Efficient Classification of Student Help Requests in Programming Courses Using Large Language Models

Jaromir Savelka
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
jsavelka@cs.cmu.edu

Paul Denny
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Mark Liffiton
Illinois Wesleyan University
Bloomington, Illinois, USA
mliffito@iwu.edu

Brad Sheese
Illinois Wesleyan University
Bloomington, Illinois, USA
bsheese@iwu.edu

Abstract

The accurate classification of student help requests with respect to the type of help being sought can enable the tailoring of effective responses. Automatically classifying such requests is non-trivial, but large language models (LLMs) appear to offer an accessible, cost-effective solution. This study evaluates the performance of the GPT-3.5 and GPT-4 models for classifying help requests from students in an introductory programming class. In zero-shot trials, GPT-3.5 and GPT-4 exhibited comparable performance on most categories, while GPT-4 outperformed GPT-3.5 in classifying sub-categories for requests related to debugging. Fine-tuning the GPT-3.5 model improved its performance to such an extent that it approximated the accuracy and consistency across categories observed between two human raters. Overall, this study demonstrates the feasibility of using LLMs to enhance educational systems through the automated classification of student needs.

1 Introduction

The emergence of large language models (LLMs) has opened up new possibilities for enhancing educational tools and services. In particular, one promising application of LLMs is providing personalized on-demand assistance at scale [9, 10]. This is especially valuable in courses with growing enrollments, such as introductory programming courses, where student-to-instructor ratios are large [17]. When students seek help from an automated assistant, they may ask a wide range of different types of queries related to their programming assignments or projects. The ability to classify these queries into distinct categories can have important educational implications, as evidenced by the related work (Section 2). For example, if a student requests help implementing code directly related to an assignment, an appropriate response may be to restate the specifications more simply or ask the student to clarify what is unclear to them. On the other hand, if a student requests assistance in debugging code, then a targeted hint about resolving the bug may be a useful response. Moreover, identifying the types of queries that students tend to ask most frequently can be valuable feedback for instructors and researchers.

Classifying student queries into suitable categories is difficult and time consuming as queries can differ in subtle ways and require expert knowledge to assess reliably. In addition, automatic classification is necessary for integration into a tool, but expensive because it typically requires large amounts of expert-labeled data for training classifiers. Given the recent successes of LLMs in computing education [24, 13, 31, 33], we explore the viability of using GPT-3.5 and GPT-4 to automatically classify

NeurIPS'23 Workshop on Generative AI for Education (GAIED).

student help requests when there is either little or no labeled training data available. Specifically, our research questions were as follows:

(RQ1) How accurately can GPT-3.5 and GPT-4 perform zero-shot classification of student help requests based on the coding instructions originally designed for human raters?

(RQ2) To what extent can classification performance be improved by fine-tuning using a limited amount of data?

2 Related Work

Researchers have long been interested in automatically categorizing student requests for online help and educational forum posts. Gao et al. used a gradient boosting framework to classify student help requests based on the sufficiency of information provided (e.g., *useless*, *sufficient*, or having a *copied error*) [7]. Similarly, Švábenský et al. used several traditional ML algorithms (e.g., random forest or linear regression) to classify student posts according to their urgency on an ordinal scale (e.g., *not actionable*, *extremely urgent*) [37]. Xu and Lynch utilized a combination of a convolutional neural network and long short term memory model (CNN-LSTM), and Bi-directional LSTM (BiLSTM) to automatically classify MOOC discussion posts as to whether they were *seeking help*, and to identify what kind of question was being asked (*course content*, *technique*, or *course logistic*) [42]. Sha et al. compared several traditional ML algorithms (e.g., random forest) to deep learning algorithms (e.g., CNN, LSTM) on classifying student forum posts from two datasets—the Stanford MOOC posts dataset [1] encoding, e.g., *urgency* or *sentiment* of the posts, and their own dataset with posts labeled as *content* and *process* [35]. Onan and Toçoğlu utilized clustering (unsupervised – no training data required) to assign student questions to topic categories[20].

A number of studies have demonstrated the value of classifying student help requests and forum posts by manually categorizing them into schemes based on the nature of the questions. For example, Gao et al. analyzed the proportion and evolution over time of student request types, dividing them into eight categories related to, e.g., *general debugging and addressing issues* or *implementation and understanding* [6, 5]. Vellukunnel et al. analyzed discussion forum posts, distinguishing student posts where students did not demonstrate effort (*active*) from posts that did showed effort to solve a problem (*constructive*) [39]. To date, classification has required time-intensive manual coding by researchers. However, LLMs have the potential to enable faster, cheaper, and more consistent analysis of patterns and trends in student queries as evidenced by work in other domains [32, 34].

Several studies have investigated unproductive help-seeking behaviors in tutoring systems, such as *help abuse* and *try-step abuse*, in both general tutoring [29, 28] and programming contexts [16]. However, there is limited research on leveraging help request categorization to improve interactions in programming assistance chatbots. To our knowledge, Carreira et al. has developed the only programming chatbot (Pyo) that utilizes predefined categories for student questions like (*exercise assistance*, *error guidance*, *concept definitions*) [3]. Although programming chatbots are an active research area, most do not distinguish between different student question types (e.g., Python-bot [18], RevBot [19], Duckbot [30] and others [11, 40]). Existing systems do not tailor responses based on the intent behind students' inquiries. Categorizing questions allows personalized interactions that target the specific help students request. This study provides an initial step in demonstrating the feasibility of using query categorization to improve programming chatbots.

3 Dataset

One of the authors of this paper developed CodeHelp, an automated assistant that responds to semi-structured student queries in programming and CS courses [14]. CodeHelp uses LLMs to generate responses to requests posed in natural language. Students request help via a form with separate inputs for the programming language they are using, a snippet of relevant code, an error message if they have one, and a question or description of the issue they are facing. Responses are generated by a series of prompts to LLMs. One prompt checks whether the student's inputs are sufficient to be able to provide them with effective help, and if additional information is needed, it generates a request for clarification that is presented to the student. Another prompt, run concurrently, combines the student's inputs with instructions to provide guidance and explanations, along with class-specific context provided by the instructor, and its completion is used as the main response for the student.

Language:
Python
Code:
`major_series.sort_index['MUSIC']`
Error:
`'method' object is not subscriptable`
Issue:
I'm trying to index and find the value associated with the index label ('music') but it keeps saying this. How do I make it at least not give me an error?

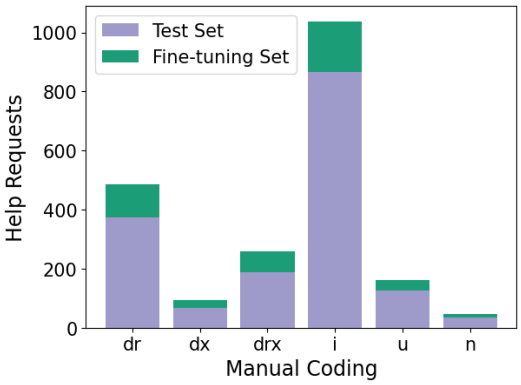

Figure 1: An example student help request is shown on the left. Counts of help requests by coding category and by data set split are shown on the right (the request codes are explained in the text).

We deployed CodeHelp in two sections of an undergraduate introductory and data-science course, totalling 52 students, taught by an author of this paper in the Spring semester of 2023. During the course, students submitted 2,082 unique queries requesting help. As reported in [36], the queries were independently coded by two of the authors into the following categories:

1. *Debugging*: Queries seeking help to resolve errors in code; sub-categorized into queries that included: a) the error (dr); b) the desired outcome (dx); or c) both (drx).
2. *Implementation* (i): Queries about implementing code to solve specific assignment problems.
3. *Understanding* (u): Queries focused on gaining an understanding of programming concepts.
4. *Nothing* (n): Queries that provided no error or meaningful issue.

Human raters showed substantial reliability for ratings of all categories and sub-categories ($\kappa = .75$). Overall reliability was even higher ($\kappa = .83$) when Debugging sub-categories were collapsed into a single Debugging category [23]. For the current research, if there was disagreement between human raters, we used the rating from the more experienced rater as the "gold-label" classification.

We divided the data set into a fine-tuning set and a test set. The fine-tuning set was used to fine-tune an LLM. The split was performed on the basis of students, i.e., all the help requests submitted by a specific student were included in the same set. We randomly selected 10 students and included their requests in the fine-tuning set. Considering the number of requests submitted by each student, we made sure that two of the selected students were from the lowest quartile, six from second and third quartiles, and two from fourth quartile. Out of the 2,082 help requests, 423 were selected for the fine-tuning set, and the remaining 1,659 were included in the test set (see Figure 1).

4 Experiments

Models The original GPT model's core capability is *fine-tuning* on a downstream task [25]. The GPT-2 model displays remarkable *multi-task learning* capabilities [26]. The main focus of [2] was to study the dependence of performance on model size where eight differently sized models were trained—the largest of the models is commonly referred to as GPT-3 (175 billion parameters). The interesting property of these very large models is that they appear to be very strong *zero- and few-shot learners* [2]. The work of [22] focused on the *alignment problem*, demonstrating the apparent usefulness of fine-tuning the LLMs to follow instructions (RLHF). In this paper, we evaluated `gpt-3.5-turbo-0613` and `gpt-4-0613` [21]—some of the recently released GPT models.

Baselines BERT (bidirectional encoder representation from transformers) [4, 38] has gained immense popularity. A large number of models using similar architectures have been proposed [12, 27], including RoBERTa (robustly optimized BERT pretraining approach) [15]. A base model of RoBERTA (125 million parameters) is used as a baseline in the current study. A *random forest* [8] is an ensemble classifier that fits a number of decision trees on sub-samples of the data set.

Table 1: Evaluation metrics examining the performance of GPT-3.5 and GPT-4 in zero-shot settings and when fine-tuned on 423 student help requests.

		ZERO-SHOT						FINE-TUNED		
		GPT-3.5			GPT-4			GPT-3.5		
Query Category	Count	P	R	F_1	P	R	F_1	P	R	F_1
Debugging	630	.84	.91	.87	.90	.77	.83	.94	.92	.93
(error) – dr	374	.64	.02	.04	.69	.44	.54	.76	.90	.82
(outcome) – dx	67	.10	.09	.09	.23	.36	.28	.63	.36	.46
(error & outcome) – drx	189	.23	.75	.35	.50	.51	.50	.62	.46	.53
Implementation – i	867	.82	.89	.85	.78	.93	.85	.94	.93	.93
Understanding – u	127	.82	.24	.38	.74	.48	.58	.77	.85	.81
Nothing – n	35	.33	.06	.10	.50	.11	.19	.70	.89	.78
Overall	**1659**	**.82**	**.83**	**.81**	**.82**	**.82**	**.81**	**.92**	**.92**	**.92**
(debugging types)	**1659**	**.67**	**.58**	**.53**	**.70**	**.70**	**.68**	**.83**	**.84**	**.83**

We included a random forest model in our experiments so that we could compare the GPT models to a well-regarded traditional ML technique.

Experimental Design In the zero-shot settings, we submit the requests from the test set one by one using the `openai` Python library[1] which is a wrapper for the OpenAI's REST API.[2] In our experiments we did not encounter any issues stemming from the models' prompt length limitations. Consequently, we neither adapted nor explored any measures to mitigate prompt length limitation issues. We included the coding instructions originally designed for human raters in the system part of the prompt (Appendix A.1) and a student help request in the user message (Appendix A.2), which were then combined into the prompt directly provided to the LLM to generate the completion. Each prompt completion (response) returned a predicted label, which we then compared to the gold-label (i.e., the human-assigned category). We fine-tuned the `gpt-3.5-turbo-0613` model on 50, 100, 200 and all 423 student help requests included in the fine-tuning set. Hence, we could observe the effects of fine-tuning on progressively larger data sets. Each data point was structured following the exact same format of the system part of the prompt and the user message described above. All of the models were fine-tuned for 3 epochs. To evaluate the performance of the models, we used Precision (P), Recall (R), and F_1-measure.

5 Results

Table 1 shows per class metrics as well as their overall weighted averages. The performance on the four main categories appeared to be similar for both, `gpt-3.5-turbo-0613` and `gpt-4-0613`, in the zero-shot settings, achieving the overall F_1 score of .81. There was a noticeable difference when it came to handling the *Debugging* sub-categories. When these were considered, the GPT-4 model achieved overall $F_1 = .68$ while the F_1 score of the GPT-3.5 model dropped to .53. Closer examination shows that, the drop was explained by the 339 *Debugging – error (dr)* help requests that were predicted as *Debugging – error & outcome (drx)* by the GPT-3.5 model (Figure 2). This was also reflected in the κ agreement scores with the manually assigned codes. Both the models achieved similar agreement with the gold-labels on the four main categories ($\kappa = .69$ for GPT-3.5, $\kappa = .67$ for GPT-4). When the *Debugging* sub-categories were considered, the GPT-4 model ($\kappa = .52$) clearly outperformed the GPT-3.5 model ($\kappa = .36$).

Fine-tuning the `gpt-3.5-turbo-0613` model substantially improved performance. GPT-3.5 fine-tuned on all the 423 data points from the fine-tuning set achieved the overall F_1 scores of .92 (top-level categories) and .83 (with *Debugging* sub-categories). The agreement of the fine-tuned model with the gold standard matched the agreement between the two human raters—$\kappa = .75$ ($\kappa = .75$ human)

[1] OpenAI Python Library. Available at: `https://pypi.org/project/openai/0.28.0/` [2023-09-17]

[2] We set the `temperature` 0.0 (no randomness), `max_tokens` to 10 (a response is a single label consisting of 1–3 letters), `top_p` to 1 (recommended when `temperature` is set to 0.0), and both `frequency_penalty` and `presence_penalty` to 0 (no penalty to repetitions or to tokens appearing multiple times in the output).

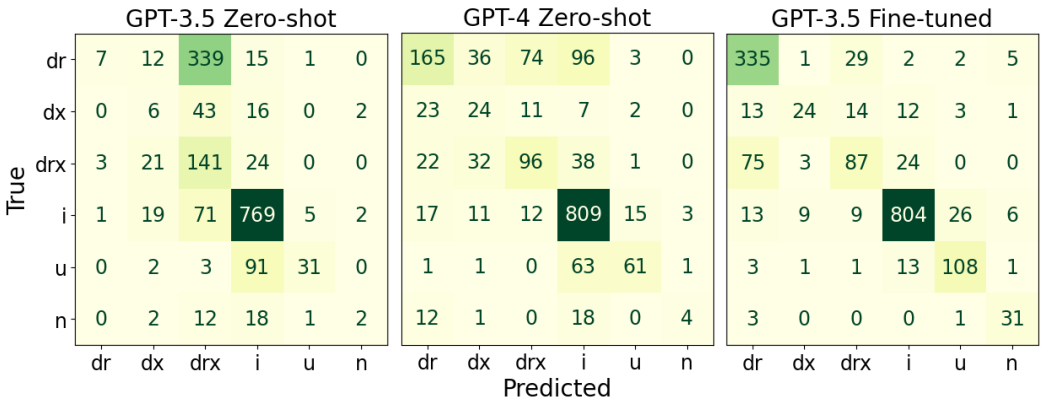

Figure 2: Confusion matrices of GPT-3.5 and GPT-4 classification output in zero-shot settings and when fine-tuned on 423 student help requests (refer to Table 1 for codes).

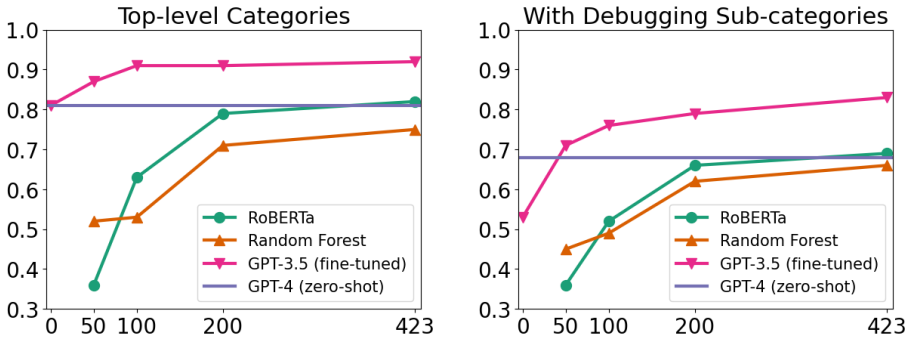

Figure 3: The comparison of GPT-4 and GPT-3.5 performance compared to random forest and RoBERTa base when trained/fine-tuned on progressively larder pool of data points up to 423.

with *Debugging* sub-categories and $\kappa = .86$ ($\kappa = .83$ human) when *Debugging* sub-categories were collapsed. Figure 2 provides detailed insight into the differences in handling the student help request classification task between the GPT-3.5 (zero-shot and fine-tuned) and GPT-4 (zero-shot).

Figure 3 demonstrates the key benefit of performing the classification with LLMs, such as GPT-3.5 or GPT-4, as compared to traditional ML algorithms or smaller LLMs. The LLMs performed reasonably even if no or very little ($n < 100$) labeled data were available. A smaller LLM (RoBERTa base) required several hundred labeled data points to match the zero-shot performance of GPT-4, while a traditional ML algorithm such as random forest required even more labeled data (consistent with findings in other domains [32, 34]). A small amount of labeled data ($n \simeq 100$) was sufficient for the fine-tuned GPT-3.5 to perform comparably to humans on the easier task of labeling the four top-level categories. While it was also possible to match human performance on the more challenging task with the *Debugging* sub-categories, a larger amount of labeled data was required ($n \simeq 400$).

6 Discussion

Our results suggest that LLMs can perform classification tasks like ours on student help requests both accurately and inexpensively. Compared to human raters, LLMs reach similar levels of performance at a very *small fraction of the cost*, with much higher speed, low setup complexity, and greater flexibility to adapt to new or modified labeling schemes and educational contexts. This can enable novel features in automated assistance systems such as CodeHelp. As to the cost, the fine-tuning of the `gpt-3.5-turbo-0613` on the 423 requests was performed over 3 epochs (1,269 steps). The overall number of submitted tokens was 1,003,722. At the time, the cost of fine-tuning the model was

set to $0.008/1K tokens.[3] Hence, the overall cost of the procedure was $8.03. For fine-tuning on 50 requests (117,609 tokens), the price was $0.94. The current cost of using a fine-tuned GPT-3.5 model was $0.012/1K for input tokens and $0.016/1K for completions. The employment of the fine-tuned model as a classification component in CodeHelp over the Spring'23 semester would have amounted to less than $30 additional cost.[4] Using the general (not fine-tuned) GPT-3.5 model would cost less than $4 while GPT-4 would cost roughly $80.

By automatically classifying student requests into types, an LLM-powered system can provide instructors with rich, real-time aggregated information about their students' questions and help-seeking behaviors both across a class and for individual students. This could allow an instructor to, for example, identify a shift in query types within a class that could suggest an increased difficulty in a module. Similarly, they could observe a heavy reliance on one type of query by an individual student, such as a student only ever asking *Debugging* questions without providing an error or incorrect outcome. This could trigger an intervention to identify the cause and help the student improve their approach. The system itself could also utilize the classification as part of its operation to improve its responses. For example, it could use the classification of a query to choose among different specialized prompts when generating a main response. The user interface could automatically request additional information from the user when certain query types are recognized. Students often do not know how to communicate effectively about technical subjects, and automated classification of their requests can play an important role in guiding them to more effective requests.

Using LLMs as a service to perform classification tasks is more accessible than using other ML-powered methods. LLMs are hosted and available via APIs, requiring little to no local infrastructure and relatively little technical expertise. The available models perform reasonably well with no labeled data. In our context, a small amount of labeled data to fine-tune a model yielded performance similar to that of human classifiers. The fine-tuning is performed via an API as well, and it is fast and inexpensive. This all suggests both an ease of integration with existing systems and a low barrier to experimenting with many different labeling schemes. This allows tailoring a system to specific educational contexts as well as rapid iterative improvement of existing schemes.

Limitations This study is an initial exploration rather than a comprehensive benchmark. We did not seek to maximally optimize model performance and do not claim that our results allow models to exhibit their best performance. Thus, this study should not be viewed as precisely measuring model capabilities, but rather hinting at the potential of LLMs in zero-shot settings or with minimal fine-tuning. There are several potential avenues to explore with regards to improving performance: prompt instructions for classification included an unaltered copy of the coding instructions developed for human raters. The prompt also included instructions to prevent the model from explaining its predictions, but generating an explanation followed by a prediction could lead to improvements [41]. More thorough experimentation with hyper-parameters could yield improved performance across all the studied models. It is not clear to what degree our findings would generalize to student queries from other courses, or other query classification schemes, or to non-English speaking courses.

7 Conclusions and Future Work

We explored the use of LLMs for the classification of student help requests in introductory programming classes. We found that GPT-3.5 and GPT-4 models achieved reasonable accuracy in a zero-shot setting. Our results also showed that fine-tuning the GPT-3.5 model on a small amount of labeled data greatly improved its performance, reaching human-level accuracy. Our findings have important implications for personalized and scalable assistance in education. Automated systems that accurately classify student queries can provide tailored and effective responses to students and deliver insights to educators about how students are interacting with such tools.

For future work, it would be valuable to explore the generalizability of our methods to other disciplines and model architectures. Additionally, further research can investigate the impact of different prompt instructions and hyper-parameters on the performance of LLMs for student query classification. Furthermore, it would be worthwhile to study the potential of fine-tuned LLMs in improving student interactions with assisting chatbots.

[3] OpenAI: Pricing. Available at: `https://openai.com/pricing` [Accessed 2023-09-17]

[4] 2,591 submitted requests (not de-duplicated) with length of 1,000 input tokens and 10 for completions.

References

[1] Akshay Agrawal and Andreas Paepcke. The stanford mooc posts dataset. `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`, 2015. Accessed: 2010-09-30.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Gustavo Carreira, Leonardo Silva, Antonio Jose Mendes, and Hugo Goncalo Oliveira. Pyo, a Chatbot Assistant for Introductory Programming Students. In *2022 International Symposium on Computers in Education (SIIE)*, pages 1–6, Coimbra, Portugal, November 2022. IEEE.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT 2019*, NAACL-HLT '19, pages 4171–4186. Association for Computational Linguistics, 2019.

[5] Zhikai Gao, Bradley Erickson, Yiqiao Xu, Collin Lynch, Sarah Heckman, and Tiffany Barnes. Admitting you have a problem is the first step: Modeling when and why students seek help in programming assignments. *International Educational Data Mining Society*, 2022.

[6] Zhikai Gao, Bradley Erickson, Yiqiao Xu, Collin Lynch, Sarah Heckman, Tiffany Barnes, et al. You asked, now what? modeling students' help-seeking and coding actions from request to resolution. *Journal of Educational Data Mining*, 14(3):109–131, 2022.

[7] Zhikai Gao, Collin Lynch, Sarah Heckman, and Tiffany Barnes. Automatically classifying student help requests: A multi-year analysis. *International Educational Data Mining Society*, 2021.

[8] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[9] Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stepha Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023.

[10] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. Studying the effect of ai code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[11] Mario Konecki, Nikola Kadoic, and Rok Piltaver. Intelligent assistant for helping students to learn programming. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 924–928, Opatija, Croatia, May 2015. IEEE.

[12] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[13] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. Comparing code explanations created by students and large language models, 2023.

[14] Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling Conference on Computing Education Research*, Koli Calling '23, New York, NY, USA, 2023. Association for Computing Machinery.

[15] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[16] Samiha Marwan, Anay Dombe, and Thomas W Price. Unproductive help-seeking in programming: What it is and how to address it. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 54–60, 2020.

[17] National Academies of Sciences, Engineering, and Medicine and others. *Assessing and responding to the growth of computer science undergraduate enrollments*. National Academies Press, 2018.

[18] Chinedu Wilfred Okonkwo and Abejide Ade-Ibijola. Python-Bot: A Chatbot for Teaching Python Programming. *Engineering Letters*, 29:25–34, 02 2021.

[19] Chinedu Wilfred Okonkwo and Abejide Ade-Ibijola. Revision-Bot: A Chatbot for Studying Past Questions in Introductory Programming. *IAENG International Journal of Computer Science*, 49(3), 2022.

[20] Aytuğ Onan and Mansur Alp Toçoğlu. Weighted word embeddings and clustering-based identification of question topics in mooc discussion forum posts. *Computer Applications in Engineering Education*, 29(4):675–689, 2021.

[21] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.

[23] Cliodhna O'Connor and Helene Joffe. Intercoder reliability in qualitative research: debates and practical guidelines. *International journal of qualitative methods*, 19:1609406919899220, 2020.

[24] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Peterson, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. The robots are here: Navigating the generative ai revolution in computing education. *arXiv preprint arXiv:2310.00658*, 2023.

[25] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. `https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf`, 2018. Accessed: 2023-09-30.

[26] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. `https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`, 2019. Accessed: 2023-09-30.

[27] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[28] Ido Roll, Vincent Aleven, Bruce M McLaren, and Kenneth R Koedinger. Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system. *Learning and instruction*, 21(2):267–280, 2011.

[29] Ido Roll, Vincent Aleven, Bruce M McLaren, Eunjeong Ryu, Ryan SJ d Baker, and Kenneth R Koedinger. The help tutor: Does metacognitive feedback improve students' help-seeking actions, skills and learning? In *Intelligent Tutoring Systems: 8th International Conference, ITS 2006, Jhongli, Taiwan, June 26-30, 2006. Proceedings 8*, pages 360–369. Springer, 2006.

[30] Margot Rutgers. Duckbot: A chatbot to assist students in programming tutorials. Master's thesis, University of Twente, 2021.

[31] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 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*, ICER '22, page 27–43, New York, NY, USA, 2022. Association for Computing Machinery.

[32] Jaromir Savelka. Unlocking practical applications in legal domain: Evaluation of gpt for zero-shot semantic annotation of legal texts. In *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law*, ICAIL '23, page 447–451, New York, NY, USA, 2023. Association for Computing Machinery.

[33] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. Thrilled by your progress! large language models (gpt-4) no longer struggle to pass assessments in higher education programming courses. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, page 78–92, New York, NY, USA, 2023. Association for Computing Machinery.

[34] Jaromir Savelka and Kevin Dean Ashley. The unreasonable effectiveness of large language models in zero-shot semantic annotation of legal texts. *Frontiers in Artificial Intelligence*, 6:1279794, 2023.

[35] Lele Sha, Mladen Raković, Jionghao Lin, Quanlong Guan, Alexander Whitelock-Wainwright, Dragan Gašević, and Guanliang Chen. Is the latest the greatest? a comparative study of automatic approaches for classifying educational forum posts. *IEEE Transactions on Learning Technologies*, 2022.

[36] Brad Sheese, Mark Liffiton, Jaromir Savelka, and Paul Denny. Patterns of student help-seeking when using a large language model-powered programming assistant, 2023.

[37] Valdemar Švábenskỳ, Ryan S Baker, Andrés Zambrano, Yishan Zou, and Stefan Slater. Towards generalizable detection of urgency of discussion forum posts. In *Proceedings of the 16th International Conference on Educational Data Mining*, pages 302–309. International Educational Data Mining Society, 2023.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] Mickey Vellukunnel, Philip Buffum, Kristy Elizabeth Boyer, Jeffrey Forbes, Sarah Heckman, and Ketan Mayer-Patel. Deconstructing the discussion forum: Student questions and computer science learning. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pages 603–608, 2017.

[40] James Walden, Nicholas Caporusso, and Ludiana Atnafu. A Chatbot for Teaching Secure Programming. In *Proceedings of the EDSIG Conference ISSN*, volume 2473, page 4901, 2022.

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

[42] Yiqiao Xu and Collin F Lynch. What do you want? applying deep learning models to detect question topics in mooc forum posts. In *2019 KDD Workshop on Deep Learning for Education*, 2019.

A GPT Prompts

A.1 System Part of the Prompt

The system part of the prompt is typically used to steer the GPT dialogue-focused models towards performing the desired task. We introduced only minimal changes to the coding instruction to ensure close mapping between the original task performed by human raters and the task performed by GPT-3.5 and GPT-4 automatically. Below is an excerpt from the system part of the prompt used in our experiment. The gray `[...]` tokens indicate a part has been left out for brevity of the presentation.

```
You are an educational assistant bot focused on analysis of student help requests in introductory
programming courses. Given a help request you assign it with one of the below defined codes.

CODES
dr - Debugging (with error)
dx - Debugging (with expected outcome)
drx - Debugging (with error and expected outcome)
u - Understanding
i - Implementation / How to
n - Nothing

DEBUGGING REQUESTS
Cases where students are looking for help to solve specific errors and faults in their code. Must include
some description of or pointer to an error -or- an indication of what it was supposed to do (even if no
error is provided).

Typical requests in this category:
- Why doesn't/can't/isn't the code X
- The code doesn't do X
- It is supposed to X but it is doing Y
- I am trying to do X but my code does Y
- Error submitted with no description or context
- Error submitted with desired outcome described

Sub-classifications (either or both may be true, must be at least one):
dx) Does the query include an indication of the problem or incorrect outcome? (Error message or
    description of what it does that is wrong.)
dr) Does the query include an indication of what the code is supposed to do?
drx) Does the query include both an indication of the problem and what the code is supposed to do?

GUIDANCE ON CODE IMPLEMENTATION
Queries about implementing code or functions to solve specific assignment problems. Must include code
and/or reference assignment instructions. High issue-instructions equivalence indicates that the student
is likely asking for help with a specific assignment problem.

Typical requests in this category:
- How do I create X
- How do I write a function that X
- How do I get it to X
- I want to X
- No request, just restates course instructions

DEVELOPING UNDERSTANDING
Requests centered around gaining an understanding of programming concepts, algorithms, data structures,
language and library features but not obviously asking how to complete a given assignment problem. Must
not include code or reference assignment instructions.

[...]

INPUT
As input you will receive four elements:
Issue - a string describing the issue if any
Code - a string containing the code submitted by the student if any
Error - a string containing the error message if any
Issue-instructions equivalence - a percentage indicating how much the issue matches the assignment
instructions

OUTPUT
As a response to the provided help request return one of the codes. Do not provide any explanations.

EXAMPLE OUTPUT 1
i

EXAMPLE OUTPUT 2
drx [...]
```

A.2 User Message Template

A student help request is provided via a user message. The green tokens with curly braces are replaced with the actual data. The filled-in user message is combined with the system part of the prompt and submitted as a prompt to an LLM, which is expected to generate a completion.

```
Student Message: {{issue_description}}

Student Code: {{code}}

Student Error: {{error}}

The {{issue_eq}}% of the described student message is copied from course assignment instructions or code.
```

A.3 User Message Example

An example of filled-in user message template is shown below.

```
Student Message:
I'm trying to index and find the value associated with the index label ('music') but it keeps saying
this. How do I make it at least not give me an error?

Student Code:
major_series.sort_index['MUSIC']

Student Error:
'method' object is not subscriptable

The 12.65% of the described student message is copied from course assignment instructions or code.
```