LLM Ability to Answer University Student Questions in CS Trained Exclusively on Auto-Generated QA

Anonymous ACL submission

Abstract

With the advent of generative AI as commonplace to daily-life within the past couple of years, there is a strong need for extensive research as we apply this technology to educational contexts. This study supports that body of research as we explore two driving questions: **(1) Can we use LLMs to create synthetic student-like question-answer datasets? and (2) Can we train an LLM to embody an instructor in answering real student questions?** In this paper, we explore both of these questions, grounded by prior works and approaches. Ultimately, our findings suggest that synthetic QA data can be generated, but still requires significant improvement to aptly represent the range of questions asked by real students. Additionally, while LLMs can be trained to give reasonable answers, the generated responses often struggle with alignment to instructional intent and semantic accuracy, requiring further fine-tuning and advanced evaluation frameworks.

1 Introduction

The release of ChatGPT in November 2022 brought with it a new wave of AI and the potential of its applications in all aspects of modern life. It showcased to the public the power of LLMs, and their potential to reduce human workloads. Based on its extensive and ever growing training data, ChatGPT and other generative AI models can write papers, read and understand code, source information across the internet to answer niche questions, and so much more. Companies tend to hold a shared view of AI as a way to optimize their workforce, opening greater potential for cost savings. In the realm of software development, coders can use LLMs to optimize code, debug, write pseudocode and boilerplate functions, etc., potentially streamlining and simplifying the development process.

Depending on the context, these uses are immensely powerful and time-saving, but with unknown effects and unregulated usage also carry strong potential for negative ramifications. The integration of this technology into educational contexts is naturally a lot more careful and cautious. Instead of accepting AI in full force, education also has to consider problems with students cheating and AI hallucinating as potential threats to learning. Unfortunately, given the opportunity, many students will have the tendency to offload portions of their work to generative models, causing them to miss out on critical learning. Enforcing learning in a world where this technology is readily, openly available is a difficult question, and one that has inspired innumerable research projects in the past couple of years. Instructors are seeking how to safely integrate generative technologies into their curriculum while also GPT-proofing assignments, bringing exams back to being written in-person, and taking a number of other measures to ensure student learning. Ultimately, we see gen-AI as an inevitability; we must turn our focus to improving/structuring interactions such that it can enhance the learning process.

As we will touch on in our related works, the accessibility of 1-1 teaching (ie. instructors and course staff) is a weakness of conventional teaching environments. As students aren't able to ask questions of their teachers, they turn to question-answering technologies as an alternative. Unfortunately, AI is often not trained to fully understand the context/assignment students are asking about and often yields incorrect or incomplete responses. To some end, this can actually also result in students garnering incorrect understandings based on their interactions with hallucinative/un-informed AI systems. The accessibility issue is an ongoing one, even prior to the advent of generative AI, and brings up an important question: can we fine-tune/train generative AI systems to act like instructors when they aren't available? Specifically, can we train them to understand the requirements of specific assignments/projects such that they can provide useful feedback without also giving away answers to students? Finally, to what extent is prior data (rather than synthetic data) necessary to train such a system to sensibly answer real student questions?

Answering these questions is complex. In this paper, we explore how well modern general-purpose fine-tuned LLMs can generate synthetic QA data; how well can they generate the kinds of questions we expect from students with regard to a project specification? Next, we compare the use of this synthetic data with real student QA to train a model to answer student questions how instructors would. To contextualize the importance of the synthetic data, we acknowledge that new courses have non-existent prior student data, and even existing courses may have gaps in datasets. So, we will examine both the process of generating synthetic QA data and using that data to inform a model in answering student questions.

2 Related Works

There are a number of related, but fundamentally different works that inspired our pursuit of this project. To contextualize these works, we note that the progress of LLMs in the past five years has been especially significant. Integrating these technologies into educational contexts holds much promise, but also has to be done carefully. Educational research has been around for a long time; a hallmark of this space comes from **Bloom's 2 Sigma Challenge**, published in 1984 (BLOOM, 1984).

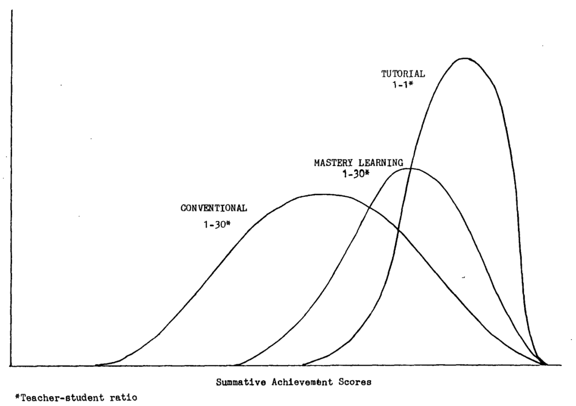

Figure 1: Curves comparing scores for students in varied learning environments.

Figure 1, pulled from this study, highlights the differences between conventional (1-30) teaching environments, personalized (1-1) tutoring settings, and enhanced (1-30) ***"mastery learning"*** settings. As expected, average student performance is the highest in 1-1 tutoring settings; while such teaching settings are most desirable, they are also highly impractical and expensive. Mastery learning serves as a generalization for settings in which students receive a sense of personalized learning, often aided by technology (ie. cognitive tutors) in an otherwise conventional (1-30) teaching setting. There are a number of technologies that have been developed in support of mastery learning, and in our paper we discuss those related to answering student questions through formal course forums (ie. Piazza).

In our paper, we turn to Question-Answer generative systems. While round-the-clock instructor QA is the most desirable scenario, this is again far too expensive to be practical. Instead, we aim to observe to what extent generative language models can support course instructors in correctly answering student questions. While some courses are likely to have extensive prior data (from past semesters), we further seek to answer the question if **synthetic data**, QA generated by an LLM based on extracting data from an assignment specification, would provide sufficient training for such a model?

Prior work has explored QA generation in a number of ways. First Basu et al. aims to reduce workload for instructors by generating multiple choice questions from text inputs. Next, Riza et al. looks at generating reading comprehension based short-answer questions via KNN techniques. Finally, Virani et al. looks at how QA generation systems can be designed to support generation of a variety of question types. Ultimately, our work differentiates itself in two ways. First, we narrow the problem scope to the context of Computer Science education as we will seek to analyze synthesis of questions from coding project specifications; can we train an LLM to generate a range of sensible questions for a CS assignment based on its specification? Secondly, we want to look at using generated QA as synthetic training data for a final student-question answering LLM; can synthetic QA train a system to answer real student questions with high accuracy? Ultimately, the prior work we have touched on so far highlights how our project fits into the broader space of QA generation for educational purposes.

Another interestingly related work is the CodeAid system produced by Kazemitabaar et al. which offers as a custom "coding assistant" as a direct alternative to larger-purpose LLMs (ie. chatGPT) that helps students learn to code with strictly no-code responses. Their approach focuses more on few-shot learning in prompt engineering for OpenAI API calls rather than training/fine-tuning an LLM, and focuses more on the usefulness of an LLM answering coding-specific questions (ie. fixing, writing, and understanding code). In contrast, our project aims to build two models working in tandem: one that can generate synthetic student QA data, and another that can answer hyper-specific project/assignment-based questions. While this work provides a vastly different approach, it does address a problem similar to the broader one (around-the-clock support for student questions) we are exploring in this paper.

Next, we will turn to some of the research that informed our technical approach to this project. Majority of the works we've mentioned utilize a T5 model as a baseline encoder-decoder tool; we do the same in building our LMs (Raffel et al., 2023). `SQuAD` (Stanford Question Answering Dataset) is a very popular QA dataset used across works in the space, and consists of a combination of answerable and unanswerable questions (Rajpurkar et al., 2018). For example, one of the T5 models we use to generate our synthetic QA data has been fine-tuned extensively on `SQuAD` (Manakul et al., 2023). The other model we use fine-tunes on `SQuAD` as well as `CoQA` and `MSMARCO`, two other large general-purpose QA datasets (Reddy et al., 2019; Bajaj et al., 2018). The `SQuAD` dataset is sourced from Wikipedia articles, `CoQA` has a focus on conversational QA, and `MSMARCO` is sourced from questions asked on Bing, such that the data from each of these datasets is expectedly quite different from the types of questions CS students may be asking about project specifications. In an ideal world, we would be able to use a model that has been fine-tuned on coding-related questions. Regardless the SQuAD (and other large dataset) fine-tuning aids the baseline T5 models in generating more sensible synthetic QA.

A significant problem in the realm of generative AI is model hallucination. Specifically in the QA space, large LLMs have been trained on *a lot* of information, and have the tendency to carry answers beyond the scope of the question being asked. Simultaneously, fine-tuning a model to concepts within a limited domain is a challenging, resource-intensive process. Retrieval Augmented Generation (RAG) is a popular technique for combating these issues by basing model responses in a set knowledge base (Meyur et al., 2024; Barron et al., 2024). In our final question answering model we attempt a model configuration that emphasizes RAG techniques, utilizing a combination of synthetic and existing CS QA data to ground the model.

For evaluation, we utilized a combination of metrics to capture different aspects of quality in natural language generation. BERTScore (Zhang et al., 2020) assesses semantic similarity using contextual embeddings from pre-trained transformers, offering a nuanced understanding beyond token-level overlap. BLEU (Post, 2018), a standard metric for machine translation, measures n-gram overlap to evaluate lexical precision, while METEOR (Banerjee and Lavie, 2005) accounts for linguistic variations such as stemming and synonymy, aligning more closely with human judgment. ROUGE-L (Lin, 2004) evaluates the longest common subsequence between generated and reference texts, emphasizing fluency and recall. Together, these metrics provide a comprehensive evaluation framework, balancing semantic, lexical, and structural quality.

3 Implementation

As we have introduced, our project seeks to answer two key research questions:

1. Can synthetically made datasets effectively train LLMs?

2. Can an LLM be trained to answer real student questions like an instructor?

In order to answer these questions, we devised a research project that uses two LLMs. The first LLM takes in a project or assignment specification and outputs a set of corresponding question and answer pairs. The second LLM utilizes the synthetic data (question-answer pairs) generated by the first LLM as training data to ultimately take real student questions as input and output instructor-like responses. Throughout our report, we will divide analysis into two, corresponding to each of these two LLMs.

3.1 LLM #1

Curating the first LLM had three core steps:

1. Generate Question Contexts

2. Generate Question/Answer (QA) Pairs

3. Evaluate the Question/Answer (QA) Pairs

3.1.1 Input Data

We selected the University of Michigan EECS 280 Project 3 spec as our input for this LLM due to our access to real student Piazza question-answer data. Additionally, our familiarity with the project enabled a level of manual analysis of the quality of generated questions and answers.

3.1.2 Generate Question Contexts

Upon first examination of the EECS 280 P3 (Euchre) specification, we had to consider how best to consolidate the combination of code snippets, images, tables, and raw textual paragraphs into a format that would be readable by a model. While the visual aspects of a specification are certainly important and insightful, we opted to extract just the text from the project assignments. Future work might attempt to better integrate different information formats into model inputs.

After paring specifications down to just the text, we moved on to consideration of how much information to give the model at a time. A **question context** is a snippet of a larger project specification or assignment. The pre-existing models we use in this project require contexts as input to generate QA pairs. Further, context sizes are limited; we cannot pass an entire specification into a model and ask it to generate questions (this is too large), nor would we want to if we could (questions would likely be too generic). Instead, we have to feed the model chunks at a time and ask it to generate QA pairs for each context.

Our initial approach to this problem was splitting an assignment by sentences and/or paragraphs. Intuitively, this would be a natural way to break apart text, and would ensure that each context passed to the model was as sensible as possible. Unfortunately, this approach introduced several complexities. Sometimes ideas are split between sentences and/or paragraphs wherein creating a good context for good, fully-informed questions would require including multiple sentences in a single context. However, the presence of long sentences, sentence groupings or paragraphs introduces the issue of how they should be split when they are too long. Should sentences be split in half? Should splits be included with the previous or next context? Should long sentences be moved to their own context altogether? Unfortunately, it doesn't feel as though there is a singular correct answer to this question; the answer is context dependent and we would have to opt for a solution that is good enough for the general case. Again, future work might seek to use a more sophisticated approach for context creation; deeming which sentences should be grouped together, and which could be separated, whilst considering the size limitations of a context.

Instead, we opted for a slightly different approach: splitting our specification with a **sliding window algorithm**. We converted our specification into an array of words and used a fixed window and fixed sliding interval. Starting from the beginning of the assignment, we insert the first <window_size> words into a single context. Then, the window shifts by the sliding interval to generate a new context, and so on through the remainder of the document. Figure 2 below gives a visual representation of this algorithm. The downfall of this approach is the lack of intentionality in ensuring particular words/sentences/paragraphs stay together. But, the upsides are that we don't have to try to decide which window/context a given set of words is included with, as all text gets included in multiple contexts. This also provides us with a fixed/set context size, and addresses all the ambiguities/complexities we discussed in a sentence/-paragraph splitting approach.

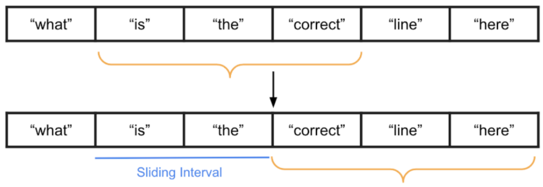

Figure 2: Visualization of sliding window algorithm concept.

3.1.3 Generate Question/Answer (QA) Pairs

For generating our QA pairs, we tested two models identified in prior works:

- `potsawee/t5-large-generation-squad-QuestionAnswer`
- `iarfmoose/t5-base-question-generator`

The first model we tested was `potsawee/t5-large-generation-squad-QuestionAnswer`. The input and output format for this model is included below. Note: <sep> serves as a separator token.

Input: context
Output: question <sep> answer

We did most of our parameter fine-tuning on this model; we discuss our results more extensively in the Evaluation section of the report, but we used this model to identify the best combination of hyper-parameters which we then carried on when evaluating the second model.

The second model evaluated was the iarfmoose/t5-base-question-generator. Again, the input and output format for this model is described below.

Input: <answer> answer <context> context
Output: questions

The difference in input and output formats between these two models is quite significant. Namely, the first model only requires a context and actually produces a QA pair while the second model requires a context and answer and aims to produce a corresponding question. For the second model, its documentation specifies that for short answer questions the context could also be passed as the answer. Since we limit our input for this model to just the project specification (and don't have answers readily available, nor would we expect this of new courses), we opt for this approach. As we will discuss later, we hypothesize that the quality of this second model performed worse than the first partly because of this limitations; our contexts weren't exactly answers (they just contained answers), meaning the model sought to curate questions based on strangely formatted "answers". For both models, we generated multiple questions per context. This was a parameter that we varied to find the best results.

3.1.4 Evaluate the Question/Answer (QA) Pairs

The final step for LLM #1 is to evaluate the quality of the QA pairs generated by each model. We utilized a combination of quantitative and qualitative methods to do so:

- Quantitative: iarfmoose/bert-base-cased-qa-evaluator numerical scoring
- Qualitative: manual evaluation

The first method we applied was an evaluator LLM provided by HuggingFace: iarfmoose/bert-base-cased-qa-evaluator. The evaluator accepts QA pairs and outputs their corresponding "scores" wherein higher scores are indicative of "better" QA pairs. A known/acknowledged limitation of this model is that it only evaluates QA pairs based on "if they are semantically related," and not on the validity/correctness of the information nor the relevance of the question. Because of the sheer volume of questions generated we used the evaluator to set a threshold score of questions that "passed" and those that "failed"; questions below a set score threshold were deemed "failed." Moreover, we hoped that sorting questions by score would provide some semblance of a ranking of the "best" and "worst" questions generated so manual evaluation would be slightly easier.

While we could utilize the evaluator for high-level semantic assessments of the generated QA pairs, we determined that manual inspection was the best way to assess their quality. For this, we randomly inspected questions from both passed/-failed datasets (for each model configuration) to validate accuracy and analyze the quality of generated QAs. This process allowed us to determine our optimal set of hyper-parameters as well as which model produced stronger QA pairs. We needed our evaluators to have a strong understanding of the project specification used as input for this evaluation to be effective. Two of our group members are veteran EECS 280 GSIs which we felt to be apt experience in comparing the questions generated by these models to those they would field in their instruction while helping students in OH and on Piazza for this project.

3.2 LLM #2

In this task, we explored approaches to develop a question-answering system tailored for a course with limited (or non-existent) historical Piazza data. The system aims to take a question as input and generate an answer in the style of course instructors. To achieve this, we harness the question-answering capabilities of large language models (LLMs) and experiment with various prompting techniques to address two key research questions:

1. Is synthetic data generated by our first LLM sufficiently reliable as a reference for building a generative QA system?
2. What challenges arise in building a system that can fully replicate a course instructor's question response style?

3.2.1 Experimental Setup

Our experiments are run on a server from Chameleon Cloud's TACC cluster, running Ubuntu 22.04, equipped with 2 AMD EPYC 7763 64-Core CPUs, 252 GB RAM, and a single NVIDIA A100 40GB GPU. The experiments were ran on torch 2.5.1, transformers 4.46.0.

3.2.2 Models

To evaluate different approaches fairly and consistently, while keeping the computing budget affordable, we use the `google/gemma-2-2b-it` variant of Gemma (Team et al., 2024b), a lightweight, state-of-the-art open model that can be viewed as the open source version of the Google Gemini (Team et al., 2024a). For generation of the question embeddings, we use the `all-MiniLM-L6-v2` model.

3.2.3 Datasets

We collected 4468 question-and-answer pairs on EECS280's Piazza Forum. Each pair contains

1. **Question Subject**: A student-written title for the question.
2. **Question Body**: The content of the student question submitted.
3. **Instructor Answer**: An official instructor response to the question asked.
4. (Optional) **Student Follow-ups**: Back-and-forth discourse in response to the instructor answer, if present.

The question subject and body are concatenated to form what we consider the "final question," while the instructor answer and follow-ups are concatenated to form what we consider the "ground truth answer." These real student questions and instructor answer pairs form our validation data. In addition, we use synthetic data generated during the optimal run of LLM #1 as QA pairs for training.

3.2.4 Methodology

To address the challenges of building a course-specific question-answering system with limited historical Piazza data, we employed two complementary methodologies:

- Zero-shot Question Answering
- Few-shot In-context Learning (RAG) Question Answering

Both approaches leverage the inherent capabilities of large language models (LLMs) to generate and refine responses tailored to the course's style and requirements.

Zero-Shot

In this approach, we directly input course-specific questions into an LLM without providing additional contextual examples. The synthetic data generated by our first LLM, and any other collected QA data are not used. The purpose of a zero-shot approach is to test the **inherent knowledge** of the LLM; how well can it naturally generate relevant, contextually appropriate answers to student questions based solely on its pre-trained abilities? We analyze an LLMs raw capacity to respond like real instructors without any supplemental or context-specific training. Because collecting existing data is not always possible, and generating synthetic data can be expensive, we want to observe the quality of responses when not informed by data. Finally, the results of this approach serve as a sort of baseline for how much and in what ways other techniques we apply improve generated responses.

Few-Shot In-Context

With this approach, we supply the LLM with a small number of QA pairs that are representative of the types of questions it should expect to receive and the corresponding types of answers we want it to generate. We do this prior to "asking" our LLM any question, and select the examples as a combination of synthetic data (collected from LLM #1) and validation data (from EECS 280 Piazza). The aim of this approach is to evaluate how much our model can be improved by providing it with examples. Is it better able to answer questions with an instructional tone, in the style of course instructors? Do examples enhance the factual relevance of responses?

3.2.5 Evaluation

In order to evaluate the results of this LLM, that requires comparing responses against a ground truth (actual instructor responses), to assess accuracy, relevance, and stylistic alignment. More broadly, we want to examine:

- Are generated answers semantically correct?
- Do generated answers align with expected instructor responses?

While semantic correctness is a relatively obvious need, recall that instructor answers have their own stylistic/conceptual/structural tendencies as a means of best enabling learning for students. For example, we would expect instructors not to readily give away answers, but rather lead students towards a better way to think about and/or approach problems. They may list steps or hints that help the student explore and discover the answer on their own. For these reasons (as with LLM #1) we apply a combination of quantitative evaluation through automated text-similarity based metrics, and qualitative evaluation through manual human inspection. Here, we discuss the three automated metrics we utilized as well as how we went about manual evaluation.

Quantitative Metric #1: BLEU and ROUGE

These metrics measure the overlap between the generated answers and the ground truth at the word and phrase levels, providing insights into lexical and structural similarities.

Quantitative Metric #2: BERTScore

This metric evaluates semantic similarity by computing embeddings of the generated and reference answers, offering a deeper understanding of how well the generated responses capture the meaning and intent of the instructor's answers.

Quantitative Metric #3: Meteor

Meteor extends traditional n-gram-based evaluation metrics by incorporating more advanced linguistic matching techniques. It calculates similarity between generated and reference texts by considering synonyms, stemming, and paraphrasing, thus providing a more nuanced assessment of translation and text generation quality beyond exact word matching.

Qualitative Metric: Human Evaluation

We employ a comprehensive human review process to assess the quality of generated responses. We evaluate a subset of answers across three critical dimensions:

- Style and tone matching instructional approach
- Factual accuracy aligned with course content
- Clarity in addressing specific questions

When significant deviations are identified, we conduct a systematic error analysis to uncover potential issues such as question misinterpretation, training data limitations, or response overgeneralization.

By combining these evaluation methods, we aim to gain a holistic understanding of LLM #2's performance, identify its strengths and limitations, and inform future improvements to better align with course instructor expectations.

3.3 Implementation of the In-Context Learning Approach

To enable our LLM to better answer student questions, we feed it sample QA pairs. The intuitions here are that (1) real student questions pulled from Piazza may show underlying patterns over time, and (2) the existing answers can act as "role models" from whom LLMs can learn about the appropriate tone and explicitness. The specific implementation is shown in Figure 3, on the following page. We first apply the `all-MiniLM-L6-v2` to generate embeddings for the questions. It maps sentences to a 384 dimensional dense vector space and can be used for tasks like clustering or semantic search. When answering new questions, the embedding of the new question will be used to search in the embedding space for which questions are similar to this one. We fetch the top-k similar question answer pairs and use them to construct the final prompt to the system. Finally, the system generates the answer for the new question.

4 Evaluation

4.1 LLM #1

For our first LLM, we ran a total of 13 different configurations (via altering hyper-parameters and models). The first 12 configurations were all performed with our first model (`potsawee/t5-large-generation-squad-QuestionAnswer`) and the final configuration was performed using the optimal hyper parameters identified for the first model on our second one (`iarfmoose/t5-base-question-generator`). In the following section, we will discuss the options we considered and decisions we made in selecting values for each hyper parameter, supported by QA examples. The approach we took in testing hyper parameters was first setting each of them to a sensible default value. We then progress through the list,

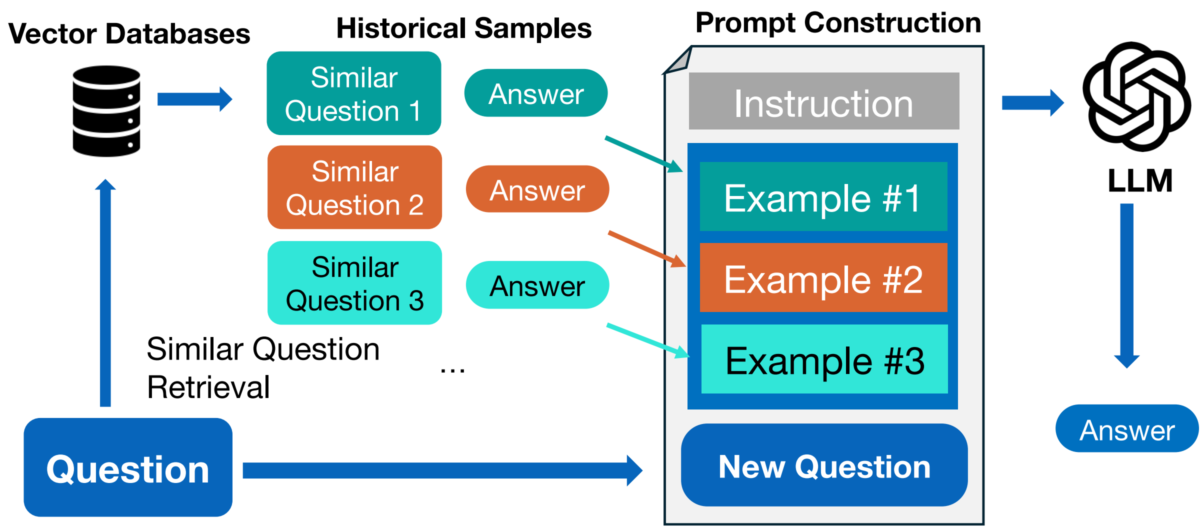

Figure 3: Workflow of In-Context Learning Approach for Question Answering

modifying one at a time, determining the optimal value for that parameter and then holding it at that value for the remainder of our configurations. First, here's a list of of the parameters we varied:

- Window Step Size
- Context Window Size
- Number of Questions per Context
- Score Threshold
- Model

4.1.1 Window Step Size

The window step size refers to how much our context window was shifted forward in the input text between each iteration (context generated). The essence of this parameter is balancing a step size that is too small wherein QA generated across different contexts are overly redundant, against one that is too large wherein information does not get properly represented by any of the contexts it is included in.

For the larger window step sizes (5, 10), it felt like questions either (a) didn't have enough context and/or (b) were missing relevant context such that both passed and failed questions weren't super sensible. We noticed several questions that were actually just sentences, sometimes pulled from the specification. For a window step size of 1, we saw noticeably better performance; questions made more sense, but were slightly more redundant. The contrast between some of the top-scored QA pairs for a larger vs. smaller window step size was quite apparent.

The top scoring QA pair and score for a window size of 10 was:

> **Question**: "Why might your code be easier to test and debug?"
> **Answer**: "May make for easier testing and debugging"
> **Score**: 3.76

The top scoring QA pair and score for a window size of 1 was:

> **Question**: "What does a user enter in order to discard a card?"
> **Answer**: "The user will then enter the number corresponding to the card they want to discard"
> **Score**: 3.79

While the questions produced by all configurations had some weaknesses, the one utilizing a window size of 1 was remarkably better. This was consistent throughout the QA pairs for these configurations. Not only were questions more grammatically correct, they also made more sense, had better coverage of information contained in the assignment, and were more similar to the types of questions we would expect from real students. From these observations, we moved forward with a window step size of 1 for the remaining configurations.

4.1.2 Context Window Size

The context window size refers to how much text (how large a context) we opted to send to the generative QA model. This parameter aids in the processes of determining the optimal way to send contexts to our model within a sliding window approach. Specifically, it balances a context window that is too small wherein contexts cannot hold enough information to construct sensible questions despite a small window step size, against a window that is too large wherein the model fails to understand contexts in a modular way.

For smaller context windows (40, 50, 60), the QA pairs produced were not noticeably different in any meaningful way. A context window size of 70 produced some really good questions, but definitely still produced questions that don't make sense and are slightly repetitive. For the largest window size (100), there wasn't a huge qualitative difference in the questions generated (compared to 70). It seems that at a certain point, expanding the window size doesn't improve question quality but limits the diversity of questions asked.

While we observed that expanding the window size did have a positive effect on the quality of QA pairs produced by the model, varying this parameter also demonstrated the limitations of having a model that cannot hold onto the knowledge conveyed in neighboring contexts. Regardless, we saw that smaller context window sizes limited the sensibility of questions generated while context windows that are too large cause the model to be too unfocused. For these reasons, we moved forward with a context window size of 70 in the remaining configurations.

4.1.3 Number of Questions per Context

The models we used allows for multiple QA pairs to be generated for each context sent to the model, so this variable specifies that value. This parameter balances generating too few QA pairs per context where the model fails to generate questions covering the entire context space, against too many QA pairs per context where questions become overly redundant. We tested generating 1, 2, and 4 questions per context. There was already some redundancy in questions in prior configurations, and the 4 QA pair generation configuration aligned with this observation as it failed to produce questions that were noticeably unique. At the same time, we did notice some of the QA pairs generated by the 2-question configuration were meaningfully unique. For example, here's a student-like question we didn't see in the 1-question configuration:

> **Question**: "Does a Simple Player consider whether their opponent is the dealer?"
> **Answer**: "Do not consider whether they are the dealer and could gain an additional trump by picking up the upcard"
> **Score**: 3.73

Again, the QA generated here is by no means semantically perfect. But, it does capture meaningful content from the project specification. We moved forward generating 2 questions per context in the remaining configurations.

4.1.4 Score Threshold

As we mentioned in our Methods section for LLM #1, we utilized a QA evaluator to numerically score each of the questions we generated (for semantic correctness), and sorted them in descending order. Thus far, we have separated passed/failed questions with a threshold of 1.5. In this step, we examined questions with scores surrounding this threshold, specifically values of 1.5, 2, 2.5, and 3. Ultimately, we observed several questions with scores in the high 2s that were still relevant, but felt that a lower threshold than that was likely to include more bad questions than good ones. Here's an example:

> **Question**: "What two players are in the game?"
> **Answer**: "A simple AI player and a human-controlled player that reads instructions from standard input cin"
> **Score**: 2.72

Ultimately, we moved forward with a score threshold of 2.5 in the remaining configurations.

4.1.5 Model

The model refers to whether QA pairs were generated with `potsawee/t5-large-generation-squad-QuestionAnswer` or `iarfmoose/t5-base-question-generator`. So far, all examples we have observed were generated from the first model. In running our second model, questions were significantly less semantically correct, didn't take the form of actual questions, and were long/rambly. Here's one such example:

> **Question**: "<pad> True ID: a vector is added to a card and no member variable can be used to type it. When using a const function, sort result in your swap function showing a weird error.</s>"
> **Answer**: "in Card.hpp. Use the STL to

sort a vector<Card> hand: Pitfall: Using sort on a member variable in a const member function leads to a confusing error, no matching function for call to 'swap'. Instead, call sort when adding a"
Score: 3.81

Despite somehow having a higher evaluation score than many of the questions produced by our first model, because of the significantly lower quality of the questions produced by this model, we carried forward with the results of our first model.

4.1.6 Summary

Ultimately, the combination of quantitative and qualitative metrics used to evaluate the QA pairs generated by our tested model in this section led us to the following final parameters:

- Window Step Size = 1
- Context Window Size = 70
- Number of Questions per Context = 2
- Score Threshold = 2.5
- Model = `potsawee/t5-large-generation-squad-QuestionAnswer`

This run resulted in a decent combination of sensible, student-like project questions and some attempted questions which still held room for improvement. Regardless, this was the best synthetic QA data we were able to produce, and what we carried forward as partial training data for LLM #2.

4.2 LLM #2

Our evaluations of LLM #2 serve as answers to the key questions guiding this project:

1. Can synthetic data serve as credible samples for effective curation of question answer bots?
2. Is the RAG-based in-context learning approach effective for answering previously unseen questions?

Finally, based on our findings with regards to each of these questions, we identify what challenges there are to building a good question-answering system to replicate 1-1 instructor-student tutoring.

4.2.1 Main Results

The evaluations/comparisons made in the following sections are an analysis of the data presented in Table 1.

	BLEU	ROUGE-L	BERTScore	Meteor
zero-shot	0	0.013	0.792	0.002
rag	0.009	**0.133**	**0.801**	**0.167**
rag simulation	**0.013**	0.128	0.775	0.148

Table 1: Evaluation Results of LLM #2. *zero-shot*: our baseline, no additional context given to the model, *rag*: our in-context learning approach that uses the synthetic data as the reference, *rag simulation*: our in-context learning apporach that uses the validation data capped up to the date of the current question as the reference.

Synthetic Data as Credible Samples

The experimental results suggest that synthetic data generated by LLM #1 can serve as credible samples for question-answering tasks. Specifically, the 'RAG' approach, which uses synthetic data as a reference for in-context learning, achieves substantial improvements over the 'zero-shot' baseline (which has no training). For instance, the **ROUGE-L** score improves from 0.013 to 0.133, and the **Meteor** score increases significantly from 0.002 to 0.167. These improvements indicate that the synthetic data provide valuable contextual information that aids the model in generating more sensible, accurate answers. Interestingly, the 'rag simulation' approach, which uses validation data (real Piazza QA) capped to the question date as the reference, performs slightly worse than 'rag' (synthetic data) in terms of **ROUGE-L** (0.128 vs. 0.133) and **Meteor** (0.148 vs. 0.167). This outcome highlights that synthetic data might better align with the model's pre-training distribution compared to real-world validation data, making it easier for the model to generalize. However, the marginal gap also suggests that real-world validation data still plays a vital role as it also still improves on the zero-shot context.

Effectiveness of RAG-Based In-Context Learning

The results strongly support the effectiveness of the RAG-based in-context learning approach for answering previously unseen questions. When compared to the 'zero-shot' baseline, 'rag' demonstrates significant improvements across all evaluation metrics except for **BLEU**. Notably, the **Meteor** score sees an 83.5% increase from 0.002 to 0.167, and the **ROUGE-L** score improves by an order of magnitude. These gains emphasize the ability of RAG to leverage context effectively, thereby bridging the gap between training data and unseen questions. It is worth noting that the **BERTScore** metric for ‘rag’ (0.801) shows only a marginal improvement over the baseline (0.792). This observation suggests that while RAG aids in structuring answers more coherently, the semantic similarity to the ground truth is still limited. This limitation could arise from inconsistencies in synthetic or reference data or the inherent complexity of the domain-specific questions.

4.3 Challenges in Building a Question-Answering System

Despite the positive results, the study reveals several challenges in building an effective question-answering system:

First, all approaches yield relatively low **BLEU** scores, with the highest being 0.013 for ‘rag simulation’. This indicates that lexical overlap between generated and reference answers remains a significant bottleneck. Such low scores highlight the difficulty in achieving exact word-level matches, especially in domain-specific and highly nuanced questions. Simultaneously, we acknowledge that lexical similarity and exact overlaps in words/phrases are not necessarily the most important metrics when analyzing answer quality.

Second, the improvement in **BERTScore** across approaches is modest, with the highest being 0.801 for ‘rag’. This suggests that while the generated answers improve in structure, their semantic alignment with the ground truth is still limited. A deeper analysis of the quality of both the synthetic and real validation data is required to address this issue.

Third, the dependence on synthetic data quality is evident. The superior performance of ‘rag’ over ‘rag simulation’ indicates that the synthetic data generated by LLM #1 might better align with the pre-training distribution of the model, thereby enabling more effective generalization. This observation underscores the importance of high-quality and diverse synthetic data for training robust systems.

Lastly, the experiments highlight the limitations of in-context learning for complex question-answering tasks. While RAG significantly outperforms the baseline, the absolute metric values remain low, suggesting that additional techniques, such as fine-tuning or enhanced retrieval mechanisms, might be necessary to further improve performance to a deployable level.

4.3.1 Challenges: Evaluation Fairness and Style Alignment

```
*Actual Instructor Answer*:
If you follow step 1 in @6, instructors
    will be able to see your uniqname
    even
if you post anonymously (i.e., if you
    post anonymously other students won'
    t
see your uniqname). If you want to
    understand what you missed on the
    private
test cases, I recommend going to
    Proffice hours. You can attend any
    Proffice
hours even if you don't attend that
    Professor's lectures.

*Model Answer*:
It's a good question, and it highlights
    a common misconception. While you
    can
certainly add getter functions to your `
    player` class in `euchre.cpp`, you
**shouldn't**. Let's break it down:

1. Visibility Within the Class
    - Getter functions are intended to
        allow clients or other parts of
        your code
      to safely access an object's
          private data.
    - By defining a getter function
        within your `player` class, you
        are
      potentially making the hand data
          accessible from anywhere.
```

Figure 4: Example Generated Answer and the Ground Truth

One possible explanation for the extremely low scores for all of our approaches might be related to the fairness of our evaluation metrics and style alignment. The primary challenge in evaluating educational QA systems lies in ensuring fairness and aligning evaluation metrics with the desired style and intent of responses. Lexical-based metrics, such as BLEU and ROUGE, often penalize responses that diverge in phrasing or structure from the ground truth, even if the generated answers are semantically relevant or meaningful. For example, as shown in Figure 4, the model-generated answer focuses on coding practices and getter functions, which is tangential to the actual query about private tests but may still provide valid insights. However, such responses are unfairly penalized due to their deviation from the expected response format. Furthermore, educational contexts often require adherence to an instructive and supportive tone, which can lead to further mismatches if models produce technical or overly casual responses. These challenges underscore the need for evaluation metrics that balance semantic correctness, relevance, and stylistic alignment to ensure fair and contextually appropriate assessments of model performance. Ultimately, it is a very complex problem to determine how to form an evaluation metric that can fairly assess the quality of these kinds of question answers. To an extent, real instructor answers are highly subjective such that there is no ultimately "correct" (truth) answer to these kinds of questions. Even within real instruction, there is a lot of ambiguity in how to balance vagueness in responses with giving away answers to students, and this issue is only exacerbated when we try to assign answersxw numerical scores that assess their quality.

5 Discussion of Results

The results of this project demonstrate both the promise and the limitations of utilizing synthetic data and retrieval-augmented generation (RAG)-based in-context learning for educational QA systems. Synthetic data generated by LLM #1 proved to be credible and effective for training purposes, as evidenced by significant improvements in metrics such as ROUGE-L and Meteor when compared to the zero-shot baseline. This highlights the potential of synthetic QA data to fill gaps in historical datasets, particularly for new or evolving courses.

However, the study also reveals the inherent challenges in aligning generated answers with both the semantic and stylistic expectations of instructional responses. For instance, while RAG-based methods significantly improved contextual understanding and coherence, metrics like BLEU and BERTScore indicate limitations in achieving precise lexical and semantic alignment. This is further complicated by the evaluative metrics themselves, which often penalize stylistic or structural deviations, even when the content of the generated answer is relevant and insightful. The discrepancy between the generated answers and the ground truth highlights the need for alternative metrics that account for stylistic alignment and the instructive tone essential in educational contexts.

Additionally, while synthetic data demonstrated a closer alignment with the pretraining distribution of the models, real-world validation data offered complementary benefits, emphasizing the importance of a hybrid approach. The reliance on high-quality synthetic data underscores the necessity of refining QA generation techniques, including improved context creation and filtering strategies.

Overall, while RAG-based in-context learning has shown considerable potential, the relatively low absolute metric values indicate that further refinements—such as domain-specific fine-tuning, enhanced retrieval mechanisms, and the integration of diverse data sources—are required to build robust and reliable question-answering systems. These findings underscore the complexities of creating AI systems capable of emulating the nuanced responses of human instructors while providing actionable insights for future research.

6 Conclusion

This research project explores the potential of LLMs to improve computer science education. It shows that while using LLMs to generate practice questions and offer student support is possible, there are still obstacles to overcome. Improving the quality of the synthetically generated data is crucial for creating a reliable system that can accurately answer student questions. This involves exploring better techniques for creating context and retaining knowledge from previous interactions. The evaluation of the question-answering LLM suggests future research in having more accurate benchmarks for complex question answering systems, and how LLMs can align with the actual instructors to give inspiring yet not explicit answers. These findings highlight both the promise and the complexities of integrating LLMs into educational settings.

7 Division of Work

Our team consists of four members. For the actual implementations of each LLM, we split into two subteams (one for each LLM). The team members with teaching experience for EECS 280 were selected to work on LLM #1 while the other team members worked on LLM #2. The team worked collaboratively on the project proposal, presentation, and final paper to ensure all findings were well represented and to provide complete, insightful final results.

8 Codebase URL

https://github.com/stoneann/AutoQATrainedLLM/tree/main

References

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. Ms marco: A human generated machine reading comprehension dataset. *Preprint*, arXiv:1611.09268.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Ryan C. Barron, Ves Grantcharov, Selma Wanna, Maksim E. Eren, Manish Bhattarai, Nicholas Solovyev, George Tompkins, Charles Nicholas, Kim Ø. Rasmussen, Cynthia Matuszek, and Boian S. Alexandrov. 2024. Domain-specific retrieval-augmented generation using vector stores, knowledge graphs, and tensor factorization. *Preprint*, arXiv:2410.02721.

Rahul Basu, Dolasaha, Chiranjib Dutta, and Ananjan Maiti. 2023. *Automatic Transformation and Generation of Question Papers in Education with NLP Techniques*.

BENJAMIN S. BLOOM. 1984. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16.

Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary 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. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA. Association for Computing Machinery.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Potsawee Manakul, Adian Liusie, and Mark JF Gales. 2023. Mqag: Multiple-choice question answering and generation for assessing information consistency in summarization. *arXiv preprint arXiv:2301.12307*.

Rounak Meyur, Hung Phan, Sridevi Wagle, Jan Strube, Mahantesh Halappanavar, Sameera Horawalavithana, Anurag Acharya, and Sai Munikoti. 2024. Weqa: A benchmark for retrieval augmented generation in wind energy domain. *Preprint*, arXiv:2408.11800.

Matt Post. 2018. A call for clarity in reporting bleu scores. *Preprint*, arXiv:1804.08771.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint*, arXiv:1910.10683.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *Preprint*, arXiv:1806.03822.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. Coqa: A conversational question answering challenge. *Preprint*, arXiv:1808.07042.

Lala Septem Riza, Yahya Firdaus, Rosa Ariani Sukamto, Wahyudin, and Khyrina Airin Fariza Abu Samah. 2023. Automatic generation of short-answer questions in reading comprehension using nlp and knn. *Multimedia Tools and Applications*, 82(27):41913–41940.

Gemini Team, Rohan Anil, Sebastian Borgeaud, and Jean-Baptiste Alayrac et al. 2024a. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, and Shreya Pathak et al. 2024b. Gemma: Open models based on gemini research and technology. *Preprint*, arXiv:2403.08295.

Altaj Virani, Rakesh Yadav, Prachi Sonawane, and Smita Jawale. 2023. Automatic question answer generation using t5 and nlp. In *2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, pages 1667–1673.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. *Preprint*, arXiv:1904.09675.