Towards a Comprehensive AI Teaching Assistant Based on Course Forums

Bowen Gu
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
gubowen2@live.unc.edu

Hao Wang
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
harrywh@live.unc.edu

Kaizhuo Chen
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
kaizhuo@email.unc.edu

***Abstract*—Educational forum posts processing is a demanding and time-consuming task for instructors. In this paper, we present the kernels of a system based on natural language processing. Together with future implementation of the database, receiving and sending module, and user interface, the system can be served as an intelligent teaching assistant for educational forum posts processing for small courses, which aims at alleviating the workload of instructors. Currently, this system is capable of conducting the following tasks: (a) Automatically label new posts into pre-designed categories. (b) Identify posts whose content is similar or identical to another existing post. (c) Identify posts that do not contain enough information for the instructors to understand or answer the question. (d) Automatically answer students' questions about the implementation of coding assignments using English translation of the solution code. We applied six language models, four prediction models, together with two API packages to build the system. Our result showed that the system reached the highest accuracy of over 90% on task (a), over 98% on task (b), and over 95% on task (c), which shows the bright prospect of extending it to a fully functional and practical plug-in.**

***Index Terms*—natural language processing, machine learning, artificial intelligence, data processing, software engineering, education**

1. Introduction

Educational forums are useful and important auxiliaries that help students with their academic performance and assist instructors by enhancing course material delivery. With educational forums, students are free from the conventional teaching-learning style by posting questions on the forum anytime and anywhere. On the other hand, instructors can answer students' questions outside the classroom and can post important notes to notify and share students with important course materials.

To help alleviate instructors' burden to manage and monitor the forums and further extend educational forums functionality to help students with learning, we presented our AI Teaching Assistant (AITA), which is a multi-functional teaching assistant that helps the instructors with post processing. Our AITA is semi-automatic, which means that it needs to contact the instructor before making critical decisions.

The completed AITA will have the following functionalities. (a) Identify "active" [32] post questions and reply by requesting for more thinking from the student by providing the student with specific aspects that the student needs to elaborate more on (b) Identify "constructive" [32] post questions and notify human TAs to answer the question by e-mail. (c) Identify duplicated posts and reply with the link to the previous posts that are similar (d) Identify office hours requests posts that do not specify the meeting time and reason with the instructor and reply by giving students instructions about how to revise their requests. (e) Reply with a line by line English translations of the solution code to the student who asks the implementation of a specific method or function. (f) Identify "content clarification" post questions and notify the professor to provide answers to the questions by e-mail. (g) Generate an instructor report at any time during the semester that shows the number and type of questions that each student has posted since the beginning of the semester, the prediction of each student's final grade, and the prediction of each student's course drop probability. For tasks (a), and (c) to (e), the AITA will send its proposed reply to the instructor and ask for confirmation before it post the response. If confirmed by the instructor, the AITA will directly post its reply on the forum. Otherwise, the instructor can edit the reply before it gets post or cancel the reply. The instructor can also choose to trust the AITA and make some functionalities fully-automated without granting confirmation from the instructor. For tasks (f) and (g), they are designed to be instructors only and are only accessible to the instructors.

The completed AITA will contain the following major components: (a) A database that helps store and manipulate posts in different pre-designed categories. (b) A kernel that consists of different modules, in which different machine learning models are encapsulated and used to perform different tasks mentioned in the paragraph above. (c) A connection module that fetches new posts and sends messages generated by the kernel to target students or instructors. (d) A user interface that makes the interaction between the instructors and the system easier.

This thesis will focus on the implementation of the

kernel, which contains the following four modules. (a) The automatic labeling system, which labels the new posts into the different pre-designed categories. (b) The duplicate post identifier, which is used to identify duplicate posts. (c) The incomplete office hours requests identifier, which is used to identify office hours requests that do not specify the time or reason components. (d) The automatic assignment assistant, which is designed to help students with questions related to code implementation.

For the seven functionalities mentioned above, the automatic labeling system serves as the basis for functionality (a), (b), (f), and (g) since all these functionalities requires identification of posts of certain categories. The duplicate posts identifier is used for functionality (c). The incomplete office hours requests identifier is used for functionality (d). The automatic assignment assistant is used for functionality (e).

The organization of the paper will be as follows: Section 2 will survey the work related to this research. Section 3 will focus on describing the major techniques and tools we used to build different models in each module. Section 4 will show our detailed steps of building each model and the reasons behind our decisions. Section 5 will show and analyze the result of each completed model. Section 6 will be the conclusions and directions for future work.

2. Related Work

2.1. AI Teaching Assistant

An advanced intelligent teaching assistant can help instructors in a variety of ways from monitoring the forum activities to automatically answer students' questions. Many groups of researchers are developing the AI teaching assistants due to its broad application prospects. Jill Watson, an AI teaching assistant designed to automatically answer students' questions on educational forums, had achieved great success since its first appearance in 2016. Jill Watson was developed using the IBM Watson platform by the research teaming led by Dr. Ashok Goel in Georgia Tech. Jill Watson was fully-automated and was able to provide informative and coherent answers to students' questions, which provided quick and useful help to students and saved instructors' time. Despite the satisfying performance, Jill Watson was uni-functional and was only capable for question answering. Since a human teaching assistance's job also involves post processing and many of the post processing procedures can be automated, this shows the importance of the development of a multi-functional post processing AI teaching assistant which serves as a complement of Jill Watson.

2.2. Forum Posts Classification

The classification of Educational Forums is critical not only because the classification schemes chosen can tell instructors about different aspects of students' learning progress, but also because it can serve as the training data for automatically categorizing posts into these pre-designed schemes. Do [1] presented a classification scheme that split the posts in educational forums into two categories and eight subcategories to not only split whether the posts are logistic or not but into the depth of what topics of students' and instructors' interest that the post is related to. Vellukunnel et al [32] present another classification scheme that is used to help instructors understand how they can best support students in learning computer science through educational forums. Besides, Chaturvedi et al [33] propose several prediction models designed to capture unique aspects of MOOCs, combining course information, forum structure, and post contents to identify situations in which instructor intervention is warranted. It is clear that by analyzing educational forums through different classifications, instructors can have a detailed idea about students' performance and the problems they have with the course.

2.3. Duplicate Posts Identification

Duplicate posts identification system is essential to educational forum management, especially with the ones that have a large number of users and posts since it alleviates the instructors' and forum managers' workload and helps students get the answers more quickly without creating another post. Zhang et al [34] propose a methodology designed for the Programming community-based question-answering (PCQA) domain to detect duplicate questions. Ahasanuzzaman et al [35] performed a manual investigation to understand why users submit duplicate questions in educational forums like Stack Overflow and proposed a classification technique that can identify duplicate questions with reasonable accuracy. Despite the work that has been done to identify duplicate posts, most of them concentrated on large educational forums such as Stack Overflow, which shows the deficiency of work on identifying duplicate posts for smaller educational forums like Piazza.

2.4. Natural Language Processing on Programming Language

The relation between natural language and programming language is one of the hottest research topics in recent years. Hindle et al [36] suggested that "natural" code in repositories is highly repetitive, which can be captured by language models developed in the field of statistical natural language. Since then, a great amount of work has been done in using language models to help code writing. For example, Ray et al [37] suggested entropy may be a valid, simple way to complement the effectiveness of debuggers since code with bugs tends to have higher entropy. As the computing power grows, language models with stronger capability were introduced, and more focus was put on using natural language to summarize code. LeClair et al [40], Haque et al [39], Ahmad et al [41], and Bansal et al [38] proposed their improved code summarization model using graph neural network, attention mechanism, Transformer,

and project-level encoder respectively. One of the products that lead the trend in this area is the Codex models [31], which aim not only at the summarization of code but also at translating natural language into different programming languages. These work shown above are important to our work since it can help automate reply to students' questions about particular sections of an assignment which is related to code. While direct sharing of the solution code is not allowed, a hint based on the translation of the solution code can help students proceed with their assignments without going to office hours, which helps reduce the workload of instructors.

3. Techniques and Tools

3.1. Language Models

Existing machine learning methods often cannot directly process text data, so it is necessary to find an appropriate method to transform text data into numerical data. To apply machine learning to natural language processing, language models are introduced. The key idea of these models is to map the context into numerical vectors of certain dimension [2], namely an array that contains a certain number of elements, where the value of each element indicates the significance of the semantic meaning of the text in one direction. Therefore, these word vectors can be regarded as features of the context and served as learning materials of the machine learning model. The dimension of the word vector depends on the language model used. In the following subsections, we will introduce, in time sequence, six popular and influential language models that we used to get embedding features in our research.

3.1.1. Word2Vec. Word2Vec [12] [13] is a revolutionary open-source toolkit developed by Google in 2013 for obtaining word vectors, which greatly simplified the distributed representation method proposed by Mnih and Hinton [14] in 1986. This optimization shortened the training cost greatly, which made the word vector application in the industry possible. Before Word2Vec, people often used one-hot encoding, which is a sparse matrix, to represent words when it comes to natural language processing problems. This will cause a lot of trouble in the calculation since (a) The dimension of the word vector will increase with the number and type of words. (b) Any two words are isolated from each other and cannot represent relevant information between words at the semantic level.

Word2Vec solved the above two problems about one-hot coding by (a) Changing each element of the vector from an integer to a float, which enables each element of the word vector to present the entire range of real numbers. As a result, the original large dimensional sparse matrix is compressed into a much smaller dimensional space. (b) Embedding semantic meaning into vector space through context so that each dimension of the word vector now represents certain semantic meaning.

The above innovation made Word2Vec a useful and popular tool and greatly promoted the development of natural language processing. However, it still needs improvement. One of the major flaws of Word2Vec is that it does not consider the sequential relationships between words, nor does it consider the statistical information of the whole corpus as it is a prediction-based word embedding method. Its process of modeling semantics through context also makes it lose a lot of semantic and syntactic information. These deficiencies leave a lot of room for further word embedding models.

3.1.2. GloVe. GloVe [6] is one of the most commonly used word embedding language models developed by Stanford NLP Group in 2014. GloVe stands for Global Vectors for Word Representation. It combines the Global statistics of matrix decomposition in Latent Semantic Analysis (LSA) [15] and the advantages of local Context Window [8], which overcomes the difficulty that Word2Vec faced about losing semantic and syntactic information. Moreover, GloVe's integration of global prior statistics can speed up the training of the model even more and enables the control of relative weight of words.

The realization of GloVe is divided into the following three steps: First, constructing a co-occurrence Matrix based on the corpus. Second, constructing the approximate relationship between the word vector and the co-occurrence Matrix. Third, construct the loss function.

The major problem of GloVe is the loss function it applies. According to the definition of its loss function, any constant vector added to the loss function is still the solution to that loss function. This will make the word vectors meaningless if the same constant vector is added continuously since it makes all word vectors very similar to each other regardless of what the original word vectors are. In practice, the length of GloVe's stopword vectors is much longer than the corresponding Word2Vec's stopword vectors. If these stopwords are not filtered out in advance, GloVe's performance will decrease.

3.1.3. BERT. BERT [3], which stands for Bidirectional Encoder Representation from Transformers, is a word embedding language model proposed by Google AI Research Institute in October 2018. Since it was introduced, BERT showed astonishing performance in various natural language processing tasks. Specifically, BERT showed amazing results in SQuAD1.1, a Top-level test of machine reading comprehension, in which its performance surpassed human baseline in all two measurable indicators. Also, it improved the GLUE benchmark from 72.8% to 80.4% and achieved a MultiNLI accuracy of 86.7%, compared to the highest accuracy of 81.1% achieved by traditional word embedding models [3].

BERT's amazing performance originates from its revolutionary design and model training methods, which broke the convention of traditional word embedding models. Before BERT, the latest language model that was used in natural language processing was GPT-1 [16], which used previous words to predict target words. This unidirectional design

failed to utilize information after the target words. BERT managed to solve this problem by proposing a bidirectional learning model, whose prediction was based on the context both above and below the target words. To make this possible, BERT implemented a mask language model, which was inspired by cloze. This model covered 15% of the words in a text and then predicted the covered words with other words. The covered words were replaced with the symbol [mask]. To reduce the impact of this symbol, among the 15% of the covered words, 80% of them were covered with [mask] symbols, 10% of them remained the original words, and the last 10% were replaced with new random words. The above decision was purely based on probability.

The pros and cons of BERT are both prominent. Since BERT is a bidirectional language model, it has strong language representation and feature extraction ability. On the other hand, since BERT involves probability when dealing with the masked words, it is very difficult for individual computers to replicate the performance of BERT. Also, since BERT's inner structure is highly complicated, it requires very strong computing power to train the model. As a result, it is only feasible to fine-tune the pre-trained model based on different natural language processing tasks.

3.1.4. RoBERTa. RoBERTa [7] is an advanced word embedding language model based on BERT introduced by Facebook AI in 2019. RoBERTa can be regarded as an optimized BERT model, which makes it more robust.

RoBERTa's optimization focused on the following parts. First, it enlarged the corpus that BERT was trained from 16 GB to 160 GB. Second, it increased the batch size of the BERT model from 256 to 2048 and 8192. Third, it increased the pre-train time and training sequence of the model.

RoBERTa can outperform BERT in various natural language processing tasks, but it is mainly due to its larger corpus and longer training time instead of the design innovation. Also, RoBERTa is more demanding on computing power than BERT, which makes training the model more unfeasible.

3.1.5. DeBERTa. DeBERTa [4] is one of the most advanced word embedding language models available nowadays. It was first introduced by the Microsoft NLP group in January 2021. It was the first model whose overall performance surpassed the human baseline in SuperGLUE [17], a stickier benchmark for general-purpose language understanding systems. Comparing to BERT and RoBERTa, DeBERTa applied the following three new techniques.

First, the split-attention mechanism [4]. Different from BERT and RoBERTa, each word in DeBERTa is represented by two vectors encoding its content and position respectively, and the decomposition matrix is used to calculate the attention weight between words according to their content and relative position respectively. The reason for this approach is that the attention weight, which is a measure of the strength of word-to-word dependencies, of word pairs depends not only on their content but also on their relative position. For example, the words "deep" and "learning" are much more dependent when they appear back-to-back in the same sentence than when they appear in different sentences.

Second, the enhanced mask decoder [4]. Like BERT and RoBERTa, DeBERTa was pre-trained using mask language modeling (MLM). DeBERTa applies the content and location information of context words to MLM. The split-attention mechanism already takes into account the content and relative position of context words, but not the absolute position of those words, which is crucial for prediction in many cases. For example, in the sentence "A new store opened beside the new mall", "store" and "mall" are masked when used for prediction. Although the local contexts of the two words are similar, they play different syntactic roles in sentences such as the protagonist of a sentence is "store" rather than "mall". These syntactic nuances depend largely on the absolute position of the word in the sentence, so it is important to consider the absolute position of the word in the language modeling process. DeBERTa incorporated absolute word location embedding before the SoftMax layer, in which the model decoded the masked words according to the aggregated context embedding of word content and location.

Third, the scale-invariant-fine-tuning (SiFT) [4]. Virtual adversarial training is a regularization method to improve model generalization. It does this by improving the robustness of the model to adversarial examples, which are created by subtle interference with the input. The model is regularized so that when given a sample of a particular task, the model produces the same output distribution as that produced on the adversarial version of the sample. For the natural language understanding task, interference is used for word embedding instead of the original word sequence. However, the range of values of embedded vectors varies from word to word and model to model. For a large model with billions of parameters, the variance will be large, resulting in the instability of adversarial training. Inspired by layer normalization, DeBERTa developed a scale-weiden-fine-tuning (SiFT) method to improve the training stability, which applies interference to normalized word embedding.

Due to the above innovation, DeBERTa outperformed RoBERTa with only about half of its corpus size (78 GB VS 160 GB). DeBERTa is also currently an active model which is improving by the Microsoft NLP group. In November 2021, the group proposed its third generation DeBERTa model, which improves the original DeBERTa by using the RTD training loss and a new weight-sharing method [5].

3.1.6. GPT-3 and Its Ancestors. Generative Pre-trained Transformer (GPT) [16] is a fully generative pre-trained language model developed by OpenAI that can achieve amazing results in complex NLP tasks such as article generation, code generation, machine translation, and Q&A. These tasks do not require supervised learning to fine-tune the model. For a new task, GPT needs very little data to understand the requirements of the task and achieve the performance that approaches or exceeds state-of-the-art, which stands for the newest technology and ideas in the development of a product.

The training of the GPT model requires a large training corpus, a large number of model parameters, and supercomputing resources. The model structure of the GPT series inherits the idea of continuously stacking Transformer [28], and completes the iterative update of the GPT series by continuously improving the scale and quality of the training corpus and the number of network parameters. GPT also proves that the power of the model can be continuously improved by increasing the model capacity and corpus scale.

GPT-1 [16] was introduced in June 2018 with 117M parameters and about 5GB corpus, which was smaller in scale compared to the contemporary BERT-large model's 340M parameters and about 16GB corpus.

Traditional NLP models often use a large amount of data to conduct task-related model training for supervised models, but this supervised learning task has two disadvantages: First, a large amount of labeled data is needed, and high-quality labeled data is often difficult to obtain since, in many tasks, labels are not unique or do not have clear boundaries. Second, a model trained on one task is difficult to generalize to other tasks, thus it can only be called a "domain expert" rather than a linguist of general-purpose.

GPT-1 tried to solve the above problems of traditional NLP models by proposing its generative pre-training technique, which uses unsupervised learning to control the pre-training objectives of the supervised model. The idea of GPT-1 is to tackle supervised tasks by learning a generative language model on unlabelled data and then fine-tuning it for specific NLP tasks such as natural language inference, question answering commonsense reasoning, semantic similarity, and classification.

According to the experiment results, GPT-1 outperformed the state-of-the-art NLP model in 9 of the 12 tasks with supervised learning. The GPT-1 model is also more stable than the LSTM-based model in zero-shot tasks, which are tasks that the learning model observes samples from classes that were not observed during training, and needs to predict the class they belong to. The performance of GPT-1 gradually improves with the increase of training times, indicating that GPT-1 has very strong generalization ability and can be used in other NLP tasks. GPT-1 proves the transformer's powerful ability to learn word vectors, and learning downstream tasks based on the word vectors obtained by GPT-1 can make downstream tasks achieve better generalization ability. For training on downstream tasks, GPT-1 often achieves excellent results with simple fine-tuning. GPT-1 also made some progress on untuned tasks, but its generalization ability was much lower than that of fine-tuned supervised tasks, suggesting that GPT-1 is still a simple domain expert rather than a general-purpose linguist.

GPT-2 [26] was introduced in February 2019 with 1.5B parameters and about 40GB corpus, which was similar in scale compared to the contemporary RoBERTa-large model's 355M parameters and about 160GB corpus.

GPT-2 aims to train a word vector model with stronger generalization ability. It does not make too much structural innovation compared to GPT-1 network but uses more network parameters and larger data sets. The key idea of GPT-2 is that any supervised task is a subset of the language model. When the capacity of the model is very large and the amount of data is rich enough, other supervised learning tasks can be completed only by training the language model. As a result, an unsupervised pre-training model based on large enough training data can be used to perform any supervised NLP tasks even without training in these specific categories.

The greatest contribution of GPT-2 is to verify that word vector models trained from massive data and large numbers of parameters can be transferred to other types of tasks without additional training. But many experiments have shown that GPT-2's ability to learn unsupervised has a lot of room for improvement, and even performs no better than random on some tasks. While the GPT-2 has performed well in some zero-shot missions, it is still not clear how well the GPT-2's strategy will work. GPT-2 indicates that there is room for further development of its potential as the model capacity and data volume increase. Based on this idea, GPT-3 was born.

GPT-3 [27] was introduced in May 2020 with 175B parameters and about 45TB corpus, which was much larger in scale compared to the contemporary DeBERTa-large model's 390M parameters and about 78GB corpus.

One of the innovation that GPT-3 had comparing to GPT-2 is the In-context learning [24] [25], which is the inner loop of meta-learning [29] [30]. The core idea of meta-learning is to find an appropriate initialization range through a small amount of data so that the model can be fitted quickly on a limited set of data and get good results.

Due to its huge number of parameters and gigantic corpus, GPT-3 is by far the most powerful language model available. According to the experiment results, GPT-3 outperforms most zero-shot or few-shot state-of-the-art methods in a large language model dataset. GPT-3 also outperforms other fine-tuned state-of-the-art language models in many complex NLP tasks, such as closed-volume question answering, pattern parsing, machine translation. In addition to the above traditional NLP tasks, GPT-3 also achieved impressive results in several other areas such as mathematical addition, article generation, and writing code.

Despite its achievement, GPT-3 is not a panacea for all NLP tasks. For some questions with meaningless propositions, GPT-3 will not judge whether the propositions are valid or not, but will fit meaningless answers. Also, due to the existence of its massive corpus, it is difficult to ensure that the articles generated by GPT-3 do not contain some very sensitive content, such as racial discrimination, gender discrimination, and religious prejudice. Furthermore, due to the transformer's modeling capabilities, GPT-3 does not guarantee the consistency of a long article or book, and there is the problem of repeating the above paragraphs in subsequent paragraphs.

3.2. Prediction Models

After getting embedding features from the language models, prediction models need to be built to train on

the embedding features and perform multiple tasks in our research. In the subsections below, we will introduce two different categories of prediction models we built for the research: neural network and decision tree.

Neural network (NN) [18] is a computational model that imitates the structure and function of the biological neural network. In most cases, an artificial neural network can change the internal structure based on external information, which is an adaptive system. In a standard neural network, each node represents a specific output function, which is called an activation function. Each connection between two nodes represents a weighted value for the signal passing through the connection, which is called the weight. The output of the network varies according to the connection mode, weight, and activation function of the network. A modern neural network is a nonlinear statistical data modeling tool, which is often used to model the complex relationship between input and output or to explore data patterns. In this paper, we presented two different types of neural networks: convolutional neural network (CNN) and long short-term memory (LSTM).

Decision tree [43] is a prediction model, which represents a mapping between object attributes and object values. Each node in the tree represents an object, each bifurcated path represents a possible attribute value, and each leaf represents a value for an object along the path from the root to the leaf. The decision tree only has a single output. If a complex output is desired, an independent decision tree can be established to deal with different outputs. In this paper, we presented two different types of decision trees: random forest classifier and XGBoost.

3.2.1. Convolutional Neural Network (CNN). Convolutional Neural Network (CNN) [19], a kind of artificial neural network, is very popular in the field of speech analysis and image recognition. Its weight-sharing network structure makes it more similar to the biological neural network, which reduces the complexity of the network model and the number of weights. This advantage is more obvious when the input of the network is a multi-dimensional image so that the image can be directly used as the input of the network, avoiding the complex process of feature extraction and data reconstruction in the traditional recognition algorithm. The Convolutional network is a multi-layer perceptron specially designed for two-dimensional shape recognition. This network structure is highly invariant to translation, scale, tilt, or syntactic deformation.

CNN is influenced by the early delayed neural network (TDNN) [20], which reduces learning complexity by sharing weights in the time dimension and is suitable for speech and time-series signal processing.

CNN is the first truly successful learning algorithm for training multi-layer network structures. It uses space relation to reduce the number of parameters to be learned to improve the training performance of the general forward BP algorithm. CNN is proposed as a deep learning architecture to minimize data preprocessing requirements. In CNN, a small part of the image (local perception area) serves as the input of the lowest layer of the hierarchy, and the information is then transmitted to different layers in turn. Each layer obtains the most significant features of the observed data through a digital filter. This approach can capture salient features of observations invariant to translation, scaling, and rotation because the local sensing area of the image allows neurons or processing units to access the most basic features, such as oriented edges or corners.

Although CNN is mainly used for image processing, it is sometimes used in natural language processing since it has a lower probability to overfit the data and runs quicker, which sometimes results in better performance.

3.2.2. Long Short-Term Memory (LSTM). Long Short-Term Memory (LSTM) [23] is a temporal recursive neural network, which is suitable for processing and predicting important events with relatively Long intervals and delays in time series. LSTM is a special kind of recurrent neural network (RNN) [21].

RNN solves the information preservation problem by using the information mentioned above to judge and understand the current word. However, RNN has the problem of long-term dependence. Specifically, RNN cannot well preserve the information from a long time ago. With the increasing time interval, the RNN network will gradually lose the ability to learn information far away, which is called the vanishing gradient problem [22].

To solve this problem, LSTM was introduced. Due to its carefully designed "gate" structure, LSTM has the ability to eliminate or add information to the cellular state, which allows LSTM to remember long-term information better.

Due to the feature introduced above, LSTM becomes a typical neural network used in natural language processing and has achieved great performance in practice.

3.2.3. Random Forest Classifier. Ensemble learning [45] is a learning method that generates many classifiers and aggregates their prediction results. There are two categories of ensemble learning: bagging and boosting. In boosting, successive trees give extra weight to points incorrectly predicted by earlier predictors. In the end, a weighted vote is taken for prediction. In bagging, successive trees do not depend on each other since each of them is independently constructed using a bootstrap sample of the data set. In the end, a simple majority vote is taken for prediction.

Random forest [44] is a modified algorithm based on bagging, which uses Classification And Regression Trees (CART) decision trees [46] as successive trees. Random forest selects samples and features randomly to prevent overfitting.

The algorithm can be summarized as the following steps. First, N samples are selected from the sample set with back samples. Second, K features were randomly selected from all features, and then the optimal segmentation feature was selected as the left and right subtrees of the decision tree to establish the CART decision tree. Third, repeat the above two steps M times, which establishes M CART decision trees. Finally, these M CART trees form a random forest,

and the results are voted to determine which category the data belongs to.

The random forest classifier is relatively simple to implement and is insensitive to partial feature loss. Besides, its training can be highly parallel, which has advantages for the training speed of large samples in the era of big data. Also, since the decision tree nodes can be randomly selected to divide features, the model can still be trained efficiently when the sample feature dimension is very high and the variance of the trained model is small and the generalization ability is strong. However, the random forest classifier tends to overfit some noisy sample sets, and features with more value division tend to have a greater impact on its decision-making, thus affecting the effect of the model.

3.2.4. XGBoost. XGBoost [42] is a modified algorithm based on boosting. Like other decision tree algorithms, the key idea of the algorithm is to keep adding trees by doing feature splitting. When getting K trees after training, the value of a sample needs to be predicted. According to the characteristics of the sample, a corresponding leaf node will fall into each tree, and each leaf node will correspond to a value. Finally, the value of the sample is predicted by adding up the scores corresponding to each tree.

XGBoost's uniqueness is its target function. Also, to prevent the tree from growing too deep, a threshold value is added so that only when the gain is greater than this threshold value can splitting be carried out. Otherwise, the tree stops growing.

XGBoost is a very useful tool in machine learning and has been widely used in the industry. It has a variety of advantages. First, it prevents data overfitting by a variety of means such as regularized items and shrinkage and column subsampling. Second, it supports parallelization, cross validation, and early stop, which speeds the training up. Third, sample weights can be set and changed during training, which enables the model to focus more on specific parts of the sample.

3.3. API

API is a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other services.

In the following subsections, we will introduce two APIs we used as part of the modules in our research, which greatly improve the performance of our models.

3.3.1. IBM Watson Natural Language Understanding API. The IBM Watson Natural Language Understanding API is a useful context analyzing tool. By inputting a paragraph, the API can identify and determine the following from the paragraph: (a) key entities such as organization names and places and their sentiment scores (positive or negative), (b) keywords and their sentiment scores (positive or negative), (c) topics that the paragraph may relate to, (d) the relations between entities, (e) sentiment scores (positive or negative) of the whole paragraph, the entities, and the keywords, (f) emotion (sadness, joy, fear, disgust, or anger) of the whole paragraph, the entities, and the keywords, (g) the subjects, actions, and objects in the paragraph, (h) the nouns, verbs, adjectives, adverbs, and pronouns in the paragraph.

The IBM Watson Natural Language Understanding API can be accessed both through its online self-service user interface and through server requests sent by programs using different kinds of programming languages. If using the latter method, a JSON file that contains the analyzing results specified by the user will be returned.

3.3.2. OpenAI API. The OpenAI API is a practical multi-functional tool, which can be applied to virtually any task that involves understanding or generating natural language or code. It offers a spectrum of GPT-3 [27] models, including the most recently introduced Codex series, with different levels of power suitable for different tasks, as well as the ability to fine-tune users' custom models. These models can be used for everything from content generation to semantic search and classification.

The OpenAI API can be accessed both through its online self-service user interface and through server requests sent by programs using different kinds of programming languages. If using the latter method, a JSON file that contains the result of the tasks specified by the user will be returned.

4. Experiment Methods and Steps

4.1. Automatic Labeling System

4.1.1. Data Collection. The data we used in this section consists of 7,279 Piazza posts from 13 courses offered by the computer science department of the University of North Carolina - Chapel Hill. The details about the name of the courses, the year and semester that the course was taught, and the number of posts in each course is shown in Table 1, where "F" in the "Semester" column of the table stands for "Fall", and "S" stands for "Spring".

In the "Course Title" column, *Foundations of Programming* (tag: FP) is an elementary course where students will learn how to reason about how their code is structured, identify whether a given structure is effective in a given context, and look at ways of organizing units of code that support larger programs. *Computer Organization* (tag: CO) is an intermediate course where students will be introduced to the conceptual design of a basic microprocessor, along with assembly programming. The course includes fundamental concepts such as binary numbers, binary arithmetic, and representing information as well as instructions. Students will learn to program in assembly (i.e., machine) language. The course covers the fundamentals of computer hardware design, transistors, and logic gates, progressing through basic combinational and sequential components, culminating in the conceptual design CPU. *Modern Web Programming* (tag: MW) is an upper-level course where students will develop applications for the World Wide Web including both client-side and server-side programming. The course

TABLE 1. DETAILED INFORMATION ABOUT COURSES USED FOR AUTOMATIC LABELLING SYSTEM

Tag	Course Title	Semester	Post
FP1	Foundations of Programming	F 2012	331
FP2	Foundations of Programming	F 2013	480
FP3	Foundations of Programming	F 2015	527
FP4	Foundations of Programming	F 2016	384
FP5	Foundations of Programming	F 2017	571
FP6	Foundations of Programming	F 2018	362
CO1	Computer Organization	S 2019	891
CO2	Computer Organization	F 2019	533
CO3	Computer Organization	S 2020	440
CO4	Computer Organization	S 2020	236
CO5	Computer Organization	F 2020	561
MW1	Modern Web Programming	F 2019	1028
MW2	Modern Web Programming	F 2020	935

emphasis is on Model-View-Controller architecture, AJAX, RESTful Web services, and database interaction.

The post data was collected by Do, J. [1]. There are three kinds of posts: questions, notes, and polls. For notes and polls, we labeled them as "Note" and "Poll", respectively. For questions, we manually labeled them into the following categories proposed by Chaturvedi et al [33]. The name, definition, and example of each category are shown in Table 2. All COMP 401, COMP 411, and COMP 426 courses are labeled by the second author, first author, and the third author, respectively.

During labeling, we found that some posts, though selected by the student as a question, do not actually contain a question. For example, some students choose to use a question post to inform others about some hints of the assignment that others may find useful, or some students delete the original post body after their questions are solved. When we encountered these cases, we labeled the posts as "None" and removed them when generating the training data for the prediction models.

4.1.2. Data Cleaning. We then input the data we have in 4.1.1 to our data cleaning function. The function cleans the data using six different methods. A detailed explanation of the methods can be found in Table 3. Each method mentioned in Table 3 except *Clean* is optional and can be switched on and off according to the performance of the prediction model.

4.1.3. Feature Engineering. After obtaining the cleaned post data, we extracted features from the posts. Here, the extracted features are the word vectors generated after embedding the posts. We embedded the posts using the language models mentioned in section 3.1. All language models mentioned in section 3.1, except the GPT-3 model mentioned in 3.1.6, are word embedding models, which map a word into a certain dimension vector space. To utilize these models, we firstly tokenized the post by words. Specifically, for the Word2Vec model, we used file *GoogleNews-vectors-negative300.bin*. For the GloVe model, we used file *glove.840B.300d*. These two files contain the result of the pre-trained Word2Vec and GloVe model, which is a dictionary that maps most common words to a 300-dimension word vector. Both files can be obtained from public online sources. Since we do not directly let the model calculate the word vector but try to find the vector we want in the dictionary, there exist cases that some words are not in the dictionary. For each word that is not in the dictionary, we create a 300-dimension word vector whose elements are all 0s, meaning that they contribute nothing to the semantic meaning of the whole post. For BERT, RoBERTa, and DeBERTa, we used the *bert-large-uncased-whole-word-masking*, *roberta-large*, and *kamalkraj/deberta-v2-xlarge* provided by the *transformers* package, respectively, which are the most capable models in these model series. All the three models above can be found in *Hugging Face (https://huggingface.co)*. For these models, the word vectors are calculated by the pre-trained language models by calling their embedding function respectively instead of extracted from a pre-made dictionary, so all words will be embedded. The dimension of the BERT, RoBERTa, and DeBERTa word vectors are 1024, 1024, and 1536, respectively.

The embedding of the whole post is another vector that has the same dimension as each word vector, and each of its elements has the value of the sum of all word vectors at that index. For example, suppose a post has only two words "Hello World" and the word "Hello" is embedded to vector

$$[a_0, a_1, ..., a_{299}], \tag{1}$$

and the word "World" is embedded to vector

$$[b_0, b_1, ..., b_{299}]. \tag{2}$$

Then, the embedding of the whole post is calculated to be vector

$$[a_0 + b_0, a_1 + b_1, ..., a_{299} + b_{299}]. \tag{3}$$

This makes sense since word vectors can be treated as mathematical vectors and the addition of the word vectors stands for the addition of the corresponding semantic meaning of each component in the word vector.

For GPT-3, due to its advanced implementation, it can embed the whole post directly, so there is no need to conduct word vector addition for the result returned by the GPT-3 model. Since it is developed by OpenAI, we obtained the embedding by using the OpenAI API embedding model. Specifically, we connected to the OpenAI server, used the *openai.Embedding.create* function with our private API key and the post content we need to embed. After the serve processed our request, a JSON file that contains the embedding result will be returned. We then extract the "embedding" part of the JSON file to get the embedding result. We used the *Davinci* engine, which is the most capable embedding engine provided. The result post embedding vector has 12288 dimensions.

TABLE 2. QUESTION CLASSIFICATION SCHEME

Type	Definition	Example
Active	Request for help that does not display reasoning and is not framed in what the student knows or has already tried.	Can somebody explain the Machine practice problem 1, 2 and 3?
Constructive	Questions that reflect students' reasoning or attempts to construct a solution to the problem.	I am trying to extract a certain number of bits from the address and store them as the index. I have tried using modulus by 16(# of bits to be extracted) but that did not work. Does anyone have any suggestions? thx
Logistical	Questions on course policies, schedules, and assignment submission mechanisms. Not necessarily related to computer science content.	Any chance an LA could hold office hours tomorrow? The only available times have been cancelled.
Content Clarification	Request for additional information on project assignments, automated software tests, and design documents. Does not involve a question on the student's own problem-solving work. (If a question does involve the student's own work, that question would either be labeled Active or Constructive.)	Does the poster need to be a foam board poster, tri-fold poster, or just a poster that can be rolled up?

TABLE 3. DATA CLEANING METHODS

Method Name	Functionality	Example String Before Cleaning	Example String After Cleaning
Clean	The default clean strategy which removes special characters such as markup tags that help display certain styles of characters or formulas.	¡/p¿n¡p¿Below is my code for "my assignment&#34¡/p¿n¡p¿.	Below is my code for my assignment.
Hyper Clean	Unify and correct spelling of the same meaning words	I have some imrovement in this cs class.	I have some improvement in this computer science class.
Remove Punctuation	Remove all punctuations that appears in the content.	Where should I place my tags???	Where should I place my tags
Remove Stopwords	Remove all stopwords that appear in the content. Here, the range of the stopwords is defined by *nltk.corpus.stopwords*.	It's just a small challenge. Believe in yourself and you'll succeed.	small challenge. Believe succeed.
Use Alternative Stopwords	Remove all stopwords appears in the content. The different from Remove Stopwords is that the range of the stopwords is limit to the following: 'the', 'a', 'an', 'and', 'but', 'if', 'or', 'because', 'as', 'what', 'which', 'this', 'that', 'these', 'those', 'then', 'just', 'so', 'than', 'such', 'both', 'through', 'about', 'for', 'is', 'of', 'while', 'during', 'to', 'What', 'Which', 'Is', 'If', 'While', 'This', which is a subset of the stopwords defined by *nltk.corpus.stopwords*.	It's just a small challenge. Believe in yourself and you'll succeed.	It small challenge. Believe in yourself you will succeed.
Stem Words	Shorten the words to their stem forms.	The technique applied makes it one of the fastest computers in the world.	The technique apply make it one of the fast computer in the world.

4.1.4. Prediction Model Set Up. We set four prediction models using RFC (section 3.2.3), XGBoost (section 3.2.4), CNN (section 3.2.1), and LSTM (section 3.2.2), respectively.

We gathered the 13 CSV files obtained in section 4.1.3 into four different groups: (a) *COMP401*, which contains embedded posts of all COMP 401 classes. (b) *COMP411*, which contains embedded posts of all COMP 411 classes. (c) *COMP426*, which contains embedded posts of all COMP 426 classes. (d) *all*, which contains embedded posts of all classes. In each group, we excluded all posts that were labeled as "Note" and " Poll". This is because the label "Note" and "Poll" can be determined without using a prediction model as these two types are set by the user who posts them and can be seen in the original data. Also, the schema in Table 2 is only applicable to questions, which is our main interest.

To determine the training and test data set, we used the following rule. Both sets can and can only use one keyword among *COMP401*, *COMP411*, *COMP426*, and *All* mentioned in the paragraph above to specify a particular set. After setting the keyword for both sets, if the keyword is the same for both sets, we use *train_test_split* from *sklearn* to split the group into two segments with 80% as the training data and the rest 20% as the test data using $random\_state = 4242$, so that we get the same training and test set splitting is always deterministic. If the keywords are different for both sets and neither of them uses *All* as the keyword, we directly use the group specified by the keyword as the corresponding set with no data splitting. If the keywords are different for both sets and one of them uses *All* as the keyword, for the set that does not use *All* as the keyword, we directly use the group specified by the keyword as the corresponding set with no data splitting. For

the set that uses *All* as the keyword, we use all the posts except those in the group whose keyword is specified by the other set as the corresponding set with no data splitting. This will ensure that there is not overlap between the training and test set.

After setting the training and test data set, we input them into the prediction models. Specifically, for RFC, we used the *sklearn.ensemble.RandomForestClassifier* function with all parameters in their default form mentioned in the online documentation. For XGBoost, we used the *xgboost* package. We set the *objective* parameter to "multi:softmax" to enable multiclass classification, the *num_class* parameter to "4" since we have 4 different labels for question posts. Also, we set the *tree_method* parameter to "gpu_hist" to enable GPU boost, which speeds up the training process. All other parameters are set to their default form mentioned in the online documentation. Since XGBoost only accepts numbers as labels, we converted the original string labels "Active", "Constructive", "Logistical", and "Content-Clarification" to integer labels "0", "1", "2", and "3", respectively. Once the prediction was returned, we reverted the labels to their original string form by rounding the numbers to the closest integer.

For neural network models, since their outputs for classification are always between 0 and 1 for any column, which indicates the probability that the target is in that category, the input matrix of our labels are four columns where each column consists of integers 0 or 1 which indicates if the target is in that category. For example, if one post is in the category "Active", instead of inputting the string label "Active" to the model, we input the matrix "[1, 0, 0, 0]", where the "1" indicates that the post is in the category "Active" and the rest three "0"s indicates that the post is not in the category "Constructive", "Logistical", and "Content-Clarification", respectively. As a result, the shape of our input label matrix is (number of posts, 4). In our code, we used *tensorflow.keras.utils.to_categorical* to help convert the original one column label matrix to the actual input matrix.

After that, we built the CNN and LSTM model. According to our optimization results, we created six convolutional layers with *kernal_size* from 1 to 6. The first four layers has $filters = 128$ and the last two layers has $filters = 32$. All layers use the "same" padding type. For the CNN mode setting, we used $activation =' softmax'$ and $loss =' categorical\_crossentropy'$ to enable multiclass classification, and $metrics = ['acc']$ to let the model evaluate its performance based on the accuracy. For LSTM model, we created one LSTM layer with $dropout = 0.15 + np.random.rand() * 0.25$ and $recurrent\_dropout = 0.15 + np.random.rand() * 0.25$ to removed some neurons from the neural network to prevent trainning data overfit. The setting of *activation*, *loss*, and *metrics* are the same as those of the CNN model.

The output of the neural network models has the same shape as the input label matrix, with a value between 0 and 1 for each element. To revert the output matrix to the corresponding string labels, we choose the column that has the largest number of each row to be the determined label of each post. For example, if the first row of the output matrix, which represents one post, is [0.7, 0.1, 0.2, 0.3], then this post is determined to belong to the category "Active" since the number in the first column is the largest, which stands for "Active". Note that the value of all elements in a row does not sum to one. This is because the model predicts the probability of each category independently.

4.2. Duplicated Posts Identifier

4.2.1. Data Collection. After some trials, we found that the ratio of duplicated posts of a particular Piazza course is much lower compared to normal posts, and it is difficult to find and label them. Moreover, even if we can identify and label all duplicated posts in one course, the duplicated - non-duplicated post distribution will be highly biased, which is unsuitable for training our model. As a result, we use the following alternative strategy. We obtained data from Quora, a popular online forum where people post their questions and answers. After that, we trained our model based on the Quora data, then we applied the trained model to the Piazza forum to see if it is capable to find duplicates of a given post. The data is one CSV file that contains 404,290 questions pairs. Each pair contains two questions and is compared and labeled by human experts, where "0" indicates the two questions are not duplicates, and "1" indicates that they are duplicates.

4.2.2. Data Cleaning. Since we have built a data cleaning tool introduced in section 4.1.2, we cleaned the data in the same way as mentioned in that section.

4.2.3. Feature Engineering. The extracted features for this task include not only word vectors generated by embedding the question pairs, but also other features defined and extracted by our algorithm.

The embedding steps and techniques we used are the same as the ones we described in section 4.1.3. The major difference here is that now we have a pair of questions, namely two posts instead of one. As a result, the embedding result of each question pair will be two vectors of the same dimension, which depends on the language model we used.

Moreover, to improve the performance of our model, we further extracted possible features that may contribute to the accuracy of the model. In total, we extracted 50 different features where each of them is a number that can be used as input for the model. A detailed explanation of the features extracted can be found in Table 4, 5, and 6. Note that features 24 - 35 is similar to features 12 - 23. The only difference is that in features 24 -35, the results can be either positive or negative for the "difference" features, and the results can be either greater or smaller than 1 for the "ratio" features. We added the prefix "Q1 to Q2" to their feature names to indicate that difference. Also, features 36 - 47 all contain "alternative" in their names, which indicates that they use alternative stop words compared to their counterparts. For features 48-50, we used the IBM API mentioned in section 3.3.1 to extract keywords from the questions. Specifically,

we provided the IBM server with our private API key and used the *natural_language_understanding.analyze* from the *ibm_watson* package to analyze and get the response from the server. The response is a JSON file that contains the extracted keywords for each question and is easy to extract.

After extracting the features, we performed Principal Component Analysis (PCA) on the features using the python *sklearn.decomposition* package to remove possible duplicity between the features. As a result, the number of features we used for training is less than those shown in Table 4, 5, and 6.

4.2.4. Prediction Model Set Up. For this task, we used the same four models as mentioned in section 4.1.4. Since this is a relatively large data set, and the question pairs are not sorted by any specific rules, we do not use *train_test_split* from *sklearn*. Instead, we use the first 400,000 rows as the training data and the last 4,290 rows as the test data to ensure the model has enough data to learn and we still have enough question pairs for testing.

For each question pair, we combined its extracted features after being reduced by PCA and the embedded result of the two questions by creating a new array, where the first two elements are the embedded vectors of the two questions respectively, and all the features were appended afterward.

Then we fed the data into the models. For RFC, the only setting that is different from the one mentioned in section 3.1 is the tag of the labels, which is changed to 0 and 1 to indicate duplication. For XGBoost, the *objective* was set to "binary:hinge" to enable binary classification. This setting makes the return value of the classification be either 0 or 1, and further label conversion is not needed. Again, the *tree method* parameter is set to "gpu_hist" to enable GPU boost. For neural networks, since this task is a binary classification, *tensorflow.keras.utils.to_categorical* is not needed and the labels can be directly input into the model. When building the model, textitactivation was set to "sigmoid" and loss was set to "binary_crossentropy", which were the corresponding activation function and loss function used for binary classification. The *kernal_size* and *filters* in CNN and the *dropout* and *recurrent_dropout* in LSTM were set to be the same as in section 4.1.4. The output of the neural network models for each question pair will be a float whose value is in between 0 and 1, which indicates the probability that the questions in the pair are duplicates. We converted the output to the indicated label by rounding the float to its corresponding integer.

4.2.5. Test on Piazza Posts. After finishing training the models, we test them using Piazza posts to see if they can identify duplicates. Specifically, we created different test data sets that fit the format of the models' training data. In each test data set, we changed the first question in the question pair to be the same post we picked specifically from the 7,279 posts mentioned in section 4.1.1 that we thought was possible to be duplicated of other posts. Then we iterated the whole set of posts that have the same label as the first question in the question pair and changed the second question in the question pair to be each one of the posts we found, excluding the first question itself. For example, if there are A, B, C, D, E, and F six posts in the same category (eg. Active, Constructive, Content Clarification, or Logistical), if we wanted the model to identify any posts in that category that is a duplicate of post A, we set every roll of the first question of our test data to be A. The second question will be B, C, D, E, and F, respectively. As a result, the question pair we input into the pre-trained model will be AB, AC, AD, AE, and AF. All the question pairs will be cleaned, embedded, and their features will be extracted as mentioned in section 4.2.2, 4.2.3, and 4.2.3 before being input into the model for prediction. The model will classify each question pair to either 0 or 1, which is the same as what the model did in the original test data set. The label here is the automatically generated label predicted by the model mentioned in section 4.1.4.

To have a clear idea about how confident the model is when making the prediction, we changed the setting for our models mentioned in section 4.2.4 to output the confidence of each question pair being duplicates or non-duplicates instead of giving the final decision. Specifically, for RFC, we used the *predict_proba* function. For XGBoots, we changed the *objective* parameter to "binary:logistic". For neural networks, we no longer rounded the prediction results to integers. Since we did not label the Piazza posts as duplicates and non-duplicates, the duplicates identified by the model, especially those with high confidence, will be an indication of how the model performs.

4.3. Incomplete Office Hours Requests Identifier

4.3.1. Data Collection. While the data set used in section 4.1 has a great number of posts, which is good for training the model, it does not contain office hour request information. To solve this problem, we used another data set that consists of three different computer science courses: COMP 401: Foundations of Programming, COMP 524: Programming Language Concepts, and COMP 533: Distributed Systems. The details about the name of the courses, the year and semester that the course was taught, and the number of posts in each course are shown in Table 7, where "F" in the "Semester" column of the table stands for "Fall", "Sp" stands for "Spring", and "Su" stands for "Summer".

In the "Course Title" column, *Foundations of Programming* (tag: FP) is the same course introduced in section 4.1.1. *Programming Language Concepts* (tag: PL) is an upper-level course where students will learn concepts of high-level programming and their realization in specific languages such as Prolog, SML, and LISP. *Distributed Systems* (tag: DS) is an upper-level course where students will learn topics related to distributed systems such as resource naming, synchronization of distributed processes, consistency and replication, fault tolerance, security and trust, distributed object-based systems, distributed file systems, distributed Web-based systems, and peer-to-peer systems.

Each of the courses in Table 7 has a specific Piazza post where students can post their office hours requests as follow-

TABLE 4. EXTRACTED FEATURES

Feature ID	Feature Name	Definition
1	Non-Stop Word Match Share	The ratio of the shared non-stop word of the question pair to the total number of words of the question pair. Here, the range of stop words is defined by *nltk.corpus.stopwords*.
2	TF-IDF Word Match Share	The ratio of the weight of the shared words of the question pair to the weight of the total number of words of the question pair. The weight of a word is defined as *1 / # of the word occurs in the whole data set*.
3	TF-IDF Non-Stop Word Match Share	The ratio of the weight of the shared non-stop words of the question pair to the weight of the total number of non-stop words of the question pair. The weight of a word is defined as *1 / # of the word occurs in the whole data set*. Here, the range of stop words is defined by *nltk.corpus.stopwords*.
4	Common Unigrams Count	Number of unigrams that the two questions share. A unigram here is defined as any words except stop words defined by *nltk.corpus.stopwords*.
5	Common Unigrams Ratio	Ratio of unigrams that the two questions share to the total unique unigrams that the question pair have. A unigram here is defined as any words except stop words defined by *nltk.corpus.stopwords*.
6	Jaccard	Ratio of the number of shared words between two questions to the total number of unique words that the question pair have.
7	Common Words	Number of shared words in the question pair.
8	Common Non-Stop Words	Number of shared non-stop words in the question pair.
9	Total Unique Words	Number of unique words in the question pair.
10	Total Unique Non-Stop Words	Number of unique non-stop words in the question pair.
11	Same Start Word	0 if the two questions do not start with the same word, 1 otherwise.
12	Word Count Difference	Difference between the total number of words in one question and the total number of words in the other question. The difference here is an absolute value, which is always non-negative.
13	Word Count Ratio	Ratio of the total number of words in one question to the total number of words in the other question. The ratio here is always smaller than or equal to 1, namely that it is always the shorter question that is at the numerator and the longer question that is at the denominator.
14	Unique Word Count Difference	Difference between the total number of unique words in one question and the total number of unique words in the other question. The difference here is an absolute value, which is always non-negative.
15	Unique Word Count Ratio	Ratio of the total number of unique words in one question to the total number of unique words in the other question. The ratio here is always smaller than or equal to 1, namely that it is always the shorter question that is at the numerator and the longer question that is at the denominator.
16	Unique Non-Stop Word Count Difference	Difference between the total number of unique non-stop words in one question and the total number of unique non-stop words in the other question. The difference here is an absolute value, which is always non-negative. The range of stop words is defined by *nltk.corpus.stopwords*.
17	Unique Non-Stop Word Count Ratio	Ratio of the total number of unique non-stop words in one question to the total number of unique non-stop words in the other question. The ratio here is always smaller than or equal to 1, namely that it is always the shorter question that is at the numerator and the longer question that is at the denominator. The range of stop words is defined by *nltk.corpus.stopwords*.
18	Character Difference	Difference between the total number of characters in one question and the total number of characters in the other question. The difference here is an absolute value, which is always non-negative.
19	Character Length Ratio	Ratio of the total number of characters in one question to the total number of characters in the other question. The ratio here is always smaller than or equal to 1, namely that it is always the shorter question that is at the numerator and the longer question that is at the denominator.
20	Unique Word Character Difference	Difference between the total number of characters of unique words in one question and the total number of characters of unique words in the other question. The difference here is an absolute value, which is always non-negative.
21	Unique Word Character Ratio	Ratio of the total number of characters of unique words in one question to the total number of characters of unique words in the other question. The ratio here is always smaller than or equal to 1, namely that it is always the shorter question that is at the numerator and the longer question that is at the denominator.
22	Unique Non-Stop Word Character Difference	Difference between the total number of characters of non-stop unique words in one question and the total number of characters of non-stop unique words in the other question. The difference here is an absolute value, which is always non-negative. The range of stop words is defined by *nltk.corpus.stopwords*.
23	Unique Non-Stop Word Character Ratio	Ratio of the total number of characters of non-stop unique words in one question to the total number of characters of non-stop unique words in the other question. The ratio here is always smaller than or equal to 1, namely that it is always the shorter question that is at the numerator and the longer question that is at the denominator. The range of stop words is defined by *nltk.corpus.stopwords*.

TABLE 5. EXTRACTED FEATURES CONTINUE

Feature ID	Feature Name	Definition
24	Q1 to Q2 Word Count Difference	Difference between the total number of words in question 1 and the total number of words in question 2. The difference here can either be positive or negative (or zero).
25	Q1 to Q2 Word Count Ratio	Ratio of the total number of words in question 1 to the total number of words in question 2. The ratio here can be either greater or smaller than (or equal to) 1.
26	Q1 to Q2 Unique Word Count Difference	Difference between the total number of unique words in question 1 and the total number of unique words in question 2. The difference here can either be positive or negative (or zero).
27	Q1 to Q2 Unique Word Count Ratio	Ratio of the total number of unique words in question 1 to the total number of unique words in question 2. The ratio here can be either greater or smaller than (or equal to) 1.
28	Q1 to Q2 Unique Non-Stop Word Count Difference	Difference between the total number of unique non-stop words in question 1 and the total number of unique non-stop words in question 2. The difference here can either be positive or negative (or zero). The range of stop words is defined by *nltk.corpus.stopwords*.
29	Q1 to Q2 Unique Non-Stop Word Count Ratio	Ratio of the total number of unique non-stop words in question 1 to the total number of unique non-stop words in question 2. The ratio here can be either greater or smaller than (or equal to) 1. The range of stop words is defined by *nltk.corpus.stopwords*.
30	Q1 to Q2 Character Difference	Difference between the total number of characters in question 1 and the total number of characters in question 2. The difference here can either be positive or negative (or zero).
31	Q1 to Q2 Character Length Ratio	Ratio of the total number of characters in question 1 to the total number of characters in question 2. The ratio here can be either greater or smaller than (or equal to) 1.
32	Q1 to Q2 Unique Word Character Difference	Difference between the total number of characters of unique words in question 1 and the total number of characters of unique words in question 2. The difference here can either be positive or negative (or zero).
33	Q1 to Q2 Unique Word Character Ratio	Ratio of the total number of characters of unique words in question 1 to the total number of characters of unique words in question 2. The ratio here can be either greater or smaller than (or equal to) 1.
34	Q1 to Q2 Unique Non-Stop Word Character Difference	Difference between the total number of characters of non-stop unique words in question 1 and the total number of characters of non-stop unique words in question 2. The difference here can either be positive or negative (or zero). The range of stop words is defined by *nltk.corpus.stopwords*.
35	Q1 to Q2 Unique Non-Stop Word Character Ratio	Ratio of the total number of characters of non-stop unique words in question 1 to the total number of characters of non-stop unique words in question 2. The ratio here can be either greater or smaller than (or equal to) 1. The range of stop words is defined by *nltk.corpus.stopwords*.
36	Alternative Non-Stop Word Match Share	The ratio of the shared non-stop word of the question pair to the total number of words of the question pair. Here, the range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
37	TF-IDF Alternative Non-Stop Word Match Share	The ratio of the weight of the shared non-stop words of the question pair to the weight of the total number of non-stop words of the question pair. The weight of a word is defined as *1 / # of the word occurs in the whole data set*. Here, the range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
38	Alternative Common Unigrams Count	Number of unigrams that the two questions share. A unigram here is defined as any words except stop words defined by the alternative stops mentioned in Table 3, row 4.
39	Alternative Common Unigrams Ratio	Ratio of unigrams that the two questions share to the total unique unigrams that the question pair have. A unigram here is defined as any words except stop words defined by the alternative stops mentioned in Table 3, row 4.
40	Alternative Unique Non-Stop Word Count Difference	Difference between the total number of unique non-stop words in one question and the total number of unique non-stop words in the other question. The difference here is an absolute value, which is always non-negative. The range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
41	Alternative Unique Non-Stop Word Count Ratio	Ratio of the total number of unique non-stop words in one question to the total number of unique non-stop words in the other question. The ratio here is always smaller than or equal to 1, namely that it is always the shorter question that is at the numerator and the longer question that is at the denominator. The range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
42	Alternative Unique Non-Stop Word Character Difference	Difference between the total number of characters of non-stop unique words in one question and the total number of characters of non-stop unique words in the other question. The difference here is an absolute value, which is always non-negative. The range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
43	Alternative Unique Non-Stop Word Character Ratio	Ratio of the total number of characters of non-stop unique words in one question to the total number of characters of non-stop unique words in the other question. The ratio here is always smaller than or equal to 1, namely that it is always the shorter question that is at the numerator and the longer question that is at the denominator. The range of stop words is defined by the alternative stops mentioned in Table 3, row 4.

TABLE 6. EXTRACTED FEATURES CONTINUE

Feature ID	Feature Name	Definition
44	Alternative Q1 to Q2 Unique Non-Stop Word Count Difference	Difference between the total number of unique non-stop words in question 1 and the total number of unique non-stop words in question 2. The difference here can either be positive or negative (or zero). The range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
45	Alternative Q1 to Q2 Unique Non-Stop Word Count Ratio	Ratio of the total number of unique non-stop words in question 1 to the total number of unique non-stop words in question 2. The ratio here can be either greater or smaller than (or equal to) 1. The range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
46	Alternative Q1 to Q2 Unique Non-Stop Word Character Difference	Difference between the total number of characters of non-stop unique words in question 1 and the total number of characters of non-stop unique words in question 2. The difference here can either be positive or negative (or zero). The range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
47	Alternative Q1 to Q2 Unique Non-Stop Word Character Ratio	Ratio of the total number of characters of non-stop unique words in question 1 to the total number of characters of non-stop unique words in question 2. The ratio here can be either greater or smaller than (or equal to) 1. The range of stop words is defined by the alternative stops mentioned in Table 3, row 4.
48	IBM Keyword Share	The ratio of the shared keyword of the question pair to the total number of words of the question pair. Here, the keywords of each question are extracted using the IBM API.
49	IBM Keyword Share Count	The total number of the shared keyword of the question pair. Here, the keywords of each question are extracted using the IBM API.
50	IBM Keyword Ratio	The ratio of the shared keyword of the question pair to the total keywords of the question pair. Here, the keywords of each question are extracted using the IBM API.

TABLE 7. DETAILED INFORMATION ABOUT COURSES USED FOR INCOMPLETE OFFICE HOURS REQUESTS IDENTIFIER

Tag	Course Title	Semester	OH Requests
FP	Foundations of Programming	Su 2021	391
PL1	Programming Language Concepts	F 2020	197
PL2	Programming Language Concepts	F 2021	206
DS	Distributed Systems	Sp 2021	58

ups. To get the office hours request data, we downloaded all the follow-ups in that post and eliminate student names' and instructor responses using our self-built parser so that the remaining data is students' office hours requests.

Then we labelled each of the office hours requests using four labels: *Time*, *Reason*, *good*, and *None*. A detailed explanation of the three labels can be found in Table 8.

One major difference between the scheme mentioned in Table 2 and 8 is that each post must be categorized into one and only one type mentioned in Table 2. However, one office hours request can be categorized into multiple types. For example, one office hours request that specifies neither a meeting time nor a detailed description about the student's problem will be categorized as both *Time* and *Reason*.

For the four courses, FP1 was labeled by the first author, and the rest were labeled by the third author. After labeling, we removed all office hours requests that were labeled as *None* before we cleaned the data.

Since the *Time* and *Reason* labels are not mutually exclusive, we decided to convert the multiclass classification problem into two separate binary classification problems, which is helpful to improve the model's performance. As a result, we split the labels into two columns. The first column indicates whether the office hours request mentions a specific meeting time with the instructor, which we will refer to as "time component" afterward, and the second column indicates whether the office hours request gives a detailed description of the student's problem, which we will refer to as "reason component" afterward. We then converted our original String labels into integers 1 and 0 to represent True and False in the two columns, respectively.

4.3.2. Data Cleaning. The data cleaning procedure is the same as the one mentioned in section 4.1.2.

4.3.3. Feature Engineering. Similar to section 4.2.3, the extracted features for this task include both word vectors generated by embedding the posts and other features defined and extracted by our algorithm.

The embedding procedure is the same as the one mentioned in section 4.1.3. We also extracted two groups of features, one for time component identification and the other for reason component identification. The features are explained in Table 9 and 10.

In Table 10, TF-IDF is a numerical statistic that is intended to reflect how important a word is in a document. TF-IDF is defined as

$$TF - IDF = TF \times IDF, \tag{4}$$

where TF is defined as the number of times a word appears in the context over the total word count of the context, and IDF is defined as the logarithm of the total number of documents over the number of documents that contain the target word plus 1. In our case, we calculated the TF of each word in an office hours request by counting its appearance in the request divided by the total word count of the request. We calculated the IDF of each word in an office hours request by summing the number of total office

TABLE 8. OFFICE HOURS REQUEST CLASSIFICATION SCHEME

Type	Definition	Example
Time	The office hours request does not mention a specific meeting time with the instructor. Here a specific meeting time is a time that is accurate to the minute instead of only mentioning the meeting date or hour.	I plan to attend today Tuesday March 2 about issue @87
Reason	The office hours request does not give a detailed description of the student's problem.	I plan to attend OH tomorrow from 10-11
Good	The office hours request shows both a specific meeting time and a detailed description of the student's problem.	9/21 3:00 assignment 1, having trouble building the weka tree. I think I have the general class structure correct and it is reading in the values correctly, somehow the tree isn't being built.
None	The content in the office hours request does not indicate that the student is requesting office hours.	I figured it out! Not coming at 3 anymore

TABLE 9. TIME COMPONENT FEATURES

Feature ID	Feature Name	Definition
1	Time Feature 1	Does the office hours request contains substring "at—before—after—around X/Xam/AM—pm/PM/X am/AM—pm/PM" (such as "at 10" or "10am") or HH:MM (such as 10:00) or HH-HH (such as 10-11)? 1 if true and 0 if false.
2	Time Feature 2	Does the office hours request contains substring "after someone" or "with someone" (such as " I would like to come after/with someone...")? 1 if true and 0 if false.
3	Time Feature 3	Does the office hours request contains substring "MM/DD HH" (such as "11/17 11")? 1 if true and 0 if false.

TABLE 10. TIME COMPONENT FEATURES

Feature ID	Feature Name	Definition
1	Word Count	The number of words that the office hours request have.
2	Unique Word Count	The number of unique words that the office hours request have.
3	Average TF-IDF	Average TF-IDF of the office hours request.
4	Keyword Count	Number of keywords extracted by the IBM API.
5	Post Link	Does the request contain a link to a Piazza post using "@"? 1 if true and 0 if false.
6	Keyword: Think	Does the request contains the keyword "think"? 1 if true and 0 if false.
7	Keyword: Use	Does the request contains the keyword "use"? 1 if true and 0 if false.
8	Keyword: Some	Does the request contains the keyword "some"? 1 if true and 0 if false.
9	Keyword: Question	Does the request contains the keyword "question"? 1 if true and 0 if false.
10	Keyword: Error	Does the request contains the keyword "error"? 1 if true and 0 if false.

hours requests in the 4 CSV files and divided by the total number of office hours requests that contain the target word plus one, then applying logarithm. Then we used eq. 4 to get TF-IDF for each word. For each office hours request, we average the TF-IDF value for each word in the request to indicate the uniqueness of the request.

For feature 4 in Table 10, we used the IBM API to extracted keywords from each office hours request. The procedure is the same as in section 4.2.3. For features 5, we assumed that an office hours request is considered as complete in reason when it refers to an independent post. For features 6 - 10, we analyzed the words that have significantly different appearance frequency in office hours requests that lack reason component and their counterparts. We found that the keywords mentioned in features 6 - 10 are at least six times more likely to appear in one category of office hours requests than the other, which should be an important indicator in reason component classification.

Since we have much fewer features compared to those mentioned in section 4.2.3, we did not do PCA here. Instead, we input all the extracted features into the model.

4.3.4. Prediction Model Set Up. Since the number of office hours requests for each class group was much less than the number of posts for each class group, we decided not to split the data into different course groups. Instead, we combined all office hours requests into one data set and used *train_test_split* from *sklearn* to split the data into two segments with 80% as the training data and the rest 20% as the test data using $random\_state = 4242$.

We used the same four prediction models mentioned in section 4.1.4. Since we converted the identification problem into two independent binary classification problems, the model settings are identical to the binary classification for

duplicate posts as mentioned in section 4.2.4. Each model was asked to make two independent predictions for the time and reason component, respectively.

After the model output its prediction, we asked it to generate an automatic response according to the result, which would be used to notify the student. If an office hours request is both time and reason complete, there will be no response and the student will not be notified. The automatic responses for the office hours requests that are identified as lacking time component are identical, which lets the students who post the request specify the meeting time with the instructor. The automatic responses for the office hours requests that are identified as lacking reason component are tailored according to students' requests, which not only lets them specify the problem they want to address with the TA but also informs them about the section they should specify. To achieve this, we utilized the IBM Watson Natural Language Understanding API to extract the keywords in students' requests. Then we refined the return keyword sets to eliminate uninformative keywords such as the instructors' names and common words that are unrelated to computer science. For the requests that are too general to extract any informative keywords, we used a general response which lets the students be more specific about their problems.

4.4. Automatic Assignment Assistant

4.4.1. Data Collection. The key of this task is to see if the model can identify possible functions/methods that the student needs help on, and provide the student with the English translation of the functions/methods he asks for. Based on the posts data we used in section 5.1 and the office hours requests data we used in section 4.3, we found many posts or requests that ask the implementation of specific functions/methods about compiled language such as Java and C, interpreted languages such as Prolog and SML, and assembly language such as MIPS. However, we did not have access to the instructors' solutions to these functions/methods. As a result, we used alternative solutions from students who have taken the courses and have received full credits for the coding assignments of the courses.

Moreover, we added Python as another programming language even if it is not involved in courses mentioned in Table 4.1.1 and 4.2.1 as it is widely used in other subjects such as Physics and Data Science. Since there is no assignments or posts about Python implementation, we used the Python code we wrote for cleaning the data, embedding the posts and office hours requests, and training the models as a mock assignment and designed a set of post questions that simulates typical ways for students to ask implementations about these methods.

For visualization purposes, we selected two functions/methods of each programming language listed above that are short and relatively easy to understand without too much knowledge about the corresponding programming language, the functionality and the actual code can be found in Table 11. Note that the actual code only includes the implementation of the functions/methods themselves, all other functions/methods or variables/fields that are defined elsewhere and are called or accessed by the implementation are not included.

Code Listing 1. Python tfidf_word_match_share

```
def tfidf_word_match_share(row):
    q1words = {}
    q2words = {}
    for word in str(row[3]).lower().split():
        q1words[word] = 1
    for word in str(row[4]).lower().split():
        q2words[word] = 1
    if len(q1words) == 0 or len(q2words) == 0:
        return 0

    shared_weights = [weights.get(w, 0) for w in
        q1words.keys() if w in q2words] + [weights
        .get(w, 0) for w in q2words.keys() if w in
         q1words]

    total_weights = [weights.get(w, 0) for w in
        q1words] + [weights.get(w, 0) for w in
        q2words]

    R = np.sum(shared_weights) / np.sum(
        total_weights)
    return R
```

Code Listing 2. Python jaccard

```
def jaccard(row):
    wic = set(str(row[3]).lower().split()).
        intersection(set(str(row[4]).lower().split
        ()))
    uw = set(str(row[3]).lower().split()).union(
        set(str(row[4]).lower().split()))

    if len(uw) == 0:
        return np.nan
    else:
        return (len(wic) / len(uw))
```

Code Listing 3. Java isGivenSafe

```
public static boolean isGivenSafe(final int
    distance, final int duration, final int
    exhalationLevel) {
                final boolean
                    intermediateResultOne =
                    safeCaseOne(distance, duration
                    , exhalationLevel) ||
                    safeCaseTwo(distance, duration
                    , exhalationLevel) ||
                    safeCaseThree(distance,
                    duration, exhalationLevel);
                final boolean
                    intermediateResultTwo =
                    safeCaseFour(distance,
                    duration, exhalationLevel) ||
                    safeCaseFive(distance,
                    duration, exhalationLevel) ||
                    safeCaseSix(distance, duration
                    , exhalationLevel);
                return intermediateResultOne ||
                    intermediateResultTwo ||
                    safeCaseSeven(distance,
                    duration, exhalationLevel);
        }
```

TABLE 11. AUTOMATIC ASSIGNMENT ASSISTANT TEST CODE

ID	Programming Language	Function / Method Name	Functionality	Code
1	Python	tfidf_word_match_share	Input a parameter *row*, which contains one question pair mentioned in section 4.2.1. Extract feature 2 mentioned in Table 4 from the question pair.	See Code Listing 1
2	Python	jaccard	Input a parameter *row*, which contains one question pair mentioned in section 4.2.1. Extract feature 6 mentioned in Table 4 from the question pair.	See Code Listing 2
3	Java	isGivenSafe	Input three integers: *distance*, *duration*, and *exhalation*, and return true if the parameter combination is in one row of Table 13. Otherwise, return false.	See Code Listing 3
4	Java	interpolatedDistance	Input an integer *distance*, and interpolate the parameter based on the second row of Table 12. If the input value is in between two values in that row, return the smaller number of the two values. If the input value is greater than 27, return 27. If the input value is smaller than 6, return 0.	See Code Listing 4
5	C	Factorial	Input an integer *n*, return $n!$.	See Code Listing 5
6	C	Fibonacci	Input an integer *n*, return the *n + 1* term of the Fibonacci sequence.	See Code Listing 6
7	Prolog	isGivenSafe	Input three parameters: *Distance*, *Duration*, and *Exhalation*, and return true if the parameter combination is in one row of Table 13. Otherwise, return false.	See Code Listing 7
8	Prolog	interpolatedDistance	Input two parameters *Distance* and *InterpolatedDistance*, and interpolate the parameter based on the second row of Table 12. If the input value is in between two values in that row, make *InterpolatedDistance* the smaller number of the two values. If the input value is greater than 27, make *InterpolatedDistance* 27. If the input value is smaller than 6, make *InterpolatedDistance* 0.	See Code Listing 8
9	SML	isGivenSafe	Input three integers: *distance*, *duration*, and *exhalation*, and return true if the parameter combination is in one row of Table 13. Otherwise, return false.	See Code Listing 9
10	SML	interpolatedDistance	Input an integer *distance*, and interpolate the parameter based on the second row of Table 12. If the input value is in between two values in that row, return the smaller number of the two values. If the input value is greater than 27, return 27. If the input value is smaller than 6, return 0.	See Code Listing 10
11	MIPS	Factorial	Input an integer *n*, return $n!$	See Code Listing 11
12	MIPS	Fibonacci	Input an integer *n*, return the *n + 1* term of the Fibonacci sequence.	See Code Listing 12

TABLE 12. SMALL, MEDIUM AND LARGE VALUES

	Small	Medium	Large
Distance	6	13	27
Duration	15	30	120
Exhalation	10	30	50

Code Listing 4. Java interpolatedDistance

```
public static int interpolatedDistance(final int
    distance) {
            if (distance < SMALL_DISTANCE) {
                return MIN_DISTANCE;
            }
            if (distance < MEDIUM_DISTANCE) {
                return SMALL_DISTANCE;
            }
            if (distance < LARGE_DISTANCE) {
                return MEDIUM_DISTANCE;
            }
            return LARGE_DISTANCE;
    }
```

Code Listing 5. C Factorial

```
long Factorial(int n)
```

TABLE 13. ASSUMED GIVEN SAFE COMBINATIONS

Distance	Duration	Exhalation
13	30	30
6	30	10
27	30	50
13	15	50
13	120	10
27	120	30
6	15	30

```
{
  if (n == 0)
    return 1;
  else
    return(n * Factorial(n-1));
}
```

Code Listing 6. C Fibonacci

```
int Fibonacci(int n)
{
    if (n <= 1)
        return n;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

Code Listing 7. Prolog isGivenSafe

```
isGivenSafe(Distance, Duration, Exhalation):-
        givenSafeFact(Distance, Duration,
            Exhalation).
```

Code Listing 8. Prolog interpolatedDistance

```
interpolatedDistance(Distance,
    InterpolatedDistance):-
        largeDistance(DistanceLarge),
        Distance >= DistanceLarge,
        largeDistance(InterpolatedDistance);

        largeDistance(DistanceLarge),
        mediumDistance(DistanceMedium),
        Distance < DistanceLarge,
        Distance >= DistanceMedium,
        mediumDistance(InterpolatedDistance);

        mediumDistance(DistanceMedium),
        smallDistance(DistanceSmall),
        Distance < DistanceMedium,
        Distance >= DistanceSmall,
        smallDistance(InterpolatedDistance);

        minDistance(InterpolatedDistance).
```

Code Listing 9. SML isGivenSafe

```
fun isGivenSafe(13, 30, 30) = true |
        isGivenSafe(6, 30, 10) = true |
        isGivenSafe(27, 30, 50) = true |
        isGivenSafe(13, 15, 50) = true |
        isGivenSafe(13, 120, 10) = true |
        isGivenSafe(27, 120, 30) = true |
        isGivenSafe(6, 15, 30) = true |
        isGivenSafe(distance, duration, exhalation
            ) = false;
```

Code Listing 10. SML interpolatedDistance

```
fun interpolatedDistance(distance) =
                if distance < smallDistance then
                                minDistance
                else if distance < meidumDistance
                    then
                                smallDistance
                else if distance < largeDistance
                    then
                                meidumDistance
                else
                                largeDistance;
```

Code Listing 11. MIPS Factorial

```
Factorial:
        addi $sp, $sp, -8
        sw $ra, 4($sp)
        sw $fp, 0($sp)
        addi $fp, $sp, 4

        addi $sp, $sp, -4
        sw $s0, 0($sp)

        slt $t0, $0, $a0
        bne $t0, $0, If

        addi $v0, $0, 1
        j End


If:
        addi $s0, $a0, 0
        addi $a0, $a0, -1
        jal Factorial
        mult $s0, $v0
        mflo $v0
        j End

End:
        lw $s0, -8($fp)
        addi $sp, $fp, 4
        lw $ra, 0($fp)
        lw $fp, -4($fp)
        jr $ra
```

Code Listing 12. MIPS Fibonacci

```
Fibonacci:
        addi $sp, $sp, -8
        sw $ra, 4($sp)
        sw $fp, 0($sp)
        addi $fp, $sp, 4

        addi $sp, $sp, -4
        sw $s0, 0($sp)

        slti $t0, $a0, 2
        beq $t0, $0, Recursion
        beq $a0, $0, Zero
        addi $v0, $0, 1
        j End

Zero:
        addi $v0, $0, 0
        j End


Recursion:
        addi $s0, $a0, 0
        addi $a0, $a0, -1
```

```
        jal Fibonacci

        addi $a0, $s0, -2
        addi $s0, $v0, 0
        jal Fibonacci

        add $v0, $v0, $s0

End:
        lw $s0, -8($fp)
        addi $sp, $fp, 4
        lw $ra, 0($fp)
        lw $fp, -4($fp)
        jr $ra
```

Except for the python code snippets, all other code snippets displayed are from students' correct solution of specific assignment of a course mentioned in section 1 and 7. However, since most code snippets shown above are relatively easy implementations, there are few questions about them. To test if the model can return a satisfying English translation of the code when a specific implementation is asked, we designed some sample questions that directly ask the implementation of the above functions/methods.

4.4.2. Data Cleaning. The designed sample questions went through the data cleaning process same as what is mentioned in section 4.1.2. We removed the punctuations and stopwords to retain only key components of the questions.

4.4.3. Feature Extraction. The cleaned questions were input into the IBM API, and the keywords for each question were extracted using the same way as in section 4.2.3. We did not extract features other than the keywords due to the reason mentioned in the next subsection. Different from the previous feature engineering sections, the keywords extracted are phrases instead of numbers and are used to identify the specific part of the implementation that the student is asking instead of feeding into the prediction models as the learning material of the semantic meaning of the post.

4.4.4. Prediction Model Set Up. Since the main task of the model here is to translate the implementation into English, which is not a classification problem, we did not use the models introduced in section 4.1.4. Instead, we used the OpenAI API summary feature, which is one functionality of the OpenAI API completion model. For any input text, the model will generate a text summarization of it. Although there are no available examples about summarizing code using this model, the training data of the model contains code. To achieve better performance, we used the *Davinci* engine, which is the most capable engine of the model, and followed the recommended model setup instructions for summarization and set the *temperature* parameter, which controls the randomness of the output, to be zero, and kept all other parameters at their default values.

We connected to the OpenAI server using the *openai.Completion.create* function with our private API key and the code snippet we wanted to translate. After the request process finishes, a JSON file that contains the summary will be returned. We then extracted the summary and created a code summarization map. To distinguish the same function/method that is implemented using different programming languages, we made the key of the summarization map to be a two-element string array, where the first element is the programming language identifier, and the second element is the function/method name identifier. The value of the map is the summarization of each function/method. Our model determines the programming language and the implementation that the student asks for by matching the keywords extracted by the IBM API and the keys in the code summarization map. Since both the programming language identifier and the function/method name identifier should be a single word in most cases while each keyword extracted by the IBM API may be a phrase that contains multiple words, we use the *split* function to split each phrase into separate words by whitespaces. We stored the whole set of available programming languages and available function/method names into two arrays. For each word in the keyword set, the model will firstly compare it to the programming language array and then the function/method name array. To allow fuzzy matching, we used the *BERTSimilarity* package, which helps calculate the similarity between two strings. The return value is a float ranges from 0 to 1, where 0 indicates that the two strings are completely different and 1 indicates that the two strings are identical. We defined a match to be the case when the similarity between the extracted keyword and the key in the code summarization map is greater than 0.9. The model will then create two temporary sets that store the matched non-repeated programming language and function/method name elements. After the matching phase is over, the model will construct the key by selecting one element from both temporary sets and respond to the student about the summarization of all functions/methods that associate with the possible keys that it can construct. When a student mentions a function/method name without specifying the programming language, the model will autofill the programming language part of the key by trying all the available programming languages stored in the array and return all the matches it gets. If a student does not specify a function/method name but specifies the programming language in his question, the model will send a notification to the student and ask him to specify the function/method name.

5. Results and Discussion

After experimenting with different data cleaning strategy, we found that the model usually achieved its highest prediction accuracy when using the *hyper_cleaning* = *True*, *remove_punctuation* = *True*, *remove_stopwords* = *True*, *use_alternative_stopwords* = *True*, and *stem_words* = *False* combination, so we decided to use the data cleaned by this combination as the standard cleaned data for all following subsections.

The reason why we prefer the reduced stopwords set is that the standard stopwords set defined by *nltk.corpus* is too large that it may over clean the data and confuse some

inchoate language models. One example is shown in Table 14, where the first column is the original uncleaned post and the language model names or human that label the post. The second column is the post cleaned using the *hyper_cleaning = True*, *remove_punctuation = True*, *remove_stopwords = True*, *use_alternative_stopwords = False*, and *stem_words = False* combination and the labeling result of each language model together with the label provided by human. The third column is the post cleaned using the *hyper_cleaning = True*, *remove_punctuation = True*, *remove_stopwords = True*, *use_alternative_stopwords = True*, and *stem_words = False* combination and the labeling result of each language model together with the label provided by human.

According to Table 14, the first two language models output incorrect predictions about the post. There are two possible reasons for this. First, the post is relatively short compared to other constructive posts. With a further elimination on stopwords using the standard stopwords set, the resulting post will be so short that the inchoate language models, which do not have a numerical vector large enough to understand the semantic meaning of the post very precisely, will label it as an active post since its length falls into the category of a typical active question. Second, the phrase "I have" was eliminated when using the standard stopwords set but was kept when using the reduced stopwords set. Since this is a critical phrase where the student shew what he had attempted to solve the problem, removing this phrase will make the subject of the sentence vague, which hinders the language models' decisions about whether the student has shown attempt or reasoning in his post. Since both reasons originated from the over clean of the context using the standard stopwords set and can be avoided using the reduced stopwords set, we decided to use the reduced stopwords set.

5.1. Automatic Labeling System

To visualize the distribution of data, we plot the ratio of the four different question categories mentioned in Table 2 for COMP 401, COMP 411, COMP 426, and all courses. The result is shown in Fig. 1.

We found through Fig.1 that for each course group, the distribution is not very even, especially for COMP 401 and COMP 426, where the number of "Constructive" and "Content Clarification" is much less than other types of posts. Fortunately, the overall distribution indicates that the data can still be considered unbiased if we train our models on the overall data.

To visualize the performance of each language model and prediction model, we set both the training and test data set to use all the posts and calculate the accuracy of each prediction model based on its prediction on the embedded data provided by each language model. We were most interested in this prediction result since the model gets the most data to learn in this scenario and becomes the most comprehensive. In practice, the model is required to predict posts from courses other than the courses mentioned in Table 1 and we planned to use all embedded posts as its training data. Therefore, even if the posts in the test set still belongs to one of the three course groups Table 1, the training data is similar to which we decided to use to develop the final product, so this result can reflect how well the model can be when we finalize it. Also, according to Fig. 1, the data is most unbiased if we use the overall data as the training data, which is helpful when training the model. Since the performance of each prediction model differs a little in multiple experiments with the same training and test data, we experimented with each language model - prediction model pair ten times and average the accuracy to be the accuracy of the language model - prediction model pair. The result is shown in Fig. 2.

according to Fig. 2, there is a clear improvement in accuracy from Word2Vec, the oldest model, to GPT-3, one of the latest models. Compared to the difference of different language models, the performance difference between different prediction models is less obvious. The CNN model gets the highest accuracy for most language models. From our perspective, this is due to the complexity of the model. Since we used six convolutional layers with different filter and kernel sizes, it has a stronger capability to identify features that are neglected by other models. On the other hand, XGBoost's performance is relatively worse compared to other models. This is because the parameters of XGBoost models usually need to be fine-tuned in advance to ensure better performance, while we kept most of the parameters in their default settings.

To see how each language model performs in other prediction tasks such as using one group of COMP courses to predict another group of COMP courses, we plot the heat map of the prediction of each language model paired with the CNN prediction model using *seaborn.heatmap*, where the x-axis stands for the group of courses we used as test data and the y-axis stands for the group of courses we used as training data. This result is shown in Fig. 3. We do not show the prediction results for other prediction models as it may course redundancy and the performance difference of prediction models is much more subtle than that of the language model.

Due to the number of tests required to generate Fig. 3 and the time needed to train the model, we only conduct each experiment once, except for the cell at the bottom right of each heat map, where we experimented 10 times to make Fig. 2.

Similar to the finding in Fig. 2, there is a pattern of performance improvement from Word2Vec to GPT-3. We also found from Fig. 3 that the cells in the leading diagonal have higher accuracy than other cells. This is reasonable since the same courses taught in different semesters are usually closer in style than other courses. As a result, students tend to ask similar questions. Also, since the training data and the test data are more similar in style, all models result in better performance. On the contrary, since some course groups have biased data and different course groups have very different post category distribution, training on one course group and predicting on another course group is not ideal. However, we also found that the difference between the diagonal cells and other cells gets smaller from Word2Vec to

TABLE 14. OVER CLEANED POST EXAMPLE

Original Post	**Post cleaned using standard stopwords set**	**Post cleaned using reduced stopwords set**
Neither of these work on my assignments 1/2. I have downloaded the Comp401 jar for gradingTools and followed the instructions until where it tells me to import gradingTools.	neither work assignments 1 2 downloaded comp401 jar gradingtools followed instructions tells import gradingtools	neither work on my assignments 1 2 i have downloaded comp401 jar gradingtools followed instructions until where it tells me import gradingtools
Word2Vec	Active	Constructive
GloVe	Active	Constructive
BERT	Constructive	Constructive
RoBERTa	Constructive	Constructive
DeBERTa	Constructive	Constructive
GPT-3	Constructive	Constructive
Human	Constructive	Constructive

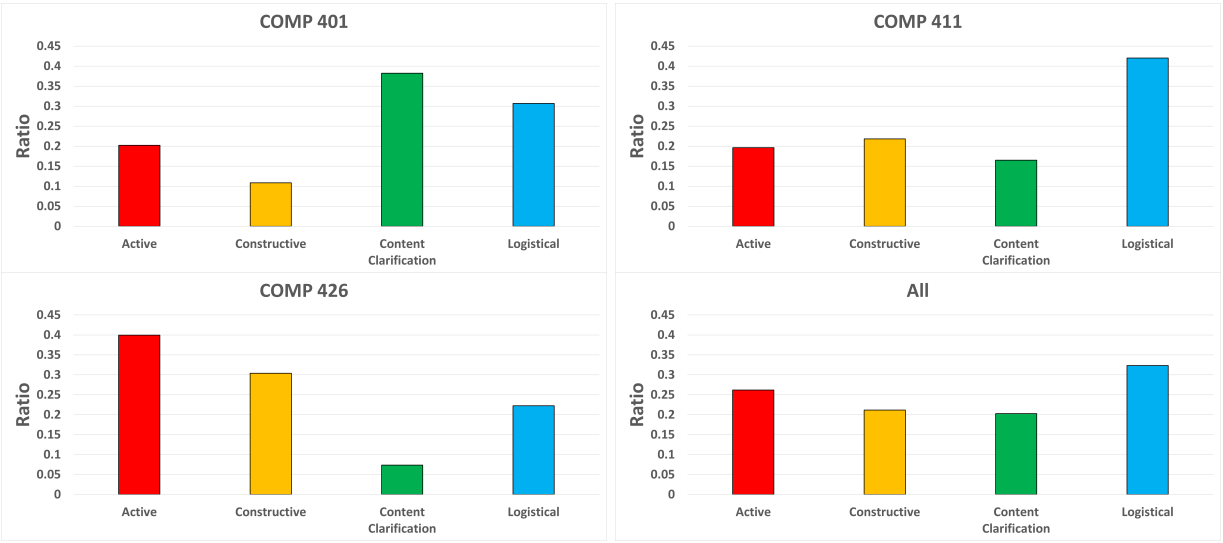

Figure 1. Distribution of four post categories of all COMP 401 courses, all COMP 411 courses, all COMP 426 courses, and all courses.

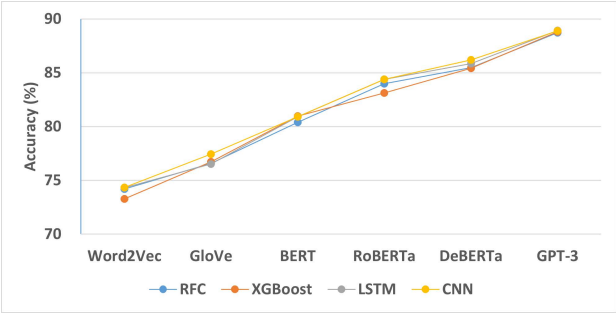

Figure 2. Average accuracy of ten independent experiments of all language models with different prediction models on classifying Piazza posts into categories described in Table 2. The training data of the prediction model is 80% of the embedded posts of all 13 courses in Table 1. The test data of the prediction model is the rest 20% of the embedded posts of all 13 courses in Table 1.

GPT-3. This is because as the language model used becomes more capable and the dimension of the embedded vector gets larger, the model can understand the semantic meaning of the posts better. As a result, the style and category ratio difference between different courses can make less impact on the models since the definition of each label, which all models try to learn, stays the same for all courses.

To visualize and compare the performance of each model on a particular task, we plotted the confusion matrix of each model when using 80% of all post data to predict the rest 20% of all post data, which is the same task mentioned in Fig. 2. Since the fluctuation of each model's accuracy is mild for multiple predictions on the same task and the values in the confusion matrix will become floats instead of integers, which makes it harder to interpret if we conduct multiple experiments and average the numbers in the cells, we decided to provide the confusion plot for one particular experiment for each model. The results are shown in Fig. 4.

according to Fig. 4, the biggest problem that all models

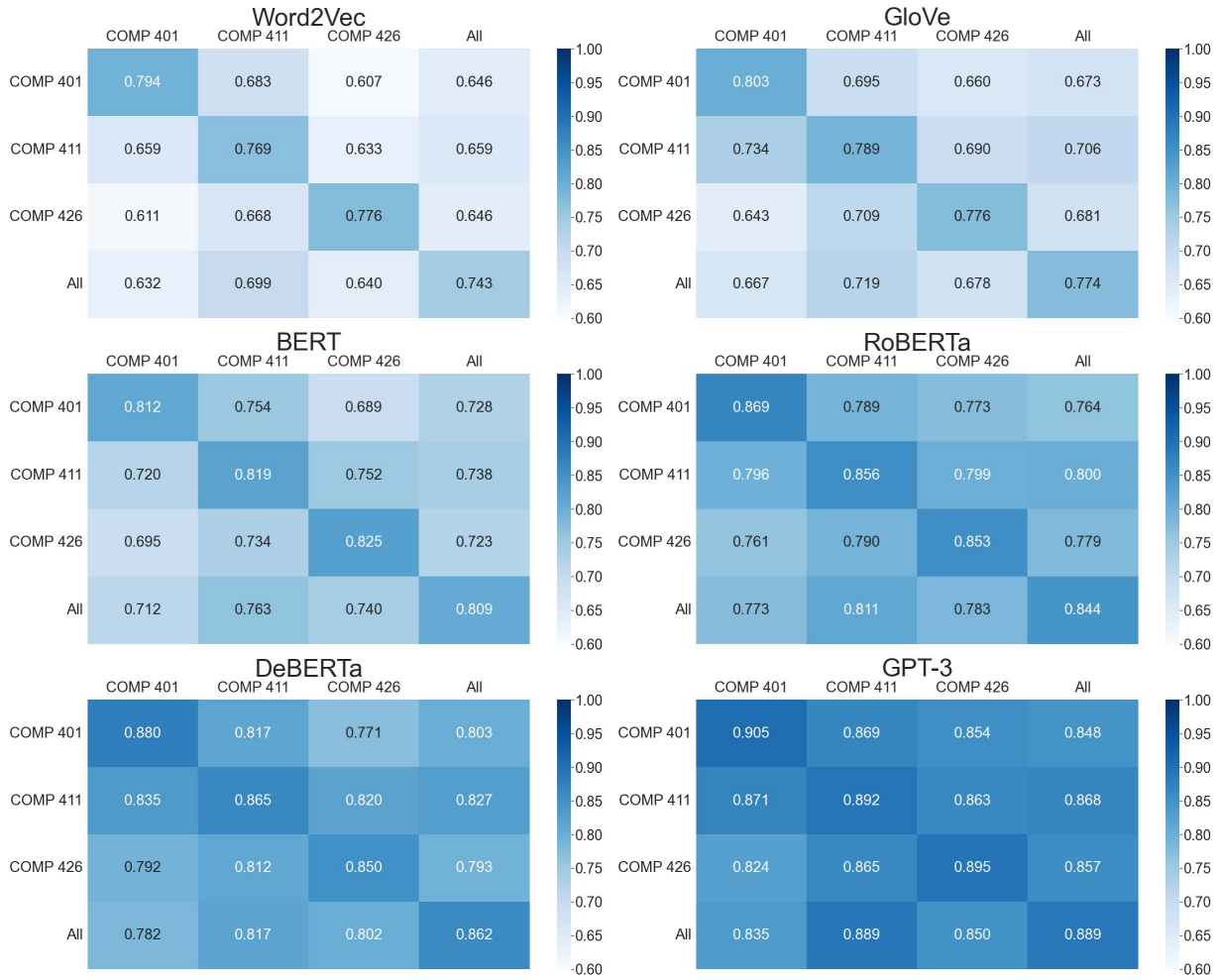

Figure 3. Prediction accuracy of the Word2Vec, GloVe, BERT, RoBERTa, DeBERTa, and GPT-3 language model on classifying Piazza posts into categories described in Table 2. The prediction model is CNN. The x-axis stands for the group of courses used as test data and the y-axis stands for the group of courses used as training data.

have is to distinguish between "Active" and "Constructive" problems. According to the definition in Table 2, the major difference between "Active" and "Constructive" problems is if the question contains what the student knows or has already attempted. Even if the definition is clear for differentiating these two types of questions, they are confusing sometimes even when we are labeling the posts. To analyze what kind of posts will make the models return incorrect predictions or disagree with each other, we selected some typical examples of misleading posts. The result is shown in table 15, where the first row is the body of the highly confusing post, the second to eighth rows are the label predicted by CNN using the embedding data provided using the six language models together with the one labeled by human, and the last row is the analysis of the potential reason behind the disagreement. Due to the cleaning strategy we used, the posts input into, learned by, and returned from the model did not contain punctuation and stop words, which makes them hard to read. As a result, we found the corresponding original uncleaned post in our data set and manually removed bad characters but kept the punctuation and stop words for visualization purposes.

Another relatively highly confusing area is the last column, where many non-logistical posts are categorized as logistical posts. Since the difference between logistical and non-logistical posts should be greater than that between active and constructive posts according to the definition, we thought that the major factor is the ratio of logistical posts. according to Fig. 1, even if the overall post category distribution is relatively unbiased, the ratio of logistical posts is still a little higher than other post categories. This makes the prediction more likely to label one post as "Logistical" instead of other categories. This is a problem for many prediction models as bias always affects their decision. However, this adverse effect gets milder as the model we used gets more capable since the model understands the meaning of the posts better.

Overall, the result shows decent accuracy achieved by

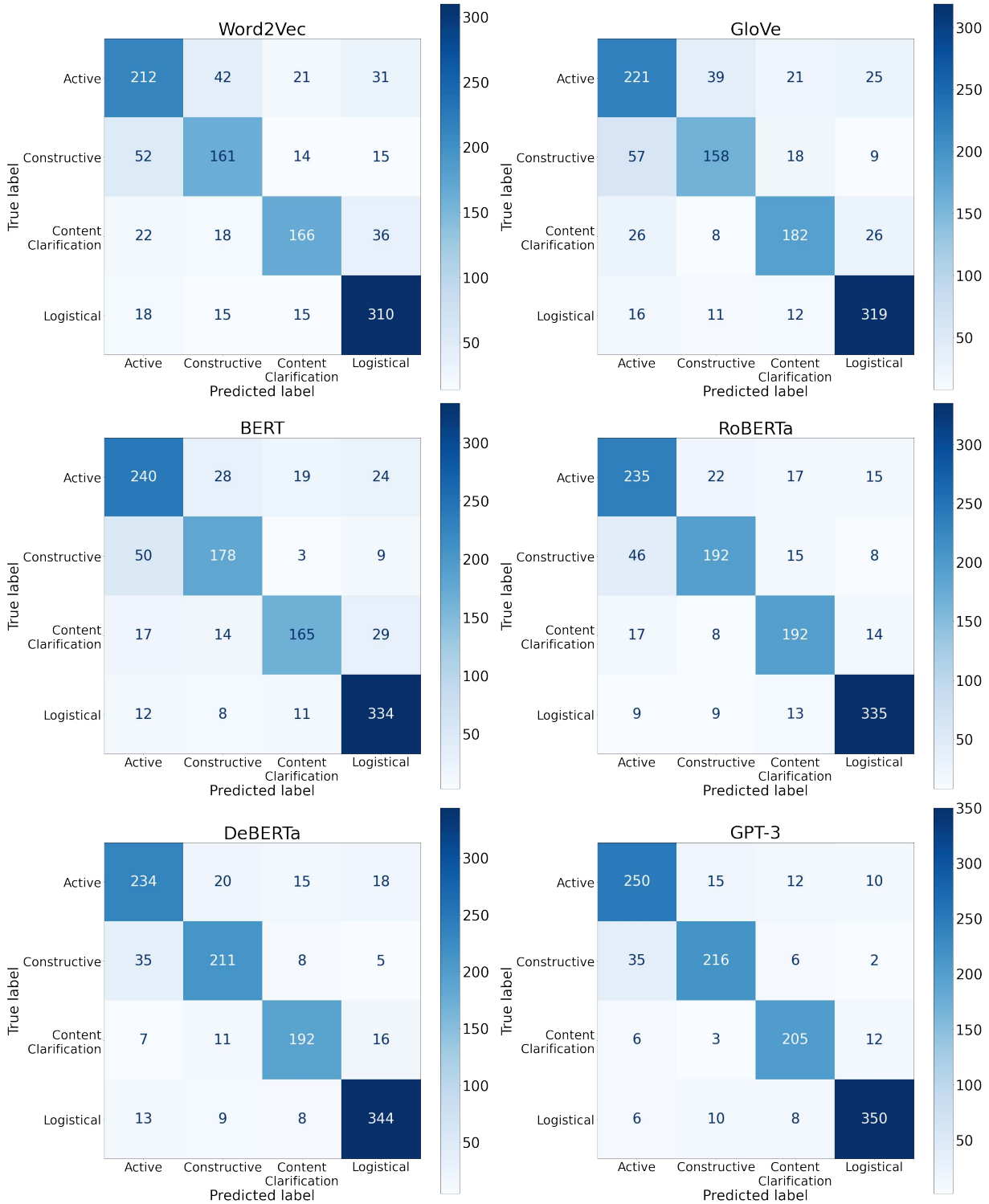

Figure 4. Confusion matrices of the Word2Vec, GloVe, BERT, RoBERTa, DeBERTa, and GPT-3 language model on classifying Piazza posts into categories described in Table 2. The prediction model is CNN. The training data of the prediction model is 80% of the embedded posts from all 13 courses in Table 1. The test data of the prediction model is the rest 20% of the embedded posts from all 13 courses in Table 1. The x-axis stands for the label predicted by the prediction model and the y-axis stands for the true label. The result is based on one experiment for each language model.

TABLE 15. HIGHLY CONFUSING POSTS ANALYSIS

	Post 1	**Post 2**	**Post 3**
Post Body	So I have my avatars disappearing and appearing when I run dorothyEnters dorothyLeaves etc methods, but I'm having some trouble with layering. For instance, when I make lionEnter, dorothy (or sometimes just pieces of her) goes underneath the background! I've been fiddling around with the @Position(1) above each getter but something always seems to mess up, different things each time. Any ideas/suggestions? I actually noticed I was having some trouble with this in OE (but not the other view) in the last assignment, but I didn't realize until now that the disappearing objects were going underneath the background. #assignment10	I believe that the way to solve this question is floor((addr/linesize)%lines/2) so why is floor((A/2**L)%2**(C-L)/2) incorrect?	So my understanding is that the JWT should be stored in local storage and we should avoid storing the user data by itself in storage. I want my project on the private end to display some information along with the username for the user who posts the information. How do I obtain the user information using the jwt?
Word2Vec	Constructive	Active	Constructive
GloVe	Constructive	Active	Constructive
BERT	Constructive	Active	Constructive
RoBERTa	Constructive	Active	Constructive
DeBERTa	Constructive	Active	Active
GPT-3	Active	Constructive	Active
Human	Active	Constructive	Active
Analysis	This post is longer than a typical active post, which makes it more like a constructive post. The student has a clear description of the problem, but he does not specify what may go wrong. Instead, he used vague phrases such as "something" and "having some trouble". As a result, his description does not show his understanding of the concept, nor does he clearly shows an attempt that tries to solve the problem. Only the GPT-3 model correctly labels this post, which mainly due to its large numerical vector size.	In this post, the student showed his understanding of the problem and proposed his attempt to solve the problem. However, since this post is relatively short compared to a typical constructive post and most of the reasoning part consists of expressions and symbols, the models are having a hard time making the right decision. Only the GPT-3 model correctly labels this post, which mainly due to its large numerical vector size.	This is an arguable post. The student states his understanding of JWT in the beginning. However, according to the question he asked in the end, this is an active question since he directly asks for help without showing any attempt, and his understanding of JWT, in the beginning, does not contribute to his understanding of obtaining user information from JWT.

our language and prediction models, especially the GPT-3 language model, which achieved at least 82.4% accuracy in all prediction tasks, and a 88.9% accuracy when using the overall data as the training data. This shows the power of the language models and the potential to develop the models into a fully-functional automatic labeling system as part of the intelligent teaching assistant.

5.2. Duplicate Posts Identifier

For this task, the data cleaning procedure is divided into two steps. We first only set *hyper_cleaning* = *True* and kept the information of punctuation and stop words. Then we extracted the features mentioned in Table 4. After that, we further cleaned the data by setting *remove_punctuation* = *True*, *remove_stopwords* = *True*, *use_alternative_stopwords* = *True*, and *stem_words* = *False*, as mentioned in the beginning of section 5. The reason that we chose to extract features using the partially cleaned data and embedding using the further cleaned data is that although the language model can have a more concise idea of the semantic meaning and the model reaches its highest accuracy when further cleaned, it lost some information about the content, and some of it may help improve the accuracy. By extracting features before the contents are further cleaned, we can retain the key information that will be lost in the further cleaning step, which should help improve the model performance.

The distribution of the training and test data set is shown in Fig. 5. according to the figure, the distribution of the twp sets are almost identical and the number of non-duplicated question pairs is about twice as many as the number of duplicated question pairs. Although this is not as unbiased as the distribution shown in Fig. 1, it is much better than the ratio of the duplicated posts in Piazza, which is less than 1% of the total posts.

For the PCA mentioned in section 4.2.3, we required that the feature selected should cover at least 95% of the total feature, namely the variance ratio should surpass 0.95. Once the requirement was met, we ranked the features according to their importance and selected the top few features suggested by the result of PCA and used them for model training. The specific features we used can be found

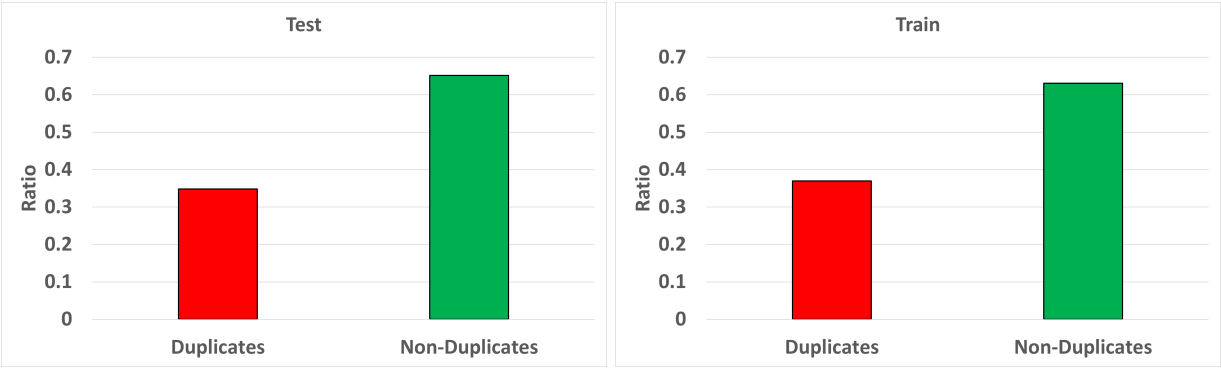

Figure 5. Distribution of duplicated and non-duplicated question pairs in the training and test data set.

in Table 16, where "1" in the "Used" column indicates that this feature was used in model training and vice versa.

According to Table 16, we found that features whose names start with "Q1 to Q2" are more often picked than their counterparts. This is reasonable since the "Q1 to Q2" features recorded the relative difference and ratio rather than the absolute difference and ratio, which keeps more information than their counterparts. Also, the result of PCA favored the features which have the "alternative" keyword in their names. From our perspective, this is due to the size difference of the two stop words set used. The stop words set defined by *nltk.corpus.stopwords* is much larger than the alternative stop words set. As a result, more words will be removed from the original question pairs, which may make the feature lose some critical features which will be retained when using the alternative stop words set. This result can also be proved by the fact that the models' reach their best performance when we used alternative stop words set instead of the stop words set defined by *nltk.corpus.stopwords* to clean the context. All the IBM keyword features are selected by the PCA, which shows the importance and usefulness of the keywords extracted by the IBM API.

Similar to section 5.1, we experimented with each language model - prediction model pair ten times and average the accuracy to be the accuracy of the language model - prediction model pair. The test result of the models on the Quora test data set is shown in Fig. 6. Again, a clear performance improvement pattern appears when more capable language models are used. Also, there is an obvious accuracy improvement for each language model compared to the result in Fig. 2. This is reasonable since the models performed multiclass classification in section 5, while they performed binary classification in this task, which reduced the possibility that the models give an incorrect prediction. Taking a close look at the figure, we found that there is a performance difference between the prediction models as the two neural network models always perform slightly better than the other two models. Our explanation for this is that since the neural network models have a more complicated structure, their advantage becomes obvious once the training data is large. However, this performance difference gradually

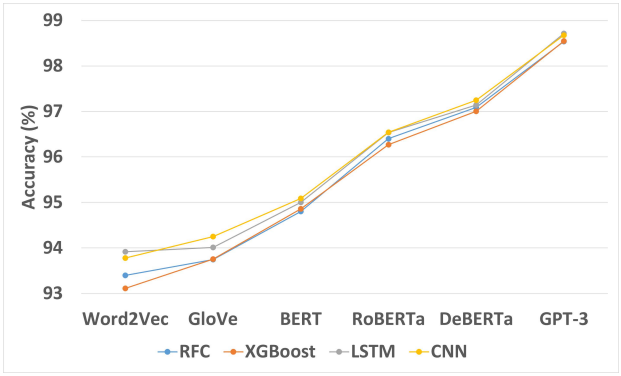

Figure 6. Average accuracy of ten independent experiments of all language models with different prediction models on classifying question pairs that are duplicates. The training data of the prediction model is the embedded 400,000 Quora question pairs with features. The test data of the prediction model is the embedded 4,290 Quora question pairs with features.

shrunk as we moved to more capable language models. This is because more capable language models provide more precise embedding results of the context, which makes the complexity of the prediction model less important. Also, as the accuracy becomes closer and closer to 100%, it is harder and harder to improve the accuracy, which makes the advantage of neural network models more and more difficult to be noticed.

To visualize how each model does when predicting duplicates, we made confusion matrices for each language model. Again, CNN was used as the prediction model for each confusion matrix. The result is shown in Fig. 7. Still, the matrix shows the prediction result of one experiment for each language model.

According to Fig. 7, we found that there are always more duplicate question pairs that are classified as non-duplicates than the opposite direction, This is mainly due to the distribution of the training data. according to Fig. 5, since there are about twice as many non-duplicate question pairs as the duplicate question pairs, the models are more likely to give incorrect predictions when the question pair is a duplicate.

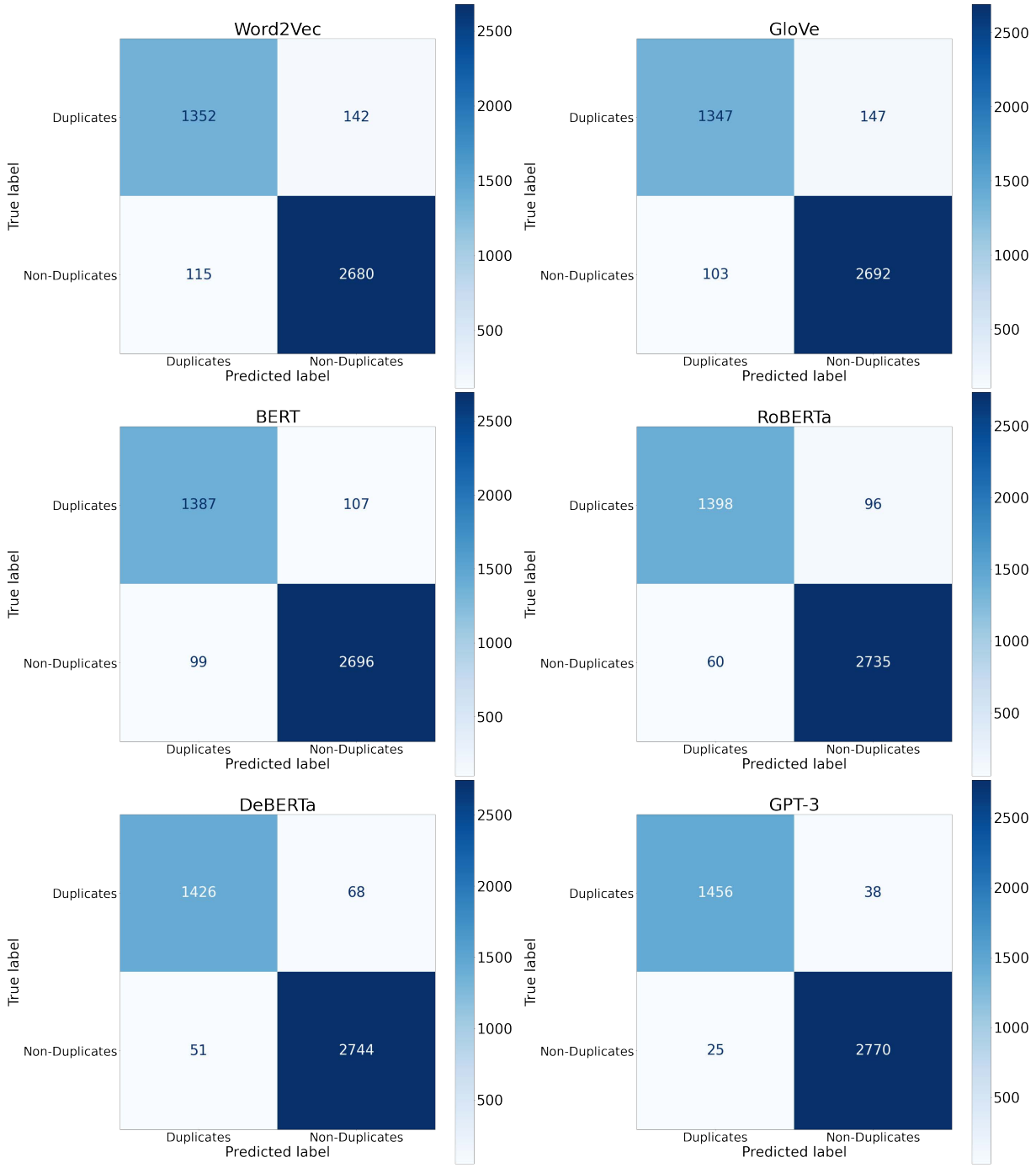

Figure 7. Confusion matrices of the Word2Vec, GloVe, BERT, RoBERTa, DeBERTa, and GPT-3 language model on classifying question pairs that are duplicates. The prediction model is CNN. The training data of the prediction model is 400,000 Quora question pairs. The test data of the prediction model is 4,290 Quora question pairs. The x-axis stands for the label predicted by the prediction model and the y-axis stands for the true label. The result is based on one experiment for each language model.

TABLE 16. FEATURES USED

Feature ID	Feature Name	Used	Feature ID	Feature Name	Used
1	Non-Stop Word Match Share	0	26	Q1 to Q2 Unique Word Count Difference	1
2	TF-IDF Word Match Share	1	27	Q1 to Q2 Unique Word Count Ratio	1
3	TF-IDF Non-Stop Word Match Share	0	28	Q1 to Q2 Unique Non-Stop Word Count Difference	1
4	Common Unigrams Count	0	29	Q1 to Q2 Unique Non-Stop Word Count Ratio	0
5	Common Unigrams Ratio	0	30	Q1 to Q2 Character Difference	1
6	Jaccard	1	31	Q1 to Q2 Character Length Ratio	0
7	Common Words	0	32	Q1 to Q2 Unique Word Character Difference	1
8	Common Non-Stop Words	1	33	Q1 to Q2 Unique Word Character Ratio	1
9	Total Unique Words	1	34	Q1 to Q2 Unique Non-Stop Word Character Difference	1
10	Total Unique Non-Stop Words	1	35	Q1 to Q2 Unique Non-Stop Word Character Ratio	0
11	Same Start Word	1	36	Alternative Non-Stop Word Match Share	1
12	Word Count Difference	1	37	TF-IDF Alternative Non-Stop Word Match Share	1
13	Word Count Ratio	0	38	Alternative Common Unigrams Count	1
14	Unique Word Count Difference	0	39	Alternative Common Unigrams Ratio	0
15	Unique Word Count Ratio	0	40	Alternative Unique Non-Stop Word Count Difference	1
16	Unique Non-Stop Word Count Difference	0	41	Alternative Unique Non-Stop Word Count Ratio	1
17	Unique Non-Stop Word Count Ratio	1	42	Alternative Unique Non-Stop Word Character Difference	1
18	Character Difference	0	43	Alternative Unique Non-Stop Word Character Ratio	1
19	Character Length Ratio	0	44	Alternative Q1 to Q2 Unique Non-Stop Word Count Difference	1
20	Unique Word Character Difference	0	45	Alternative Q1 to Q2 Unique Non-Stop Word Count Ratio	1
21	Unique Word Character Ratio	0	46	Alternative Q1 to Q2 Unique Non-Stop Word Character Difference	1
22	Unique Non-Stop Word Character Difference	0	47	Alternative Q1 to Q2 Unique Non-Stop Word Character Ratio	1
23	Unique Non-Stop Word Character Ratio	0	48	IBM Keyword Share	1
24	Q1 to Q2 Word Count Difference	1	49	IBM Keyword Share Count	1
25	Q1 to Q2 Word Count Ratio	0	50	IBM Keyword Ratio	1

Instead of analyzing specific false positive and false negative question pairs in the confusion matrices in Fig. 7, we decided to analyze the result when using the best language model we have, namely the GPT-3 model, to try to identify duplicate posts on Piazza. Again, CNN was used as the prediction model. For each target Piazza post we gave, we selected the pair that was determined by the model as a duplicate and the model has at least 90% of confidence about its decision. For the duplicated pairs identified by the model, we searched the posts and identified the courses where they are from. The result is shown in Table 17, 18, 19, and 20, where Table 17 shows possible duplicates of some logistical posts from different courses, Table 18 shows possible duplicates of specific questions from Foundations of Programming (FP), Table 19 shows possible duplicates of specific questions from Computer Organization (CO), and Table 20 shows possible duplicates of specific questions from Modern Web Programming (MW). In each table, the posts in column *Post 1* are the target post provided by us, and the posts in column *Post 2* are the possible duplicates identified by the trained model. Column *Post 1 Source* and *Post 2 Source* indicate the source of the two posts from our data, using the tag mentioned in Table 1. The last column shows the confidence of the prediction. Note that only cases with confidence that is greater or equal to 90% are shown in the table.

For Table 17, the test data we gave were all logistical posts. We found that by setting a high limit for the prediction confidence, the result is satisfying for most cases. Most of the results returned by the model were reasonable and most of them should be considered as duplicates. Due to the features we used for training the model, the confidence was affected by the closeness of the two questions in format and length. Despite the dropped confidence, the model did

TABLE 17. PIAZZA DUPLICATE POSTS IDENTIFICATION

ID	Post 1	Post 2	Post 1 Source	Post 2 Source	Confidence (%)
1.1	Where will the recorded lecture be posted?	Where can we find the recorded lecture?	FP1	FP1	99.580
1.2	Where will the recorded lecture be posted?	Hello, could someone tell me where to find the recorded course lectures, thanks.	FP1	FP3	97.949
1.3	Where will the recorded lecture be posted?	Where do we suppose to find the recorded lecture? I have search the shared Google folders but found nothing.	FP1	FP6	92.836
2.1	When will the midterm scores be available on Sakai?	When will the midterm 1 scores be posted on Sakai?	CO4	CO2	98.356
2.2	When will the midterm scores be available on Sakai?	When will the midterm grades be available?	CO4	FP2	95.486
2.3	When will the midterm scores be available on Sakai?	Are the midterm score available?	CO4	MW1	91.079
2.4	When will the midterm scores be available on Sakai?	It was mentioned in the previous instructor note that the grades for midterm II will be released yesterday on Sakai, but it was still unavailable. I wonder when can we have access to our midterm grades, thank you.	CO4	CO1	90.083
3.1	Where can we find the office hours for this week?	Where can we find the office hours information?	CO5	FP2	98.986
3.2	Where can we find the office hours for this week?	Has the schedule for this week's office hours been posted?	CO5	CO3	95.768
3.3	Where can we find the office hours for this week?	Where can we find the TA's Zoom link for each office hour session?	CO5	CO5	93.387
3.4	Where can we find the office hours for this week?	I know I had post the same question yesterday but still got no reply, so I just post it again. Where can we find the office hours for this week? I appreciate the help.	CO5	CO5	91.075
4.1	What should we watch/study to prepare for the midterm on Monday?	I was wondering how I can prepare for the midterm? Should we just look at the quiz and prepare based on it?	FP3	FP3	94.291
4.2	What should we watch/study to prepare for the midterm on Monday?	Will there be any exam review/practice? How do you recommend we prepare for the exam? Thank you.	FP1	CO5	93.469
4.3	What should we watch/study to prepare for the midterm on Monday?	How can we best prepare for Monday's midterm? What are we expected to know/ what material and resources should we review? What format should we expect?	FP3	FP1	92.881
4.4	What should we watch/study to prepare for the midterm on Monday?	Is there any answer key / explanations to the quizzes we took to prepare for the midterm? I noticed professor Bishop stopped going over the quizzes in class and I think it would be very beneficial to prepare for the midterm if we had quiz answers / explanations.	FP3	CO3	91.086
4.5	What should we watch/study to prepare for the midterm on Monday?	Is there an answer key we can use to see if we answered the questions from the practice exam correctly? Also are there any additional resources(such as even older exams) that we could use to help prepare for next week's exam?	FP3	FP4	90.001

sometimes show great performance even when one post is twice or more in length as the other one such as cases 2.4 and 3.4, meaning that the model was capable of learning the meaning of the posts and returned the right prediction. However, for some cases such as cases 3.3, 4.4, and 4.5, the outputs were incorrect and the two posts are asking different questions. We reason that since the two posts are related to the same topic and shared some keywords such as "office

hours" and "midterm", the features extracted using the IBM API will have a strong indication that the two posts are duplicates. Another critical problem of the result shown in the table is the source that each post comes from. For example, case 1.1 may be helpful to a student who posts the second post if the first post has been answered since they are in the same course of the same semester. However, for case 2.3, a student who enrolled in Modern Web Programming will not benefit from knowing when the midterm score of Computer Organization will be released. Even though the two posts look like duplicates and belong to the same category when labeled by our automatic labeling system, they come from different courses, which is a piece of information that we did not input into the model during the training phase. Our original solution was to add the source of each post to the model and set the limit so that the model will only search for the same class of the same semester for potential duplicate posts if the target post belongs to the "Logistical" category. However, we soon found that even when posts 1 and 2 are from the same course of the same semester, the duplicate post detection will not necessarily benefit the student who posts the duplicate question. For example, in case 4.1, the first post is about the midterm on Monday, which is not necessarily the same midterm mentioned in the second post since it is common for a class to have multiple midterms. To avoid this problem, the information about when the posts are created should be added to the model during the training phase, but this cannot solve all the remaining problems. For example, in case 3.4, according to the post content, it is likely that the two posts are from the same person who did not receive an answer from his previous post. As a result, it is not helpful to inform the student about his duplicate posts and give him the link to his previous post. The two problems mentioned above show the new challenge for this task, indicating that even if the model has a strong capability to learn the semantic meaning of the post, sometimes it is still not enough to benefit the student given that the real condition is much more complicated even if the duplicates identification result is considered correct.

For Table 18, 19, and 20, the test data we gave were all other non-logistical category posts, especially student questions about course assignments. Strictly speaking, some of the duplicates identified by the model are not absolute duplicates. However, even if the identification for some posts is not exactly correct, the returned results are always highly related, which may also benefit students who asked similar questions. For example, in cases 13.1 - 13.6, even if the students are asking different third-party API in post 2, they will be potentially benefited if the API they are asking happen to be in the answer of post 1. Moreover, we noticed that it is sometimes the quality of the post rather than the model capability that prevents the prediction from getting better. For example, in cases 5.1 - 5.4, the target post did not mention in the content about the test it is failing, which makes the identification difficult. We also found that the model is still capable to return highly related posts or truly duplicated posts even if the post given contains long and complicated contents that involve error messages that are output by the program. For example, in cases 7.1 and 8.1, despite the long error message shown in the post, the model successfully identified the post that shares a very similar error message as the target post. The students that post the two questions should have the same problem, but the actual questions that they asked make the two posts differ. A similar situation happens in cases 10.1 - 10.2 and 12.1 - 12.2, where the model found other posts that shared very similar error messages to the target posts. According to the questions mentioned in these posts, the model correctly found the duplicates. We think the utilization of the IBM Watson Natural Language Understanding API is critical for the model's performance. Since the words that appeared in the error message are uncommon in the English dictionary, the API can extract them and help the model find posts with similar error messages.

We also found that compared to more lengthy and complicated constructive questions, the model was more likely and accurately to identify duplicates for logistical and active posts, which are mostly short and simple questions. For example, when using test sets where the target posts are longer than 1,000 tokens, the model could find duplicate posts for none of them. Part of the reason is due to the Quora training data, despite a much more even distribution on the data, contains mostly short and simple questions, which weakens the model's ability to find duplicates for long and complex posts. Another major reason is the property and complexity of the posts themselves since it is more likely to find duplicates for logistical posts as these are always the problems shared by many students. For active posts, since there is no reason or attempts specified, these questions tend to be vaguer and have fewer keywords, which makes them easier to match another active post. On the contrary, constructive posts usually show detailed reasoning or attempt, which indicates the specific part about a problem that the student wants to know. This makes the posts personalized and hence more difficult to match another post.

Overall, not only did the model achieve high accuracy on the Quora test data set, but it was also able to find duplicate or highly similar posts when given Piazza posts. However, due to the complexity of the real condition, it still needs improvement to be put into practical use.

5.3. Incomplete Office Hours Requests Identifier

Since the identification problem has been split into two independent binary classification problems, we decided to show the result of the time and reason component classification separately.

5.3.1. Time Component. The data distribution for the time component is shown in Fig. 8

From Fig. 8, it is clear that about $\frac{3}{4}$ of the requests do specify the meeting time, which shows that most students have a good habit of requesting office hours.

The classification result of the time component is shown in Fig. 9. Similar to Fig. 2, and 6, each data points indicate

TABLE 18. PIAZZA DUPLICATE POSTS IDENTIFICATION CONTINUED

ID	Post 1	Post 2	Post 1 Source	Post 2 Source	Confidence (%)
5.1	I passed all tests on local checks, but failed the one in the image below on Gradescope. Which should I trust?	When I run local checks, I have passed the test for SceneControllerButtons, but failed the test on Gradescope. What should I do?	FP6	FP5	97.656
5.2	I passed all tests on local checks, but failed the one in the image below on Gradescope. Which should I trust?	When I submitted my project to Gradescope I noticed that the WaitingAvatars test did not pass, even though it passed on my local checks. Is there any indications as to why this may be? Thanks	FP6	FP4	93.268
5.3	I passed all tests on local checks, but failed the one in the image below on Gradescope. Which should I trust?	I'm having trouble passing the bridge and button scene dynamics test in Gradescope which are passing on my local checks. I verified that my JFrame is not being called in my controller, so I'm not sure what it's attempting to do.	FP6	FP4	91.580
5.4	I passed all tests on local checks, but failed the one in the image below on Gradescope. Which should I trust?	When I run the local checks in Eclipse, I pass all the regular tests. However, when I upload it on the Gradescope, one of the waiting tests failed. I also uploaded the jar and checkstyle file, but still have the same problem. I wonder whether you will upload the new jar / checkstyle file or it is the problem of the Gradescope. Thank you!	FP6	FP6	90.388
6.1	Does the knight have to "walk" into the gorge or across the bridge, or can it just appear?	Can we make the knight appear on the bridge scene instead of walking across the bridge?	FP2	FP3	96.366
7.1	I keep getting this error: "W***public void grail.ABridgeScene. addPropertyChangeListener (java.beans.PropertyChangeListener) should be associated with annotation: @ObserverRegisterer (Property Listener)" However, when I add the annotation "@ObserverRegisterer(Property Listener)" it keeps giving me errors?	I'm getting the following warnings: W***public void grail.composites.AGorge. addPropertyChangeListener (java.beans.PropertyChangeListener) should be associated with annotation: @ObserverRegisterer (Property Listener). Are we able to safely ignore this without losing points?	FP3	FP4	96.995
8.1	W***Use annotation @ObserverRegisterer(interface java.beans.PropertyChangeListener) for method public static void grail.composites. ABridgeScene. addPropertyChangeListener (grail.composites. BridgeScene, java.beans. PropertyChangeListener) W***Use annotation @ObserverRegisterer (interface java.beans. PropertyChangeListener) for method public static void grail.composites. AnAvatar.addPropertyChangeListener (grail.composites. Avatar,java.beans. PropertyChangeListener) W***Use annotation @ObserverRegisterer (interface java.beans.PropertyChangeListener) for method public static void grail.composites.AGorge. addPropertyChangeListener(grail. composites.Gorge,java.beans. PropertyChangeListener) We were never given explicit instructions to add this annotation. Should we go ahead and do it?	W***Use annotation @ObserverRegisterer (interface java.beans. PropertyChangeListener) for method public static void grail.composites.ABridgeScene. addPropertyChangeListener (grail.composites.BridgeScene, java.beans.PropertyChangeListener) W***Use annotation @ObserverRegisterer(interface java.beans. PropertyChangeListener) for method public static void grail.composites.AnAvatar .addPropertyChangeListener (grail.composites.Avatar, java.beans.PropertyChangeListener) Tried adding the ObserverRegisterer and it still wouldn't fix it. If I already submitted the assignment would i lose points?	FP4	FP4	98.899

TABLE 19. PIAZZA DUPLICATE POSTS IDENTIFICATION CONTINUED

ID	Post 1	Post 2	Post 1 Source	Post 2 Source	Confidence (%)
9.1	For our purposes, what is the difference between addi and addiu? They both sign-extend the immediate, don't forget, but what is the difference in the handling of overflow?	I am a little bit confused about the difference between addi and addiu. I know they both sign-extend the number, but I do not understand what is meant by addiu to not check for overflow.	CO1	CO1	98.992
9.2	For our purposes, what is the difference between addi and addiu? They both sign-extend the immediate, don't forget, but what is the difference in the handling of overflow?	Could anyone tell me the difference between addi and addiu? I used addi in the quiz and the correct answer is addiu. Any help is appreciated.	CO1	CO2	96.582
9.3	For our purposes, what is the difference between addi and addiu? They both sign-extend the immediate, don't forget, but what is the difference in the handling of overflow?	The only difference between the instructions in questions 5 and 6 are that one uses addi and the other uses addiu. However, don't both of these instructions add numbers in the same way?	CO1	CO2	91.043
10.1	Does anybody have suggestions about what might be causing "Error unresolved symbol: "$a0" at location 0x00000000[lui $sp, 0x0008]"?	Hi all, I've gone through the initial commenting and attempted translation from C to MIPS and I am receiving this message: Error unresolved symbol: "$t0" at location 0x00000000[lui $sp,0x0008] This seems to be an error occurring with the initial call to "push the stack far away". Any insight would be greatly appreciated, thanks!	CO3	CO3	93.365
10.2	Does anybody have suggestions about what might be causing "Error unresolved symbol: "$a0" at location 0x00000000[lui $sp, 0x0008]"?	I'm getting a "Error unresolved symbol: "$0" at location 0x00000000 [j skipdata]" error when I try to run my code. Nor entirely sure what this means since it seems to be a problem with the setup code and $0 should always be available.	CO3	CO3	93.289
11.1	For the MIPS syntax, do we use the registers such as add $t0 ...etc or do we use the variables they give us such as add f...etc?	When we're asked to write MIPS code, are we supposed to write using the registers or the given variable name?	CO2	CO5	96.386
11.2	For the MIPS syntax, do we use the registers such as add $t0 ...etc or do we use the variables they give us such as add f...etc?	Can we use variable names in MIPS? Or do we have to use registers?	CO2	CO3	95.876
12.1	I keep getting an error, "Cannot read directly from text segment!0x0000300e" - I understand what this error means, but I've verified that my input arrays are correctly stored. Can anyone help me about it? Thank you.	When I am attempting to load the values into registers to complete the matrix multiplication, I got the following error, "Cannot read directly from text segment!0x0000300e". Why does this happen?	CO4	CO4	95.360
12.2	I keep getting an error, "Cannot read directly from text segment!0x0000300e" - I understand what this error means, but I've verified that my input arrays are correctly stored. Can anyone help me about it? Thank you.	I am receiving this error when attempting to load a value stored in my BB array: Runtime exception at 0x00003150: Cannot read directly from text segment!0x00003004. I do not understand why I am receiving this exception.	CO4	CO4	94.656

the average accuracy of ten independent experiments for each language model - prediction model combination.

According to Fig. 9, all language model - prediction model combinations labeled all office hours requests correctly on the time component. We credited the success to not only the models we used but also the features we extracted. Although we only had three features for the time component classification, it included almost all means that students can use when specifying the meeting time with the instructors, which made the classification accurate.

Since all model combinations did perfectly in the time component classification, we decided not to show and ana-

TABLE 20. PIAZZA DUPLICATE POSTS IDENTIFICATION CONTINUED

ID	Post 1	Post 2	Post 1 Source	Post 2 Source	Confidence (%)
13.1	What API satisfies the 3rd party API requirement mentioned in the final project guideline?	What does it mean to have a 3rd party API? What API counts as a 3rd party API for our final project? Any examples?	MW1	MW2	95.328
13.2	What API satisfies the 3rd party API requirement mentioned in the final project guideline?	Will using the Fetch API satisfy the 3rd party API requirement for our project?	MW1	MW2	94.963
13.4	What API satisfies the 3rd party API requirement mentioned in the final project guideline?	Would implementing a 3rd party API that validates an email address as existing meet the API requirement of the final project?	MW1	MW1	93.687
13.3	What API satisfies the 3rd party API requirement mentioned in the final project guideline?	For our project, does creating a RESTful API count as a third party API?	MW1	MW2	93.386
13.5	What API satisfies the 3rd party API requirement mentioned in the final project guideline?	Part of the guidelines for the project is to consume a 3rd party API. Is the git that was provided to us count as a 3rd party API or do we need to find another one?	MW1	MW1	92.890
13.6	What API satisfies the 3rd party API requirement mentioned in the final project guideline?	Hello, for our final project we are trying to use GroupMe's API to log users in and then subsequently use that login token to add that user to a group of their choice based on a search-inputted class name. Does that count as a 3rd party API mentioned in the final project guideline? Or should we look to other APIs.	MW1	MW2	91.079
14.1	Is there a word limit for the tweet?	Does anyone know how long a tweet can be?	MW2	MW2	98.276
15.1	Is it sufficient to have the like button be active as a way of showing whether the current user has liked the tweet?	Is it okay if we remove the like button for tweets that are our own since we can't like our own tweet? Or should we have the like button in there for all tweets?	MW2	MW1	91.213
16.1	Can someone tell me how to retrieve a tweet?	Can someone explain why this cannot retrieve the tweet?	MW2	MW1	96.287
16.2	Can someone tell me how to retrieve a tweet?	I'm having trouble retrieving tweets with reply type using index endpoint. Does anyone know how to do it? Thanks for the help.	MW2	MW2	93.890
16.3	Can someone tell me how to retrieve a tweet?	When I try to retrieve the tweets, I am getting the last 50 tweets followed by the last 49 tweets followed by 48 etc. Anyone know why this might be happening?	MW2	MW2	92.469

lyze the confusion matrix and specific office hours requests.

An example of the requests identified as lacking the time component and the corresponding automatically generated response for these requests can be found in Table 21. Here, we selected the requests that are complete in the reason component but lack the time component. If a request lacks both the time and reason components, the automatically generated response will be extended accordingly to include the notification about the reason component. Examples of this condition will be given in the reason component section.

according to Table 21, the responses to all requests that lack time components are identical and students who made the request will receive this response and be asked to specify the meeting time.

5.3.2. Reason Component. The data distribution for the reason component is shown in Fig. 10

From Fig. 10, about 65% of the requests do specify the problem that the student has. However, there is a 10% decrease compared to the requests that specify the meeting time, meaning that more students are either unaware of the concreteness of their requests or unwilling to give enough specification to their requests.

The classification result of the reason component is shown in Fig. 11. Similar to Fig. 9, each data points indicate the average accuracy of ten independent experiments for each language model - prediction model combination.

According to Fig. 11, we found that there is still a clear trend that more modern language models have a better per-

TABLE 21. AUTOMATIC RESPONSES FOR OFFICE HOURS REQUESTS THAT LACK TIME COMPONENT

ID	Office Hours Request	Automatic Reply
1	Hi Le, I would like to come to OH today. I am working on parts 3 and 4 for A3. I am also confused by my localchecks because the test file only shows a few tests that are not comprehensive of the whole assignment. Thank you!	Please set a specific time to meet with the TA.
2	Hi Ken, could I join to figure out why my bridge scene controller avatar tests are failing? They were passing in the previous jar but now it is no longer working. Thank you	Please set a specific time to meet with the TA.
3	Hey, I would like to stop by OHs today to discuss issues with running object editor on my A1 assignment. I've been importing the same commands that were used in previous praxis but I receive a compile error once they're in the class.	Please set a specific time to meet with the TA.

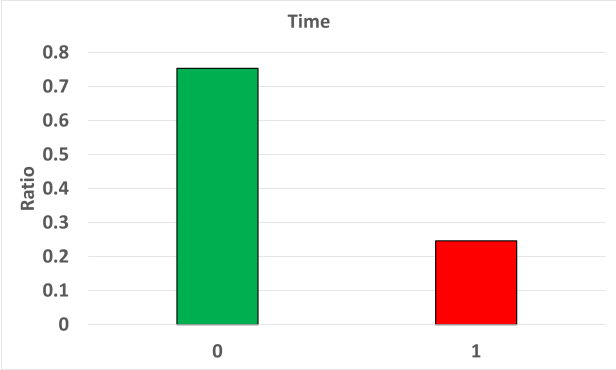

Figure 8. Distribution of office hours requests that lack or do not lack the time component. "0" indicates that the office hours request specifies the meeting time with the instructor and "1" indicates that it does not.

formance than the older language models. However, similar to what we found in Fig. 2, there is no significant difference between different prediction models. Our explanation for this is that since the amount of training data we had for the models is relatively small, the neural network models do not have a great advantage over the other two models. Also, since we extracted ten features for all the models to learn, which captures the key components of the requests, it helps to close the narrow gap between different prediction models.

The average accuracy in Fig. 11 is higher than the one in Fig. 2, which is mainly due to the decrease of the degree of the classification label. However, it is lower than the average accuracy shown in Fig. 6, which is the accuracy of another binary classification problem. This is partly due to the reduced amount of data we had to train the model. Also, since the data is labeled by us instead of experienced human experts and the idea behind reason completeness is relatively vague and subjective compared to duplication, the different opinions about the label determination of some requests can confuse the model, which lowers its accuracy.

The confusion matrices of different language models on the reason component classification are shown in Fig. 12. To keep consistent with previous confusion matrices figures, we used CNN as the prediction model, and the values in the confusion matrices are from a single experiment.

The result in Fig. 12 indicates that the models are more likely to give incorrect predictions when an incomplete office hours request is given, which is reasonable since the input training data is biased and there are twice as many office hours requests that are incomplete as the complete ones.

To visualize the highly confused office hours requests, we selected and analyzed some of them in Table 22, where the "complete" and "incomplete" in the table indicate only the completeness in the reason component. The structure of this table is identical to Table 15.

After examining the incorrectly categorized requests, we found that more capable language models such as DeBERTa and GPT-3 generally perform better than less capable language models such as Word2Vec and GloVe and are more likely to give results that are identical to human labels when it comes to confusing requests.

To examine the automatic response generated for requests that lack the reason component, we selected some examples in which all six language models agree on its deficiency in stating the reason. There is no specification of the language model we used for this table since keywords of each request that inform students about what they should specify are extracted by the IBM API, which is independent of the language model, all models will output the same response once the request is classified to be incomplete. The results are shown in Table 23, where the first two cases show the example response of the model when the request lacks only the reason component, case 3 to 4 show the example response of the model when the request lacks both the time and reason components and the last two cases show the example response of the model when the request contains too little information to extract any keyword.

From Table 23, we found that even if the model cannot give very detailed steps to the students about how they could make the request more concrete like what we did in the analysis in Table 22, most of the keywords extracted accurately indicates the concepts that the students should elaborate on, which should help them complete their request. Also, the model correctly informs students whose requests are too short to give more information, which will help the instructors know their problems better.

Overall, the models did perfectly when identifying office

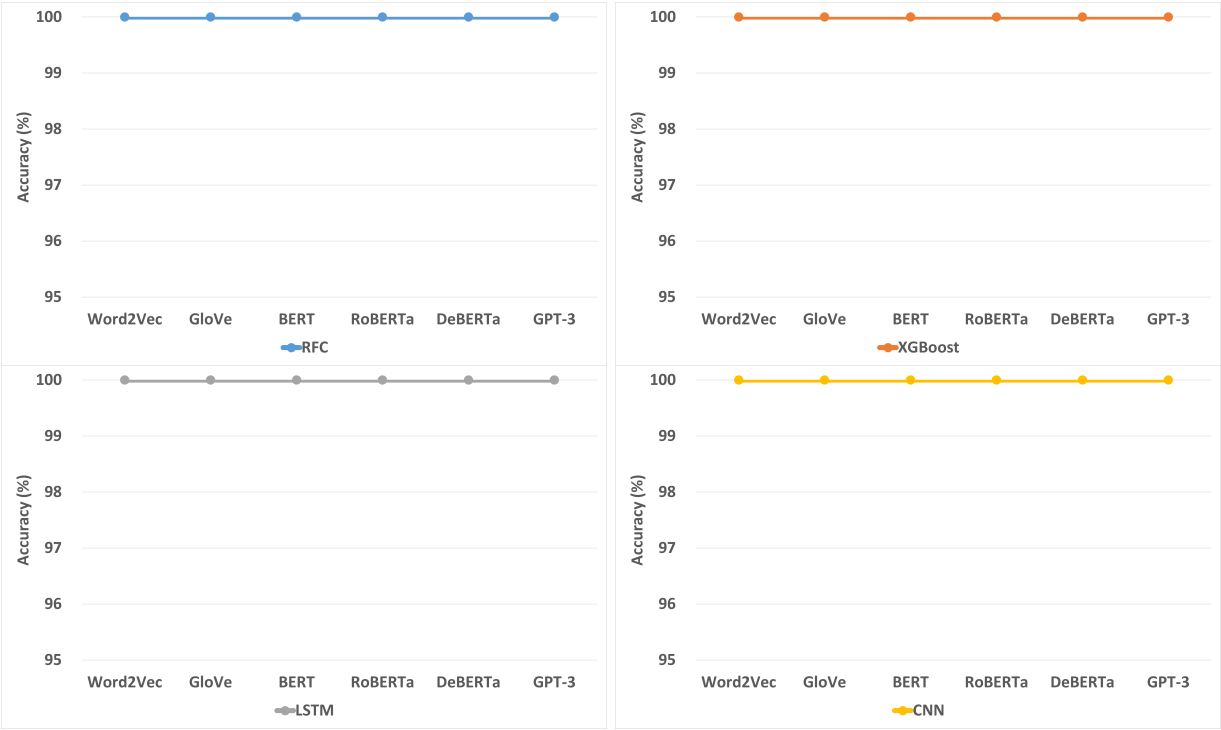

Figure 9. Average accuracy of ten independent experiments of all language models with different prediction models on classifying office hours requests that lack time components. The training data of the prediction models is 80% of the embedded office hours requests with features of all 4 courses in Table 7. The test data of the prediction model is the rest 20% of the embedded office hours requests with features of all 4 courses in Table 7. The plots are split based on the prediction model used to avoid overlapping.

TABLE 22. HIGHLY CONFUSING OFFICE HOURS REQUESTS ANALYSIS

	Request 1	**Request 2**	**Request 3**
Request Body	11/10 10 am - clarify questions about load evaluator. I feel like I'm really close	Came in 9/15, 2:50. Asked about isInferringMainTest() and how it gets tested.	I will be joining office hours today at 2:30 pm regarding the interpolated functions in A1. I am also struggling on setting up the generateSafeDistancesAndDurations() function.
Word2Vec	Complete	Incomplete	Complete
GloVe	Complete	Incomplete	Complete
BERT	Incomplete	Incomplete	Complete
RoBERTa	Incomplete	Complete	Complete
DeBERTa	Incomplete	Complete	Complete
GPT-3	Incomplete	Complete	Complete
Human	Incomplete	Complete	Incomplete
Analysis	This request should be incomplete since it does not give a specific description of the problem. The request did mention the "load evaluator", which is a keyword extracted by the IBM API and misled some less capable language models. The request only mentioned the clarification problem but did not specify what the problem is.	This request is relatively short compared to other complete requests, which makes the less capable models struggle. However, it is considered as complete since it mentions a specific method in the assignment (isInferringMainTest()) and clearly states what the student wants to know about the method (how it gets tested)	This is an arguable request. Indeed, the request mentions specific functions such as "generateSafeDistancesAndDurations()", but only saying "setting up" the function is not specific enough. Also, the phrase "regarding the interpolated functions" makes the problem vague as there may be multiple interpolated functions and the student did not specify what aspect of the interpolated functions did he need help with.

hours requests that lack time component and achieved high accuracy when detecting requests that did not describe the problem specifically enough. Also, the response generated by the model is useful in helping students complete their

TABLE 23. AUTOMATIC RESPONSES FOR OFFICE HOURS REQUESTS THAT LACK REASON (AND TIME) COMPONENT

ID	Office Hours Request	Automatic Reply
1	Hi Justin, I'd like to come to office hour at 11 to discuss the demoing of my view class.	Your request is not specific enough. Try to give more information about one or more of the following concepts: "demoing", "view class"
2	Hey Nalin, I was hoping to join your office hours at 2 to ask a couple of quick questions about the checks for A2. Thanks.	Your request is not specific enough. Try to give more information about one or more of the following concepts: "quick questions", "checks for A2"
3	Hi Ken. I would like to talk about part3. Thanks!	Please set a specific time to meet with the TA. Your request is not specific enough. Try to give more information about one or more of the following concepts: "part3"
4	Hey Andrew, I would like to come to tomorrow's OHs regarding my BridgeSceneDynamics test	Please set a specific time to meet with the TA. Your request is not specific enough. Try to give more information about one or more of the following concepts: "BridgeSceneDynamics test"
5	8.27 2:30pm	Your request is too short. Please give enough information about what you would like to ask.
6	8/18 at 11:00 am	Your request is too short. Please give enough information about what you would like to ask.

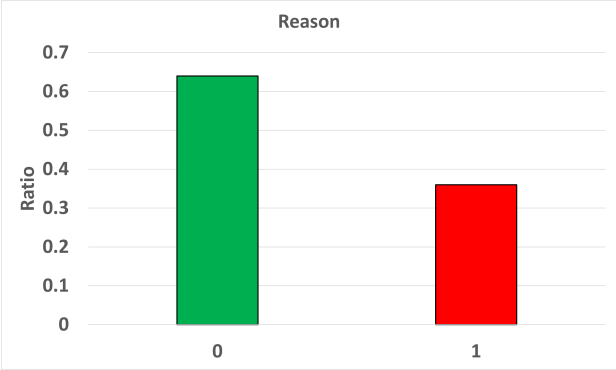

Figure 10. Distribution of office hours requests that lack or do not lack the reason component. "0" indicates that the office hours request specifies the problem that the student wants to discuss with the instructor and "1" indicates that it does not.

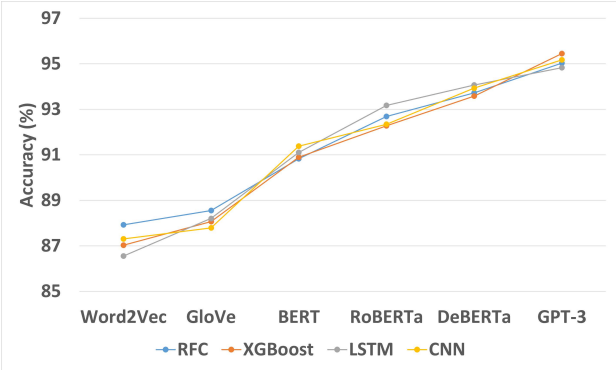

Figure 11. Average accuracy of ten independent experiments of all language models with different prediction models on classifying office hours requests that lack reason components. The training data of the prediction models is 80% of the embedded office hours requests with features of all 4 courses in Table 7. The test data of the prediction model is the rest 20% of the embedded office hours requests with features of all 4 courses in Table 7. The plots are split based on the prediction model used to avoid overlapping.

requests, which helps the instructors make better preparation for the office hours session.

5.4. Automatic Assignment Assistant

We first tested our model's ability to map the function/methods that students ask to the actual name in the code summarization map by inputting our designed questions into the model and letting the model output the actual names in the map that match the ones in students' questions. To have a clear idea of the model's performance, we output the intermediate steps including the keywords extracted by the IBM API and the Bert similarity results. The results are shown in Table 24. Note that for the Bert similarity results (columns 4 and 5), the word before "-" is the word that appears in the extracted keyword (column 2), and the word after "-" is the word recorded in the programming language or function/method name array. The value after ":" is the output similarity of the two words. Also, only cases with similarities greater than or equal to 0.9 are shown in the table.

A variety of cases are tested using the sample questions. according to Table 24, in case 1 and 5, the questions do not specify the programming language, but in both cases, the model output the expected result. The difference is that in case 1, there is one match since there is only one function/method named "tf_idf_word_match_share" in the code summarization map key set, but in case 5, three summarizations will be returned since the method in the sample question appears three times in the key set. In case 2, the question asks multiple implementations associated with one programming language. Again, the model correctly returns the intended result. In case 3, the question asks multiple implementations associated with multiple programming languages. The extracted keyword managed to catch MIPS as the programming language but missed C, which is mainly due to the ambiguity of its meaning. This problem can be solved by adding an if statement to specifically search for C as a whole word. In case 4, even though the question is

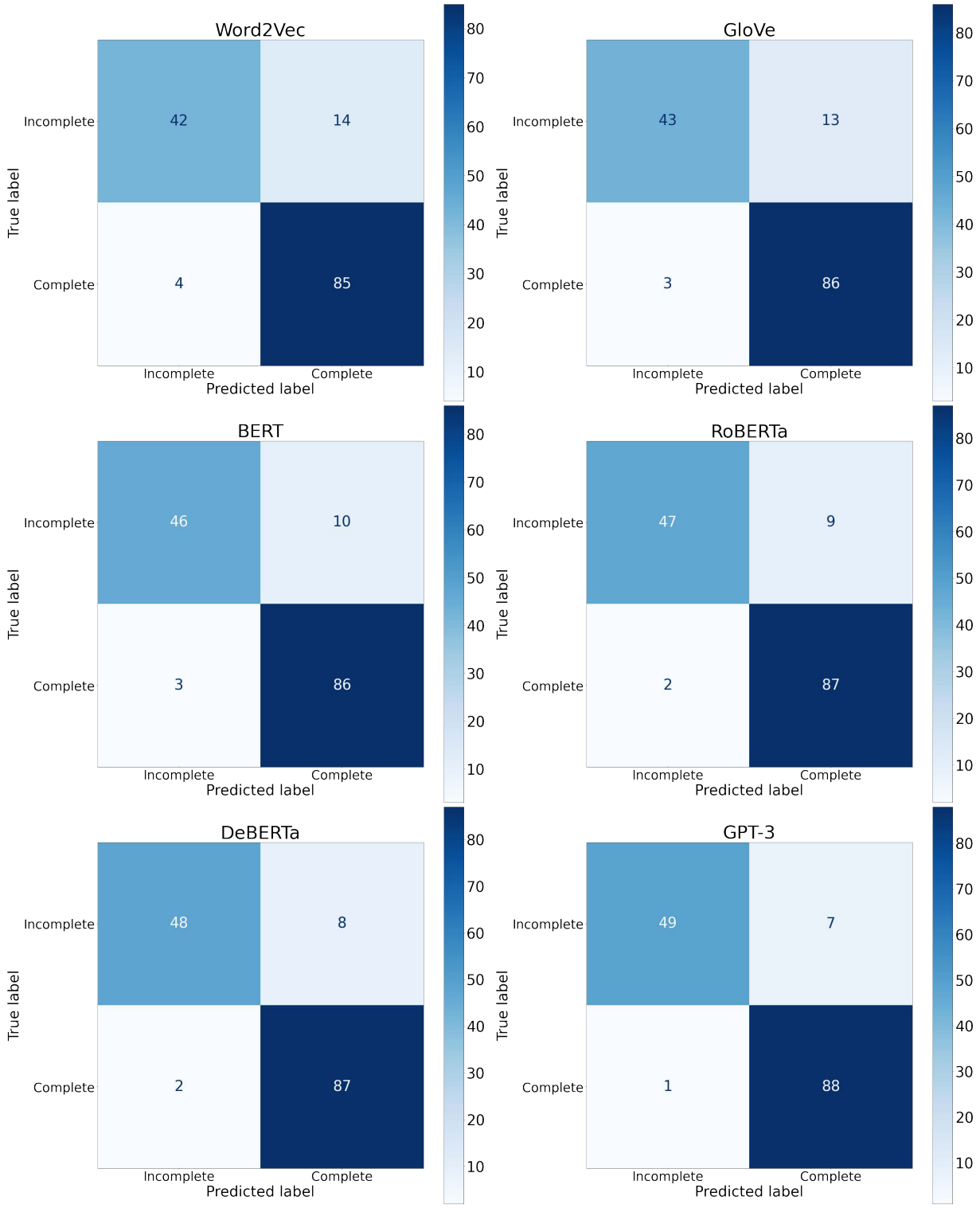

Figure 12. Confusion matrices of the Word2Vec, GloVe, BERT, RoBERTa, DeBERTa, and GPT-3 language model on classifying office hours requests that lack reason components. The prediction model is CNN. The training data of the prediction models is 80% of the embedded office hours requests with features of all 4 courses in Table 7. The test data of the prediction model is the rest 20% of the embedded office hours requests with features of all 4 courses in Table 7. The x-axis stands for the label predicted by the prediction model and the y-axis stands for the true label. The result is based on one experiment for each language model.

TABLE 24. AUTOMATIC ASSIGNMENT ASSISTANT MAPPING

ID	Sample Question	Extracted Keyword	Programming Language Similarity	Function / Method Name Similarity	Result
1	Can someone tell me how to implement the tf_idf_word_match_share function?	[tf_idf_word_match_share function]	/	tf_idf_word_match_share-tf_idf_word_match_share: 1.000	Match: Python: tf_idf_word_match_share
2	How to implement the Java "isGivenSafe" and "interpolatedDistance" methods?	[Java, isGivenSafe, interpolatedDistance]	Java-Java: 1.000	isGivenSafe-isGivenSafe: 1.000 interpolatedDistance-interpolatedDistance: 1.000	Match: Java: isGivenSafe Java: interpolatedDistance
3	I would like to ask about the C and MIPS implementation of the fibonacci and the factorial function.	[MIPS, fibonacci, factorial]	MIPS-MIPS: 1.000	fibonacci-Fibonacci: 1.000 factorial-Factorial: 1.000	Match: MIPS: Fibonacci MIPS: Factorial
4	What should be do in the prolog and sml isgivensafe and interpolateddistance function?	[isgivensafe, interpolateddistance function, prolog, sml]	prolog-Prolog: 1.000 sml-SML: 1.000	isgivensafe-isGivenSafe: 1.000 interpolateddistance-interpolatedDistance: 1.000	Match: Prolog: isGivenSafe SML: isGivenSafe Prolog: interpolatedDistance SML: interpolatedDistance
5	How should be implement the interpolatedDistance method?	[interpolatedDistance method]	/	interpolatedDistance-interpolatedDistance: 1.000	Match: Java: interpolatedDistance Prolog: interpolatedDistance SML: interpolatedDistance
6	What is the MIPS implementation of this part in the assignment?	[MIPS implementation of this part, assignment]	MIPS-MIPS: 1.000	/	It looks like you are asking for the implementation of a function/method. To get the English translation of the implementation you need, please specify the function/method name.

trickier than that of case 3 by changing the case of both the programming language names and the function names, the keyword extraction process successfully catches all of them and the model managed to output all summarization that the question is asking. In case 6, the question does not specify the function/method name but specifies the programming language, which is identified by the model. As a result, it works as intended by informing the student to specify the function/method name.

Next, we tested the model's ability to summarize the code snippets. The result of the summarization of each code snippet in Table 11 can be found in Table 25 and 26.

Generally, the results in Table 25 and 26 are not satisfying. For about half of the cases, the model returns code instead of English summarization. For these cases, the model either rewrites the code snippet into Python code (case 4 and 5) or rewrites using the same programming language as the one used in the snippet (cases 7, 8, and 10). In most of these cases, the rewriting output is misleading since the output code completely changed the key component of the original code (cases 7, 8, and 10). For other cases, the model tried to summarize the code, but its summarization is either incorrect or irrelevant. For example, in case 1, the model explained how to calculate the TF-IDF weight of a word, which is different from our description for this feature. Also, one could not follow the guidance of the output to get the TF-IDF weight of a word since the guidance did not make sense if taken a closer look. In case 3, the summarization stated that the isGivenSafe() method is a combination of the isSafe() method and the isIntermediate() method, then it began to summarize the functionality of the two methods, both of which do not appear in the snippet. In case 9, the model was trying to rephrase the snippet into another form by giving the input and result combination. However, it got the last three results wrong and mistakenly output false. In case 11, there is no evidence of the appearance of the C code or Python code. In case 12, instead of illustrating how the MIPS code makes the Fibonacci sequence work, the model was comparing the snippet to a "previous version", which does not exist as well.

A possible explanation for the behavior above is the corpus used to train the model. Since the model was trained using 45TB of data, some of the data likely contain code that is very similar to the code snippets in the example. As

TABLE 25. AUTOMATIC ASSIGNMENT ASSISTANT TEST CODE

ID	Programming Language	Function / Method Name	Code	Summarization
1	Python	tfidf_word_match_share	See Code Listing 1	The function tfidf_word_match_share() returns the tf-idf weight of a word in a question. The tf-idf weight of a word is the product of the tf-idf weight of the word in the question and the tf-idf weight of the word in the answer. The tf-idf weight of a word in the answer is the tf-idf weight of the word in the question divided by the tf-idf weight of the word in the answer. The tf-idf weight of a word in the question is the tf-idf weight of the word in the question divided by the tf-idf weight of the word in the question.
2	Python	jaccard	See Code Listing 2	The Jaccard similarity coefficient is a similarity coefficient between two sets. It is defined as the size of the intersection divided by the size of the union of the sets.
3	Java	isGivenSafe	See Code Listing 3	The isGivenSafe() method is a combination of the isSafe() method and the isIntermediate() method. The isSafe() method is called with the distance, duration and exhalationLevel parameters. The isIntermediate() method is called with the distance, duration and exhalationLevel parameters. The isIntermediate() method returns true if the distance is less than or equal to the distance threshold, the duration is less than or equal to the duration threshold and the exhalationLevel is less than or equal to the exhalationLevel threshold. The isIntermediate() method returns false if the distance is greater than the distance threshold, the duration is greater than the duration threshold or the exhalationLevel is greater than the exhalationLevel threshold.

4	Java	interpolatedDistance	See Code Listing 4	def interpolatedDistance(distance):
				if distance < SMALL_DISTANCE:
				return MIN_DISTANCE
				elif distance < MEDIUM_DISTANCE:
				return SMALL_DISTANCE
				elif distance < LARGE_DISTANCE:
				return MEDIUM_DISTANCE
				else:
				return LARGE_DISTANCE

5	C	Factorial	See Code Listing 5	def factorial(n) if n == 0 1 else n * factorial(n-1)
6	C	Fibonacci	See Code Listing 6	The above code is a Python implementation of the Fibonacci sequence.
7	Prolog	isGivenSafe	See Code Listing 7	safeFact(Distance, Duration, Exhalation) safeFact(0, _, _) = true safeFact(Distance, Duration, Exhalation) = safeFact(Distance, Duration, Exhalation - Exhalation) end.
8	Prolog	interpolatedDistance	See Code Listing 8	mediumDistance(DistanceMedium) :- DistanceMedium = Distance * 2. smallDistance(DistanceSmall) :- DistanceSmall = Distance / 2. largeDistance(DistanceLarge) :- DistanceLarge = Distance * 10.

a result, the model can detect this similarity and output the actual code in its corpus or the analysis associated with the code, which seems close but is irrelevant to us.

For some cases, the model gave some correct or relevant information, but it did not go in-depth about the implementation. For example, in case 6, the model correctly identifies that the implementation is about the Fibonacci sequence, but it did not give any details about how the code implements the sequence. Also, in case 2, the model correctly caught the key that Jaccard is the intersection of the sets divided by the union of the sets, but it did not specify what the sets are in our case.

Overall, the model did well in identifying the programming language and function/method names and can achieve high accuracy if students in a class are instructed to specify both parts if they need help on the implementation of some parts of the assignment. However, an alternative model needs to be used to translate the code in a more meaningful

TABLE 26. AUTOMATIC ASSIGNMENT ASSISTANT TEST CODE CONTINUED

ID	Programming Language	Function / Method Name	Code	Summarization

9	SML	isGivenSafe	See Code Listing 9	distance = 13;
				duration = 30;
				exhalation = 30;
				isGivenSafe(distance, duration, exhalation) = true;
				distance = 6;
				duration = 30;
				exhalation = 10;
				isGivenSafe(distance, duration, exhalation) = true;
				distance = 27;
				duration = 30;
				exhalation = 50;
				isGivenSafe(distance, duration, exhalation) = true;
				distance = 6;
				duration = 15;
				exhalation = 30;
				isGivenSafe(distance, duration, exhalation) = true;
				distance = 13;
				duration = 120;
				exhalation = 10;
				isGivenSafe(distance, duration, exhalation) = false;
				distance = 27;
				duration = 120;
				exhalation = 30;
				isGivenSafe(distance, duration, exhalation) = false;
				distance = 6;
				duration = 15;
				exhalation = 30;
				isGivenSafe(distance, duration, exhalation) = false;

10	SML	interpolatedDistance	See Code Listing 10	distance = interpolatedDistance(distance);
				if distance $<$ smallDistance then
				minDistance = distance;
				else if distance $<$ meidumDistance then
				smallDistance = distance;
				else if distance $<$ largeDistance then
				meidumDistance = distance;
				else
				largeDistance = distance;
				end if;

11	MIPS	Factorial	See Code Listing 11	The output is the same as the C version. The Python version is a bit longer than the C version, but it is still quite short. The Python version is also a bit slower than the C version, but not by much.
12	MIPS	Fibonacci	See Code Listing 12	The output is the same as before. The code is a little more complicated than the previous version, but it is still straightforward. The only new feature is the use of the mult instruction to multiply the accumulator by the value in the accumulator. This instruction is a little more complicated than the addi instruction, but it is still easy to understand. The mult instruction multiplies the accumulator by the value in the accumulator. The result is placed in the accumulator. The instruction is written as mult s0,v0. The $s0 register is the accumulator, and the $v0 register is the value to multiply by. The result is placed in the accumulator. The mult instruction is a little more complicated than the addi instruction because it has two operands. The first operand is the accumulator. The second operand is the value to multiply by. The value to multiply by is placed in the accumulator.

and reliable way before this module can be put into practice.

6. Conclusion and Future Work

6.1. Conclusion

This paper presents different modules of a practical system that can be used to greatly alleviate instructors'

burden on online education forum posts processing. It introduced different approaches to solve different classification or identification problems. It involves different strategies to clean the data and introduced different features extracted for different purposes. It also included different language models and prediction models and compared the performance difference between different combinations. Moreover, it showed how to utilize multiple advanced APIs to improve the models.

In the automatic labeling system module, we managed to achieve the highest accuracy of over 90% using the GPT-3 model, which shows the model's ability in multiclass classification.

In the duplicate posts identifier module, we managed to achieve the highest accuracy of over 98% using the GPT-3 model during the Quora testing phase and showed that the trained model had a strong capability to search for highly related posts of the target posts.

In the incomplete office hour request module, we managed to achieve 100% accuracy when identifying the time component and a highest accuracy of over 95% using the GPT-3 model for the reason component. We also showed that the model was capable to identify the deficiency in students' requests and inform them about it.

In the automatic assignment assistant module, despite the unsatisfying summarization of the code snippet, we successfully showed that the model is capable of identifying students' questions about specific code implementations.

The methods and models introduced in this paper can serve as the basis for further improvement and inspire functionality expansion. The modular design of the system is also helpful for functionality that needs to be added or disabled to fit different classes' needs.

6.2. Future Work

6.2.1. Accuracy Improvement. Despite the accuracy we have achieved in this paper, there is still plenty of room for us to further increase the model accuracy, which can mainly be achieved in three parts: fine-tuning, parameter adjustment, and model combining.

All language models introduced in section 3.1 are pre-trained for general-purpose natural language processing. However, our tasks are computer science specific and involved many terms about the subject, which prevents the models from achieving higher accuracy. To solve this problem, all language models should be fine-tuned for our task. This can be achieved by further training on the model using the course posts and office hours requests we have as well as the labels we provided.

Also, as mentioned in section 4.1.4, we left most parameters of our models to be the default values, which hinders the models to achieve better performance since the optimal parameters should be different for each task. In the future, it is necessary to fine-tune the models by adjusting the parameters. One reasonable way is to use the control variable method, which fixes all the parameters except one. By changing the parameters gradually within a wide range, take the value where the model reaches its best performance as the optimal value for the parameter.

While one model may mistakenly assign an incorrect label to a post or request, combining the decision of multiple models can avoid this problem. For each task, it is better to train multiple prediction models independently and combine them. A simple combining can be achieved by geometric averaging the prediction of more than ten models.

6.2.2. Model Completeness. As mentioned in the introduction section, the paper only shows the kernel of each module. To make the system fully functional, a database needs to be set to store students' posts and office hours requests, an incomplete post identifier needs to be implemented to strength the system's ability of post preprocessing, a connection module needs to be built to serve as the communication medium between the forum and the kernel, and a user interface needs to be added to allow the instructors to change settings of the system and add or disable functionalities of the kernel more easily. It is also used to notify the instructors of specific students' posts or actions that they should take.

To set up the database, we decided to use the Python *sqlite3* package, which provides a lightweight disk-based database and allows accessing the database using a non-standard variant of the SQL query language which we are similar with.

To implement the incomplete post identifier, we planned to use the same data set as mentioned in section 4.3.1. We decided to create the trainning data for the models by manually labeling all posts with instructors answers or replies into two categories: 1 for Complete and 0 for Incomplete. The post will be considered as incomplete only if the instructor specifies in the reply to the post that more information is needed and will be considered as complete otherwise. Once the trainning data is available, we will train and test the model using procedures similar to what is mentioned in section 4.3.

To build the connection module, we decided to utilize the Piazza auto-grader tool developed by George et al. The tool is mainly designed to fetch, auto-grade, and return the grades of students' diary, a special post that records students' class attendance and activity. We planned to revise the fetching part, remove the auto-grading part, and improve the sending part to enable the tool to get any posts and send customized messages to anyone in the class.

To add the user interface, we decided to use the Atlas toolkit for Python, a quick and easy way to add versatile graphical interfaces with networking capabilities to Python programs. The toolkit can be embedded in the Jupyter notebook, which is the same program we used to develop the kernel.

6.2.3. Functionality Expansion. One of the advantages of our implementation is that it offers high modularity, which allows easy functionality updates and expansion. While the functionality presented in the paper can offer great help to the instructors, more functionality can be added to further

encourage learning, improve teaching, and even provide data for student awareness research.

The model built in the automatic labeling system module can be used not only for labeling new posts but also for encouraging students to think deeper about the course material. For example, when a student posts a question that is classified by the model as an Active question, the model can send a private message to the student by encouraging him to provide more thoughts on the problem such as his perspective on the problem or his attempt to solve the problem, which transfers the Active question to a Constructive question.

The model can also be used to help the instructors do better in the class. For example, the model can be refined to identify sudden increases of Content Clarification problems within a short period of time and inform the instructors if this happens. Since most Content Clarification problems are related to assignments and projects, this will show the instructors the necessity to clarify the assignments or projects, and improve their teaching efficiency in the future semesters.

Another helpful functionality expansion of the model is the student answer evaluator. On an educational forum like Piazza, both instructors and students can answer questions. If a student has already provided a satisfying answer to a question, there is no need for the instructor to look at the post and answer again. The student answer evaluator is a proposed module that can help the instructor skip posts that have already been provided with a satisfying answer by a student. For each of these posts, the model will examine it and output a number between 0 and 1 to indicate how helpful the answer is. If the value is large enough, the evaluator can endorse the student answer for the instructors automatically so that the instructors can skip that question. Having achieved success in our duplicate post identifier module, we decided to utilize an external data source provided by human experts. The data we had was 9,647 question-answer pairs collected by CrowdSource team at Google Research. Each pair in the data is associated with a value indicating the helpfulness of the answer, which is determined by human experts. We also planned to introduce new techniques when training this new model such as pseudo-labeling, an advanced and helpful model training technique developed recently that makes the model learn on its prediction results while labeling new data, and blending, an ensemble machine learning technique that uses a machine learning model to learn how to best combine the predictions from multiple contributing ensemble member models.

Moreover, Vellukunnel et al [32] found that there is a correlation between the number of questions that students ask on the online educational forum and their final grades. They also showed a similar correlation between the number of constructive questions that students ask on the online educational forum and their final grades. Inspired by their findings, we decided to refine the automatic labeling system module to not only label the new posts but also associate the number of each type of question that a student posts to the student's ID. According to Vellukunnel et al's findings, the model can be used to predict students' final grades based on the number and type of questions they have asked during the semester.

6.2.4. More Capable Code Summarization Model. As mentioned in section 5.4, the summarization result of the code snippets is not satisfying. This is mainly due to the training data of GPT-3, which includes mostly natural language instead of code. To make code summarization better, OpenAI is developing a more specific model named Codex [31], the descendants of the GPT-3 models that are trained specifically to translate between natural languages to various types of programming languages. Their training data contains both natural language and billions of lines of public code from GitHub.

There are three Codex models: Codex, Codex-S, and Codex-D. Codex is the basic Codex model, which is obtained after code fine-tuning GPT-3 with a different number of parameters. Codex-S is the model obtained by fine-tuning on Codex's Supervised fine-tuning data set [31] to improve its performance on HumanEval, which is a data set containing 164 manually written programming problems used to evaluate language comprehension, reasoning, algorithmic ability, and simple math. Codex-D is the model obtained by fine-tuning on Codex's Docstrings data set [31] to train Codex's ability to generate Docstrings for programming languages.

The Codex models are by far the most powerful programming language pre-trained models. They're most capable in Python and proficient in other programming languages including JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, SQL, and even Shell. Currently, the model is in private beta and cannot be accessed by us. Once it is opened to the public, we can replace our GPT-3 model with it in our automatic assignment assistant module and makes the code summarization more helpful and informative to students.

Acknowledgments

The authors would like to thank Dr. Prasun Dewan, the advisor of this program, for offering precious guidance and resources. We also gratefully thank Dr. John Majikes, the second reader of this paper, for reviewing the paper and offering constructive improvement suggestions.

References

[1] Do, J., *Dynamic Tagging of Educational Forum Posts Using NLP Post Embeddings*, 2021.

[2] Li, Y., Yang, T., Srinivasan, S., *Word Embedding for Understanding Natural Language: A Survey*, 2018.

[3] Devlin, J., Chang, M., Lee, K., Toutanova, K., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019.

[4] He, P., Liu, X., Gao, J, Chen, W., *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*, 2021.

[5] He, P., Gao, J, Chen, W., *DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing*, 2021.

[6] Pennington, J., Socher, R., Manning, C., *GloVe: Global Vectors for Word Representation*, 2014.

[7] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, 2019.

[8] Goldberg, Y., Levy, O., *word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method*, 2014.

[9] Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N., Peters, M, Schmitz, M., Zettlemoyer, L., *AllenNLP: A Deep Semantic Natural Language Processing Platform*, 2018.

[10] Zhu, H., Paschalidis, I., Tahmasebi, A., *Clinical Concept Extraction with Contextual Word Embedding*, 2018.

[11] Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., Vollgraf, R., *FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP*, 2019.

[12] Tomas, M., Kai, C., Greg, C., Jeffrey, D., *Efficient Estimation of Word Representations in Vector Space*, 2013.

[13] Tomas, M., Ilya, s., Kai, C., Greg, C., Jeffrey, D., *Distributed Representations of Words and Phrases and their Compositionality*, 2013.

[14] Mnih, A., Hinton, G., *Three new graphical models for statistical language modelling*, 2007.

[15] Koeman, J., Rea, W., *How Does Latent Semantic Analysis Work? A Visualisation Approach*, 2014.

[16] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., *Improving Language Understanding by Generative Pre-Training*, 2018.

[17] Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., Bowman, S., *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*, 2020.

[18] Dayhoff, J., *Neural network architectures: an introduction*, 1990.

[19] Dhillon, A., Verma, G., *Convolutional neural network: a review of models, methodologies and applications to object detection*, 2020.

[20] Olien, L., Bélair, J., *Bifurcations, stability, and monotonicity properties of a delayed neural network model*, 1997.

[21] Lipton, Z., Berkowitz, J., Elkan, C., *A Critical Review of Recurrent Neural Networks for Sequence Learning*, 2015.

[22] Yu, Y., Si, X., Hu, C., Zhang, J., *A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures*, 2019.

[23] Hochreiter, S., Schmidhuber, J., *Long Short-Term Memory*, 1997.

[24] Floridi, L., Chiriatti, M., *GPT-3: Its Nature, Scope, Limits, and Consequences*, 2020.

[25] Dale, R., *GPT-3: What's it good for?*, 2021.

[26] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., *Language Models are Unsupervised Multitask Learners*, 2019.

[27] Tom B., B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A. et al, *Language models are few-shot learners*, 2020.

[28] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., Polosukhin, I., *Attention is All you Need*, 2017.

[29] Rajeswaran, A., Finn, C., Kakade, S., Levine, S., *Meta-Learning with Implicit Gradients*, 2019.

[30] Finn, C., Abbeel, P., Levine, S., *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*, 2017.

[31] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. et al, *Evaluating Large Language Models Trained on Code*, 2021.

[32] Vellukunnel, M., Buffum, P., Boyer, K., Forbes, J., Heckman, S., Mayer-Patel, K. et al, *Deconstructing the Discussion Forum: Student Questions and Computer Science Learning*, 2017.

[33] Chaturvedi, S., Goldwasser, D., Daumé III, H., *Predicting Instructor's Intervention in MOOC forums*, 2014.

[34] Zhang, W., Sheng, Q., Lau, J., Abebe, E., *Detecting Duplicate Posts in Programming QA Communities via Latent Semantics and Association Rules*, 2017.

[35] Ahasanuzzaman, M., Asaduzzaman, M., Roy, C., Schneider, K., *Mining Duplicate Questions of Stack Overflow*, 2016.

[36] Hindle, A., Barr, E., Gabel, M., Su, Z., Devanbu, P., *On the naturalness of software*, 2016.

[37] Ray, B., Hellendoorn, V., Godhane, S., Tu, Z., Bacchelli, A., Devanbu, P., *On the "Naturalness" of Buggy Code*, 2016.

[38] Bansal, A., Haque, S., McMillan, C., *Project-Level Encoding for Neural Source Code Summarization of Subroutines*, 2021.

[39] Haque, S., LeClair, A., Wu, L., McMillan, C., *Improved Automatic Summarization of Subroutines via Attention to File Context*, 2020.

[40] LeClair, A., Haque, S., Wu, L., McMillan, C., *Improved Code Summarization via a Graph Neural Network*, 2020.

[41] Ahmad, W., Chakraborty, S., Ray, B., Chang, K., *A Transformer-based Approach for Source Code Summarization*, 2020.

[42] Chen, T., Guestrin, C., *XGBoost: A Scalable Tree Boosting System*, 2016.

[43] Swain, P., Hauska, H., *The decision tree classifier: Design and potential*, 1977.

[44] Liaw, A., Wiener, M., *Classification and Regression by randomForest*, 2002.

[45] Sagi, O., Rokach, L., *Ensemble learning: A survey*, 2018.

[46] Lewis, R., *An Introduction to Classification and Regression Tree (CART) Analysis*, 2000.