied by 0 when the loop starts, hence resulting in the output of 0 no matter what the input is. In the conversation, we see that the ground truth questions encourage the student to debug the code by printing the value of lines 5 and 6 to examine the variables `fact` and `i` along with the role of `range(n)`. The LLama output is very verbose and directly reveals that the problem is in the updation of the 'fact' variable. The LLama (CoT) output is very vague as it does not refer to the exact line of the code. The SFT Greedy output correctly asks the student the value of `i` but does not provide more details, such as the value of `n`. The DPO Greedy method is more helpful since it asks the student to check the value of `fact` specifically for n=1. The first and fifth questions by SFT Sample-5 are invalid and directly ask the value of `fact` when `i` is 0, which directly reveals the problem that `fact` is always 0. The third

Table 3: Socratic questions generated by different methods.

Problem	Write a function 'factorial(n:int) -> int' that computes the factorial n! of a natural number n, which is defined mathematically as: $0! = 1$ and $n! = n \times (n-1)!$. Additionally, if the input integer n is negative the function should return 0. Example Cases: factorial(-1) => 0; factorial(0) => 1; factorial(1) => 1; factorial(5) => 120
Bug Description and Fix	On line 6, 'fact' is first multiplied with 0 in the first iteration of the for loop and hence remains 0 throughout the loop. To fix the bug, replace 'i' with '(i + 1)' in line 6 and replace 'range(n)' with 'range(1, n + 1)' in line 5.
Conversation	**Student**: Hi! I implemented the factorial function but it doesn't work and I do not know why. Can you help? **Instructor**: Sure. Can you tell me for what values of n it fails and what values it returns in those cases? **Student**: For n = 1 or larger it returns the same value, 0.
Ground Truth	1. Let's see what happens when 'n' is 1. What is the first value that is assigned to variable 'i' in line 5? 2. Let's see what happens when 'n' is 1. Before line 6 is evaluated in the first iteration of the for loop, what are the values of the variables 'fact' and 'i'? 3. Let's see what happens when 'n' is 1. Can you insert a new line between lines 5 and 6 that prints the values of the variables 'fact' and 'i'?
LLama	So, it seems that the issue is that the value of 'fact' is not being updated correctly in the loop. Can you tell me why do you think 'fact' is not being updated?
LLama (CoT)	Can you explain why you think the bug is in that specific line?
SFT Greedy	What is the value of 'i' in the first iteration of the for loop?
DPO Greedy	Okay. Let's start with 'n = 1'. What do you think 'fact' is on line 6?
SFT Sample-5	1. What is the value of 'fact' at line 6 when 'i' is 0? 2. Let's take a closer look at the line where 'fact' is multiplied. Could you explain the steps you take to compute that multiplication? 3. 0 is not a factorial, right? What do you think the first step should be? 4. What does this tell us about the implementation? 5. Are you multiplying the initial 'fact' value of 1 by 'i' at each step?
DPO Sample-5	1. Alright, so for any value of 'n' that we pass to the function, what is the output of line 6? 2. Can you tell me what value of 'i' is returned in line 6 when 'n = 1'? 3. Okay. Let's look at 'factorial(1)' and 'factorial(2)'. What are the values of 'i' and 'fact' when it goes into the for loop in those cases? 4. Okay, let's go line by line. Let's start from line 5. What do we see? 5. Okay. Can you tell me the steps for computing 'factorial(n)' where 'n' equals 1?

Table 4: Buggy codes corresponding to the case studies

```
Code 1:

def top_k(lst, k):
 result = []
 for i in range(k):
  result.append(max(lst))
  lst.pop(max(lst))
 return result
```

```
Code 2:

def factorial(n):
 if n < 0:
  return 0
 fact = 1
 for i in range(n):
  fact = fact * i
 return fact
```

and fourth outputs are either irrelevant or repeated. The second question, which asks the student to examine the value of `fact` is valid since it does not directly reveal the bug. In contrast, most of the DPO Sample-5 questions are valid, since they urge the student to examine the value of `i` and `fact` on lines 5 and 6 with specific values of n, without directly revealing the bug that `i` is always 0. Through these comparisons, we see that DPO improves Socratic question generation compared to SFT and that DPO Sample-5 is highly capable of generating valid yet diverse questions.

8 Conclusions and Future Work

In this work, we propose a method for Socratic question generation in programming problem feedback scenarios. Our method consists of a data augmentation phase to create a preference dataset by synthetically generating invalid questions according to four possible categories. We then use this preference dataset to fine-tune an open-source LLM, LLama 2-7B, using direct preference optimization (DPO). Our results show that the preference-optimized LLama 2-7B model often outperforms existing state-of-the-art prompting methods (on common text similarity metrics) that rely on much larger GPT models (25 times larger), by avoiding invalid questions after training on the augmented dataset. Our method paves the way toward an open-source, accessible, cheaper, privacy-preserving, yet effective alternative to generating Socratic questions which can improve students' learning outcomes without having to rely on proprietary rate-based API-accessed models like GPT-4. There are several avenues for future work. First, we can develop a technique to differentiate types of invalid Socratic questions and not treat them

equally while performing preference optimization. This technique would require us to modify the inherent objective function of DPO to incorporate more than one unpreferred question for a single preferred question, which may give us fine-grained control over the LLM generations. Second, we can experiment with open-source LLMs that are larger than 7B to see whether DPO provides more significant gains over SFT on larger models on the Socratic question generation task. Third, we can perform a systematic human evaluation to compare the performance of our proposed method with other baselines. Also, we can focus on designing an automatic metric (based on LLM prompting (Liu et al., 2023)) other than Rouge and BERTScore which captures the helpfulness of the Socratic questions without heavily relying on assigning higher scores only to questions that have high lexical overlap with the ground-truth questions. Fourth, we can experiment with alternative preference optimization methods, such as KTO (Ethayarajh et al., 2024) which do not need explicit preference data in the form of pairs of valid and invalid questions. Fifth, we can also explore if Socratic question generation helps in improving other tasks in computer science education like test case generation (Kumar and Lan, 2024) by posing the problem as answering several Socratic sub-questions (Shridhar et al., 2022). Finally, we can also explore how to make Socratic question generation knowledge-aware, i.e., generating different questions for students with different knowledge states, which can be estimated using the open-ended knowledge tracing method for computer science education (Liu et al., 2022).

9   Limitations

Our work proposes a method for preference optimizing open-source LLMs like LLama 2 for the task of Socratic question generation for student code debugging. We use only LLama 2 as the base model for carrying out preference optimization, and not other open-source models like Mistral (Jiang et al., 2023). Since our main contribution is the data augmentation and preference optimization method, we use only one of the best models open-source models (LLama 2) to show that our method outperforms state-of-the-art models like GPT-4. Future work can also explore the performance of different open-source models using a variety of optimization methods including our data augmentation and preference optimization method

for Socratic question generation. Also, we do not formally analyze any biases that exist in the generated augmenting data or the generated Socratic questions. Future work can focus on measuring such biases to make our methods that use these LLMs more inclusive for all students belonging to different demographics.

10   Ethics Policy

Since our invalid questions are generated using an LLM potential linguistic or cultural bias related to the pre-training of the LLM might be reflected. However, we hypothesize that this bias would be minimal as Socratic questions are goal-driven, concise, and framed in the second-person perspective directed toward the student. Our work focuses on open-source LLMs like LLama for Socratic question generation as compared to rate-based API-accessed models like GPT-4 (which is used only once during data augmentation) which implies that our methods are privacy-preserving and there is minimal chance of leakage of students' confidential data. However, training LLMs like LLama on GPUs like A100 for 10 hours results in the emission of CO_2 which might not be environmentally friendly.

11   Acknowledgments

References

Erfan Al-Hossami, Razvan Bunescu, Justin Smith, and Ryan Teehan. 2024. Can language models employ the socratic method? experiments with code debugging. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2024, page 53–59, New York, NY, USA. Association for Computing Machinery.

Erfan Al-Hossami, Razvan Bunescu, Ryan Teehan, Laurel Powell, Khyati Mahajan, and Mohsen Dorodchi. 2023. Socratic questioning of novice debuggers: A benchmark dataset and preliminary evaluations. In *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, pages 709–726, Toronto, Canada. Association for Computational Linguistics.

Nischal Ashok Kumar, Nigel Fernandez, Zichao Wang, and Andrew Lan. 2023. Improving reading comprehension question generation with data augmentation

and overgenerate-and-rank. In *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, pages 247–259, Toronto, Canada. Association for Computational Linguistics.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

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. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Yuhao Dan, Zhikai Lei, Yiyang Gu, Yong Li, Jianghao Yin, Jiaju Lin, Linhao Ye, Zhiyan Tie, Yougen Zhou, Yilei Wang, et al. 2023. Educhat: A large-scale language model-based chatbot system for intelligent education. *arXiv preprint arXiv:2308.02773*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.

Zvi Galil. 1986. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys (CSUR)*, 18(1):23–38.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

Yann Hicke, Anmol Agarwal, Qianou Ma, and Paul Denny. 2023. Chata: Towards an intelligent question-answer teaching assistant using open-source llms. *arXiv preprint arXiv:2311.02775*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

James M Joyce. 2011. Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer.

Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Z Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. *arXiv preprint arXiv:2401.11314*.

Julia Kreutzer, Joshua Uyheng, and Stefan Riezler. 2018. Reliability and learnability of human bandit feedback for sequence-to-sequence reinforcement learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788, Melbourne, Australia. Association for Computational Linguistics.

Nischal Ashok Kumar and Andrew Lan. 2024. Using large language models for student-code guided test case generation in computer science education. *arXiv preprint arXiv:2402.07081*.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi. 2023. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Naiming Liu, Zichao Wang, Richard Baraniuk, and Andrew Lan. 2022. Open-ended knowledge tracing for computer science education. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3849–3862.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

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.

Richard Paul and Linda Elder. 2007. Critical thinking: The art of socratic questioning. *Journal of developmental education*, 31(1):36.

Chris Quintana, Brian J Reiser, Elizabeth A Davis, Joseph Krajcik, Eric Fretz, Ravit Golan Duncan, Eleni Kyza, Daniel Edelson, and Elliot Soloway. 2018. A scaffolding design framework for software to support science inquiry. In *Scaffolding*, pages 337–386. Psychology Press.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh*

Conference on Neural Information Processing Systems.

Baptiste Roziere, 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*.

Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 27–43.

Alexander Scarlatos, Digory Smith, Simon Woodhead, and Andrew Lan. 2024. Improving the validity of automatically generated feedback via reinforcement learning. *arXiv preprint arXiv:2403.01304*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Kumar Shridhar, Jakub Macina, Mennatallah El-Assady, Tanmay Sinha, Manu Kapur, and Mrinmaya Sachan. 2022. Automatic generation of socratic subquestions for teaching math word problems. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4136–4149, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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.

Henansh Tanwar, Kunal Shrivastva, Rahul Singh, and Dhruv Kumar. 2024. Opinebot: Class feedback reimagined using a conversational llm. *arXiv preprint arXiv:2401.15589*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Zichao Wang, Andrew Lan, and Richard Baraniuk. 2021. Math word problem generation with mathematical consistency and problem context constraints. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5986–5999, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zichao Wang, Andrew S Lan, Weili Nie, Andrew E Waters, Phillip J Grimaldi, and Richard G Baraniuk. 2018. Qg-net: a data-driven question generation model for educational content. In *Proceedings of the fifth annual ACM conference on learning at scale*, pages 1–10.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

David Wood, Jerome S Bruner, and Gail Ross. 1976. The role of tutoring in problem solving. *Journal of child psychology and psychiatry*, 17(2):89–100.

Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.