

Recommendation System for Stack Overflow Questions

Machine Learning for Model-Driven Engineering

Mariama Oliveira

Department of Information Engineering, Computer Science and Mathematics

University of L'Aquila

Email: mariamaceli.serafimdeoliveira@student.univaq.it

1. Introduction

Stack Overflow is a question-and-answer website focused on computer application development. Founded in 2008, it has a structure similar to that of platforms like Wikipedia and Reddit where users can edit posts, ask questions, provide answers, and upvote or downvote both questions and answers [1].

Daily, Stack Overflow processes a vast quantity of data. According to their website, they have registered over 28 million users and host more than 24 million questions and 36 million answers [2]. For instance, in January 2025, over 25,000 new questions were posted on the platform. While this number is relatively small compared to the same timeframe in January 2020, which saw more than 147,000 posts - likely due to the rise of large language models like ChatGPT - it remains crucial to find experts who can address these questions. As fewer people utilize these tools, it becomes even more essential to identify the right users who can provide answers aligned with their expertise.

For instance (see Figure 1), 17.1% of the questions posted in 2019 remain unanswered, and this trend continues in subsequent years. For this reason, this study explores classical approaches to leverage recommendation systems that can suggest appropriate answerers for new questions posted on the Stack Overflow platform. While other studies [3], [4] have tackled this issue with some success using supervised learning methods, this study focuses on classical techniques such as content-based filtering and collaborative filtering, as it is part of a Machine Learning course and the main goal was to learn the foundation and characteristics of these methods. Additionally, a transformer model was used to extract embeddings from the questions and ultimately provide recommendations in a content-based fashion. Therefore, the contributions of this report are as follows:

- 1) Extraction and creation of a database using real data from Stack Overflow questions and answers.
- 2) Experimentation with recommendation systems that utilize content-based filtering or collaborative filtering approaches.
- 3) Utilization of a language model to leverage content-based recommendation.

- 4) Analysis and comparison of the different approaches employed in this study.

Answered vs Unanswered Questions (2019-2024)

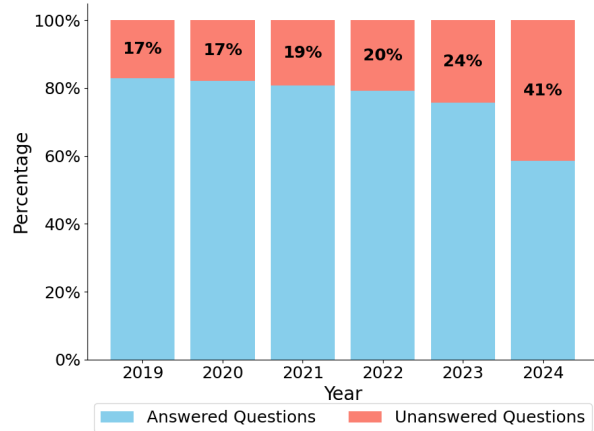


Figure 1: Percentage of answered vs. unanswered questions in Stack Overflow

To achieve these goals, the tools used were primarily Python and related libraries such as Pandas and NumPy. The study resulted in the creation of several Jupyter Notebooks that outline the steps of each approach, making it easy for readers to reproduce the experiments if they choose to do so. All the code used in the study is hosted in a public repository on GitHub¹, and the dataset can be found on Google Drive².

2. Dataset

The dataset utilized in the project was extracted from the Stack Exchange Data Explorer, a platform that allows users to execute SQL queries across various sites in the Stack Exchange network. It contains data on all questions posted on Stack Overflow during the first six months of 2019, totaling 878,718 questions. Additionally, the dataset includes 1,105,677 answers.

¹GitHub: <https://github.com/mariamaOlive/ml4mde>


²Google Drive: <https://tinyurl.com/datasetSO>

How do I undo the most recent local commits in Git?

Asked 15 years, 8 months ago


Modified 25 days ago

Viewed 15.7m times



I accidentally committed the wrong files to [Git](#) but haven't pushed the commit to the server yet.

26919



How do I undo those commits from the *local* repository?

git version-control git-commit undo

(a) Example of question in Stack Overflow.



Undo a commit & redo

29666



```
$ git commit -m "Something terribly misguided" # (0: Your Accident)
$ git reset HEAD~                               # (1)
# === If you just want to undo the commit, stop here! ===
[ edit files as necessary ]                     # (2)
$ git add .                                     # (3)
$ git commit -c ORIG_HEAD                       # (4)
```



1. `git reset` is the command responsible for the **undo**. It will undo your last commit while **leaving your working tree (the state of your files on disk) untouched**. You'll need to add them again before you can commit them again.

+500

(b) Example of answer in Stack Overflow.

Figure 2: Example of question and answer stackoverflow

It is important to note that these answers are not necessarily limited to the same time period, as users can respond to questions at any time, even years after they are posted. For this reason, analyzing a period from five years ago was considered appropriate since users are less likely to post new answers to those questions, making the dataset more stable for analysis.

Figure 2 illustrates an example of a question and its answer on the Stack Overflow platform. From this example, several elements can be identified for extraction and used in recommendation systems. For this project, not all available data regarding questions and answers was extracted; instead, only the information highlighted in red in Figure 2, as described in Tables 1 and 2, was included. The queries used in the Data Explorer can be found in a file in the repository. It is important to note that these queries were executed in batches due to the website's limitation of 50,000 rows per query. As a result, the "CreationDate" was modified during the querying process.

2.1. Brief Data Analysis

Before conducting the experiments, a brief data analysis was performed on the dataset to better understand its information. Figure 3 illustrates several interesting insights.

Figure 3a shows that the majority of questions (79.14%) have up to three answers, while only a small percentage of questions (3.21%) have more than three answers. This indicates that once a question is answered, few other users

TABLE 1: Question Dataset Variables

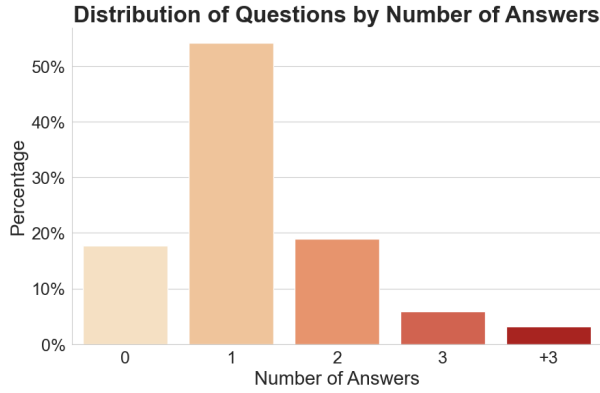
Variable	Type	Description
QuestionId	Int	Unique question ID.
QuestionOwnerId	Int	User ID of the author.
QuestionTitle	String	Question title.
QuestionTags	String	Comma-separated tags.
QuestionVotes	Int	Total vote count.
QuestionCreationDate	DateTime	Creation timestamp.
AnswerCount	Int	Number of answers.

TABLE 2: Answer Dataset Variables

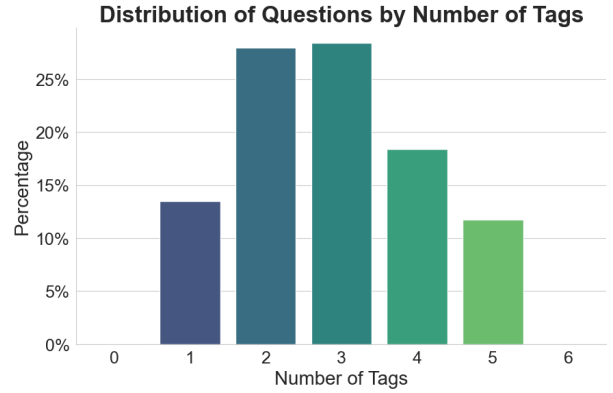
Variable	Type	Description
QuestionId	Int	Associated question ID.
AnswerOwnerId	Int	User ID of the answerer.
AnswerVotes	Int	Total vote count.
AnswerCreationDate	DateTime	Answer timestamp.

engage in providing additional alternatives to the original answer. Another concerning finding is that 17.73% of questions have never been answered, despite more than five years having passed since they were posted.

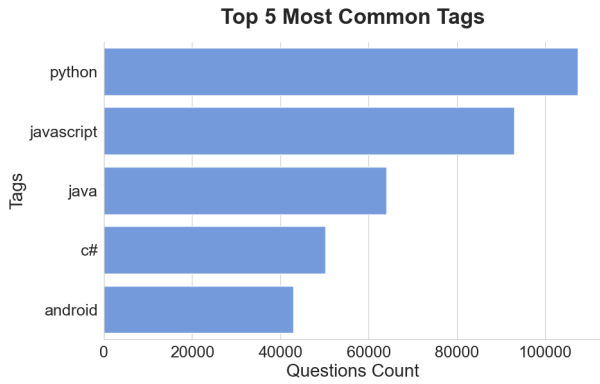
Additionally, an analysis of the number of tags attributed to each question reveals that most users tend to add only two or three tags, even though they are allowed to add a maximum of five (see Figure 3b). The most frequently recurring tags in the system include Python, JavaScript, and Java (Figure 3c). This analysis highlights an important issue:



(a) Number of answers per question

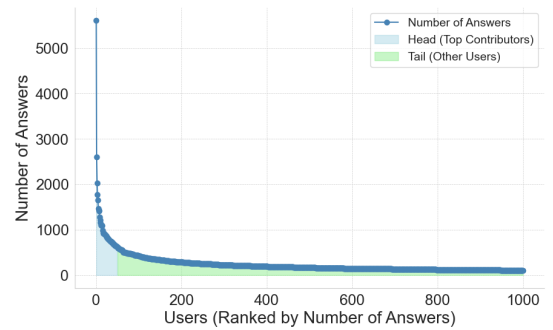


(b) Number of tags per question



(c) Top 5 tags

The Long Tail of User Answer Contributions (Top 1,000 Users)



(d) Long Tail of Answers by Users

Figure 3: Data Exploration on Stack Overflow dataset.

Python accounts for over 10% of the questions in the dataset. This results in an imbalance toward certain topics, which could be detrimental to some recommendation algorithms.

Lastly, the long-tail distribution of answers by users depicted by Figure 3d shows that some individuals are “super responders,” having answered over a thousand questions in the dataset, while the majority of users have answered very few. This disparity might pose a problem for the system, as it may lead to a reliance on these prolific responders for answering newly posted questions.

3. Methodology

This section presents the tasks performed in the dataset before the experiments and the techniques applied to identify the most suitable users to answer a new Stack Overflow question.

3.1. Data Preprocessing for Experiments

Before conducting the experiment, some preprocessing was necessary, although there were not many steps involved since the Data Explorer system provided the data in a

suitable format. The most important preprocessing step was removing questions that had no answers, which comprised 17.73% of the dataset. This was essential because we needed responders to the questions to build the recommendations and evaluate the system. Additionally, date types were converted to the appropriate format, and tags were organized into a list of strings.

3.2. Recommendation Techniques

According to Aggarwal [5], recommendation problems can be framed in two primary ways, as prediction or as ranking. The first approach aims to predict a rating for a specific user-item combination, while the second focuses on determining the top-N items for a particular user. In this study, we are addressing the latter problem since our goal is to identify the most appropriate users within the Stack Overflow system to answer a new question. To tackle this issue, we utilized two different approaches: collaborative filtering and content-based filtering, resulting in five distinct setups - one collaborative-based and four content-based. In the following paragraphs, how each of these approaches works in general will be explained, and how they were applied in the study will also be discussed in detail.

Collaborative-based Filtering

Models using this approach leverage the relationships among multiple users and a specific item to generate recommendations for an individual user. In collaborative filtering, the recommendation system compiles a list of suggestions based on items that users with similar preferences also enjoy.

Content-based Filtering

This recommendation approach uses the characteristics of a user to make suggestions. These characteristics can be derived from the items the user has interacted with or their behavior patterns. Once a user profile is created based on these characteristics, similarity metrics are employed to identify the items that are most similar to those preferred by the user. Notably, it is not necessary to know additional information about other users in order to provide recommendations.

3.3. Experiment Setup

Five different experimental setup approaches were defined to provide the best users for answering a question based on these two approaches.

3.3.1. (Setup 1) - Collaborative-based Filtering using Users' Tags. This setup implements a Collaborative-Based Filtering approach, specifically Item-Based Collaborative Filtering, using user-tag interactions. The goal is to recommend users who are most likely to engage with a given set of tags (a new question) by leveraging tag similarity and historical user engagement. The likelihood of user engagement is determined by predicting the number of interactions a user could have based on the given set of tags. This problem is analogous to estimating a user's rating for a product based on their previous ratings. However, instead of predicting ratings, we aim to estimate a user's level of engagement with tags, given the tags associated with a new question.

This approach followed these steps:

- 1) The first step is to construct a User-Tag Interaction Matrix, where rows represent users, columns represent tags, and each cell value corresponds to the number of times a user has answered a question associated with a given tag. This matrix provides a structured way to capture user engagement across different tags.
- 2) Next, a Tag-Tag Similarity Matrix is computed. This matrix measures the similarity between tags based on user interactions. Tags that frequently appear together across different users will have higher similarity values, indicating a strong relationship between them.
- 3) When a new question is posted, its associated tags are extracted. Using the Tag-Tag Similarity Matrix, the rows corresponding to these tags are retrieved. These rows are then multiplied to generate a single

vector that encapsulates the combined influence of all selected tags. This vector represents the overall tag importance for the new question.

- 4) Once the representative tag vector is obtained, the classical Item-Based Collaborative Filtering Prediction equation (Eq. 1) is applied to estimate user engagement. This formula predicts how likely each user is to interact with the given tags based on past behavior. The result is a predicted interaction score for each user.
- 5) Finally, the predicted values are sorted in descending order, producing a ranked list of users most likely to engage with the newly posted question.

The interaction of user u with representative vector of tags i is predicted using the following equation:

$$Pred(u, i) = \bar{r}_u + \frac{\sum_{j \in S_i} (sim(i, j) \times r_{u,j})}{\sum_{j \in S_i} sim(i, j)} \quad (1)$$

- \bar{r}_u - Average user interaction
- S_i - Set of tags contained in the question
- $sim(i, j)$ - Tag similarity of the representative vector
- $r_{u,j}$ - Number of times the user has utilized tag j .
- $\sum_{j \in S_i} sim(i, j)$ - Normalization factor: Sum of all values in the tag representative vector

3.3.2. (Setup 2) - Content-based Filtering using Users' Tags. This approach implements a Content-Based Filtering method to recommend users based on their past engagement with tags. Instead of leveraging collaborative interactions between users, this method relies solely on the similarity between a new question's tags and the user's historical tag usage.

This approach followed these steps:

- 1) Utilize the same User-Tag Interaction Matrix of the previous setup.
- 2) When a new question is posted, a question vector is then created, where each dimension represents a tag (using the same indexes of the User-Tag Interaction Matrix), and the indices corresponding to the tags of the new question are set to 1, while all others remain 0.
- 3) The next step involves computing cosine similarity between the newly generated question vector and the User-Tag Interaction Matrix. By computing cosine similarity, we determine how closely each user's tag usage history aligns with the new question's tags.
- 4) Once similarity scores are obtained, the users are then ranked in descending order based on their similarity scores.

3.3.3. (Setup 3) - Content-based Filtering using Questions Embeddings (User Profile). Similarly to the previous approach, this method compares the similarity between user characteristics and the newly posted question to recommend

users who are most likely to engage. However, instead of relying on tags, this approach utilizes text embeddings derived from the titles of previously answered questions. The user's profile is constructed by aggregating these embeddings and comparing them with the embedding of the new question's title.

The model utilized to extract the embeddings was all-MiniLM-L6-v2³ model which is a compact, high-performance model within the SentenceTransformers framework [6]. It is optimized for semantic similarity and sentence embeddings. The result of its embedding is a 384-dimensional dense vector. It was chosen for this project due to its ability to produce small embeddings and being computationally lightweight.

This approach followed these steps:

- 1) The first step is to extract the embeddings of the titles of all previously answered questions for each user. These embeddings capture the semantic meaning of the questions. To create a representative user profile, the mean embedding of all the user's answered questions is computed, forming a single vector that characterizes their expertise and interests.
- 2) Once user profiles are constructed, the cosine similarity between each user's profile and the embedding of the new question's title is calculated. This measures how semantically aligned the user's past activity is with the new question.
- 3) After obtaining similarity scores, users are ranked in descending order based on their similarity to the new question.

3.3.4. (Setup 4) - Content-based Filtering using Questions Embeddings. This approach is similar to the previous one, but instead of creating a user profile by averaging the embeddings of previously answered questions, it compares each individual question embedding with the embedding of the newly posted question.

This approach followed these steps:

- 1) The first step is to extract the embeddings of the titles of all previously answered questions for each user. However, instead of averaging these results, we keep them as they are.
- 2) Compare the embedding of the old questions with the new question using cosine similarity.
- 3) Once the most similar questions are ranked in descending order, the users who answered those questions are identified, and a list of users is created.

3.3.5. (Setup 5) - Content-based Filtering using Questions Embeddings + Tags Embeddings. This approach evolves from the previous one by incorporating an additional embedding vector derived from the textual value of the tags linked to each question. Instead of relying only on the question title embedding, this method extracts both the question

embedding and the tag embedding, and then concatenates them to create a more comprehensive representation of the question. Following that, the same steps from the previous setup are applied.

4. Evaluation

This section will describe how the evaluation was performed, the metric utilized, and its results.

4.1. Data Evaluation Split

To evaluate the recommendations from the different experiment setups, the questions dataset was divided into training (90%) and testing (10%) subsets. A smaller test set was chosen compared to the traditional 20% due to computational limitations encountered during the experiments; larger test datasets required significantly more computation time. However, a size of 10% is sufficient to draw stable conclusions, given that the total size of the dataset contains 722,883 questions (training: 650,900, test: 71,983).

The data split was based on a temporal method, with 90% allocated to the earlier questions in the dataset and the remaining 10% consisting of the latest questions, ordered by their timestamps. This approach was implemented to simulate real-life recommendations, which rely on user interactions over time. However, there were concerns that this type of split might lead to certain questions being misrepresented at different times. To address this issue, a brief analysis was conducted to determine whether the proportion of tags was consistent across both subsets. As seen in Figure 4, the results showed that the proportion remained stable for at least the top 10 tags, which account for 89.94% of the data.



Figure 4: Tags distribution in Train and Test set

4.2. Evaluation Metric

For evaluating the system, the Hit Rate (HR) metric was utilized. This metric measures whether at least one of the

³<https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

TABLE 3: Hit Rate Results for Different Top-N Recommendations (%)

Experiment Setup	Top-5	Top-10	Top-20	Top-50	Top-100
Baseline	1.42	1.96	2.16	4.11	6.99
S1 - Collaborative Filtering	14.28	18.36	23.03	30.36	35.99
S2 - Content-Based Filtering	3.24	5.45	8.39	13.56	18.98
S3 - User Profile (NLP)	5.29	7.73	10.90	16.25	20.90
S4 - Question (NLP)	7.86	11.40	15.55	21.83	27.05
S5 - Question + Tags (NLP)	12.92	18.10	23.71	31.80	37.94

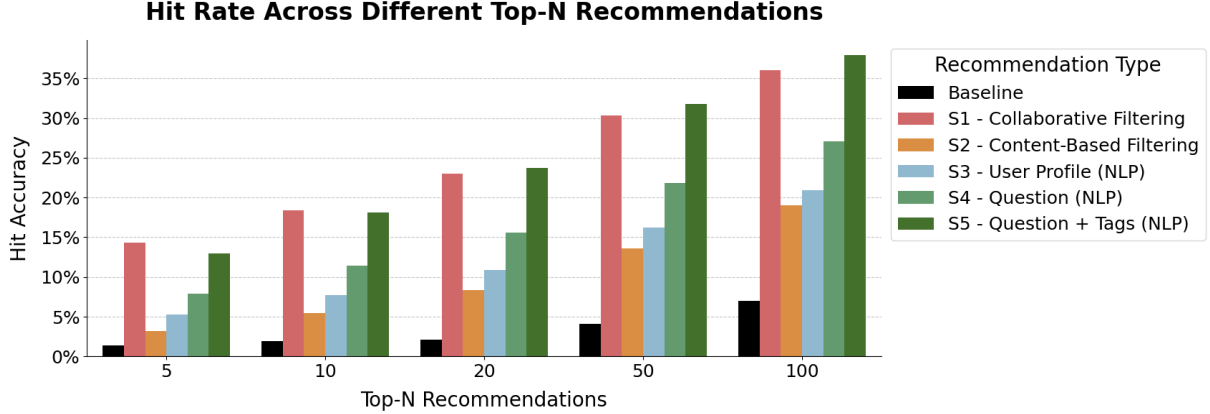


Figure 5: Experiment results

suggested users in the top-n recommendations matches the ground truth. For example, if we measure Hit Rate at Top-5 (HR@5) and we get a question that have three different users who answered it. If at least one of these users appears in the Top-5 recommendations provided by the system, it is considered a hit. The final Hit Rate is calculated as the total number of hits divided by the total number of questions (Eq. 2), providing a measure of how often the system successfully recommends a relevant user within the top-n results.

$$HR@n = \frac{\text{Total of Hits}}{\text{Total of Questions}} \quad (2)$$

4.3. Results

As shown in Table 3 and Figure 5, the methods that achieved the highest hit rate accuracy were Collaborative Filtering (Setup 1) and the Content-Based approach utilizing question and tag embeddings (Setup 5). The method that exhibited the least effectiveness was Content-Based filtering using Setup 2 and 3, indicating that the vector used to describe the question and users was insufficient for comparison. Setup 2 likely suffered due to the sparsity of the User-Tags Matrix, and the user profile vector did not appear to represent the user accurately; perhaps averaging is not the most effective approach.

Another key observation is that accuracy increases as the number of Top-N values grows, which is expected. When more users are considered in the recommendation list, the chances of selecting the correct user also increase. However, this also highlights a limitation—when only a small Top-N

is used, the system may miss other capable users who could have answered the question.

To further validate the effectiveness of these methods, a baseline, represented by the black bar in the results, was introduced. This baseline follows a simple rule: it always recommends the most engaged users. The goal was to verify whether the observed results were due to meaningful recommendations or simply a consequence of some users being significantly more active than others. As observed, all methods outperformed the baseline, indicating that the recommendations are not random but aligned with the ground truth data and provide meaningful suggestions.

5. Conclusion

The results indicate that the Collaborative-Based method (Setup 1) and the Content-Based method using question and tag embeddings (Setup 5) achieved similar outcomes. These two methods displayed significantly better performance in comparison to the other setups. Notably, both rely on tags for recommendations; however, their mechanisms differ. The Collaborative-Based method emphasizes user relationships, recommending users who have interacted with similar tags in the past. In contrast, the Content-Based approach focuses on the similarity of questions to previously answered questions rather than on user relationships.

Through the experiments, it becomes evident that a hybrid approach—combining both methods—could potentially enhance recommendations. Such an approach could leverage the behavioral similarity between users from Collaborative Filtering while also utilizing language model embeddings

to extract deeper semantic relationships for better suggestions. Furthermore, embeddings from language models might serve as an alternative when user behavior data is limited, making them particularly useful in cold start scenarios where users or questions lack prior interactions. Conversely, Collaborative Filtering is more effective when textual data is scarce but sufficient user interaction data exists to infer behavior.

Beyond accuracy, computational and memory constraints were also key challenges in implementing these methods. Due to the matrix-based nature of the algorithms, matrix multiplications significantly exceeded the memory capacity of standard personal computers. To address this, sparse matrices and batch processing were employed to optimize memory usage. However, when working with language model embeddings, sparse matrices were not an option since the generated vectors are dense. Another challenge with language models was the need to extract embeddings for all questions, which required substantial GPU resources. These computational limitations were among the primary obstacles faced during the experiments.

Although the results of this project did not achieve the same metrics performance when compared to works that performed similar tasks in the literature [3], [4], this study provided a valuable learning experience. It allowed for a deeper understanding of these models, their strengths, limitations, and real-world implementation challenges. Moving forward, an intriguing direction for improvement could be incorporating additional features, such as timestamps and upvotes, to enhance the accuracy of recommendations. Furthermore, it would be valuable to verify the diversity of suggestions, as some users are highly engaged contributors, which could lead to them being recommended more often than less active users. Therefore, it's clear that there is ample opportunity for more fruitful work.

References

- [1] Wikipedia contributors, "Stack overflow — Wikipedia, the free encyclopedia," 2025, [Online; accessed 19-February-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stack_Overflow&oldid=1270984221
- [2] Stack Exchange, "Stack Exchange Data Explorer," 2025, accessed: 2025-02-19. [Online]. Available: <https://data.stackexchange.com/>
- [3] L. Wang, L. Zhang, and J. Jiang, "Tea: an answerer recommendation approach on stack overflow," *Science China Information Sciences*, vol. 62, pp. 1–19, 2019.
- [4] B. Shao and J. Yan, "Recommending answerers for stack overflow with lda model," in *proceedings of the 12th Chinese conference on computer supported cooperative work and social computing*, 2017, pp. 80–86.
- [5] C. C. Aggarwal *et al.*, *Recommender systems*. Springer, 2016, vol. 1.
- [6] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: <https://arxiv.org/abs/1908.10084>