Записки научных
семинаров ПОМИ
Том 552, 2026 г.

E. Orlov, V. Malykh

PLEASE UPVOTE: STACK OVERFLOW ANSWER RANKING IN A TEXT-ONLY SETTING

Abstract. Stack Overflow (SO) is a long-standing resource for programming knowledge; however, recent years have seen a decline in user participation and question-asking activity. One factor discussed in prior work and community reports is the social friction associated with asking questions, particularly for newcomers, including concerns about negative feedback and moderation. Improving automated support tools could help reduce unhelpful interactions and make participation more accessible.

In this paper, we study a core component of community question answering on SO: ranking answers for a given question, which can support moderation and content curation tools. Unlike previous work that relies on post and user metadata, we consider a text-only formulation that uses only the question and answer content. This setting enables ranking for newly created accounts without reliable profile features and reduces the risk of amplifying reputation-related biases.

We train a set of LM-based rankers pretrained on SO data using two different ranking approaches and two encoder architectures. We observe that fine-tuned models consistently outperform their counterparts based on frozen embeddings in terms of nDCG. Our best model improves over a strong text-based random forest baseline from prior work. It also outperforms rankers based on general-purpose LMs of similar sizes, highlighting the value of in-domain pretraining. Finally, we analyze performance by question tags and identify a group of tags for which the differences are statistically significant.

§1. Introduction

Stack Overflow (SO) is an important resource for everyone who writes programming code. It helps to find answers to programming-related questions based on a Community Question Answering (CQA) platform. It was launched in 2008 as a collaborative platform where users can ask technical questions and receive answers from the community. Its key innovation is

Key words and phrases: Stack Overflow, community question answering (CQA), answer ranking, learning to rank, transformer models, nDCG.

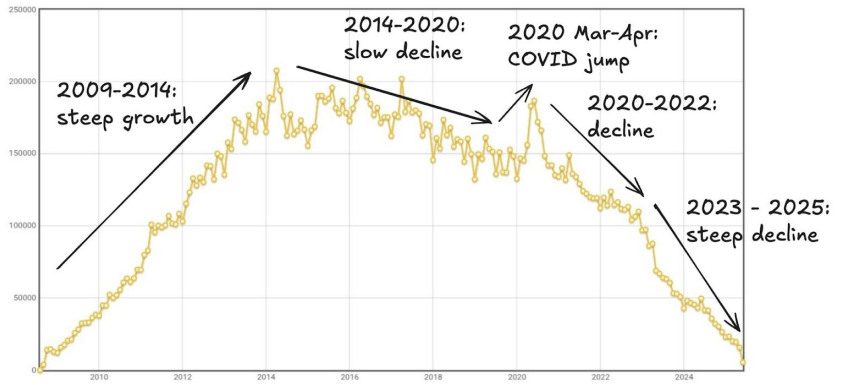


Figure 1. Number of questions asked per month on Stack Overflow (image from [36]).

the voting mechanism which allows users to upvote and downvote both questions and answers, making the most useful content more visible. Additionally, users can accept a single answer to their question, marking it as the most helpful. These mechanisms help create a reputation system that encourages constructive participation and builds trust in user contributions. Over time, SO has become the largest Q&A platform for developers, covering a broad scope of topics ranging from general programming to highly specific technical issues.

However, in recent years SO has suffered from a loss of popularity [36]. SO was once the dominant platform for programming Q&A, but it started losing engagement after 2017. This trend became more visible during the COVID-19 pandemic, when many developers shifted their habits. The platform saw a slow drop in traffic and participation, especially in answering questions. Figure 1 clearly illustrates this trend, displaying the monthly number of questions asked on SO from its launch to the current time. This decline was not due to one reason but rather a combination of structural issues and external pressures.

One major factor was the rise of better alternatives. GitHub Discussions, Discord communities, and Reddit became more attractive for informal or fast feedback. SO was also blamed for failing to adapt to contemporary

audiences' requests. For example, the platform did not make the transition to video answers which are favored by younger developers [35]. More recently, LLMs have disrupted how developers seek help. Tools like ChatGPT allow users to get answers instantly, reducing the need to post or search on SO. These models can also summarize answers or write code, which makes them more convenient for many users.

Another important reason why LLMs are preferred is the lack of social pressure. Unlike SO, they do not punish vague or beginner-level questions. Users do not have to worry about receiving sarcastic comments or being downvoted. They can ask follow-up questions without fear of being told to read the documentation.

This makes LLMs a safer space, especially for people who are just learning or exploring unfamiliar tools. The interaction feels more like a conversation than a performance in front of an audience. As a result, many developers find LLMs more approachable and less stressful to use. For these reasons, they are quickly becoming the default tool for getting programming help.

Meanwhile, SO faced internal problems. Changes in management and unclear product direction led to frustration within the community. A key moment came when many community moderators left during disputes with the company. The recent launch of Stack Overflow's AI product, OverflowAI, also failed to generate excitement. In 2024, layoffs and the shutdown of the Teams free tier further reduced trust. The platform is no longer the first choice for many developers, and recovery seems unlikely without a major shift.

On the other hand, there is evidence that LLMs can make mistakes compared to human programmers when answering code-related questions [19]. In addition, on SO, multiple users can provide answers to the same question, offering a range of perspectives and solutions [54]. This diversity can lead to a deeper understanding of the problem and its potential solutions, whereas LLMs typically provide a single, synthesized response. Given these factors, the SO platform still has value for the developers community and could find its share in the variety of help-giving instruments if it addressed issues that turn users away.

One of the problems SO is facing at the moment is the lack of high-quality automatic moderation. Now it is done mainly by human moderators, with the help of bots that handle the most obvious cases of irrelevant

content. Human moderators are blamed for their biased and aggressive policy, while bots do not make use of modern ML technologies[1]. Automatic moderation could reduce the number of rude or unhelpful comments that discourage participation. New users often receive negative feedback, which can drive them away from the platform. Automatic moderation could detect such behavior early and either flag it or guide the user to rephrase.

It could also improve the clarity and usefulness of posts. By suggesting edits to poorly worded questions or identifying duplicates, automatic tools could help users find answers faster. This would make the site more efficient and beginner-friendly. It could ease the burden on human moderators and reduce community tension. Many conflicts have emerged from disagreements about moderation policies. Automated systems, if transparent and well-calibrated, could follow the standards more consistently. This could lead to a friendlier atmosphere and increase trust in the platform. Finally, introducing automatic moderation based on contemporary AI tools could attract additional interest and raise the profile of SO as a modern platform.

Automatic moderation applied to SO includes many tasks like identifying low-quality questions, restricting toxic behaviour, and marking outdated content. In this work, we focus on SO answer quality assessment. Models trained for this task can be used, for example, to adjust the ranking of answers which have not received enough user votes yet. In Section 5.4, we discuss other possible applications of such models in more detail.

Previous studies have addressed the task of SO answer quality assessment under different problem formulations. [9, 25, 32–34, 44] approach the task as a classification problem, predicting whether each individual answer is accepted. [45, 51, 52] treat it as a ranking problem but focus solely on ranking a single accepted answer highest per question, and therefore use binary relevance scores. Much less research has concentrated on ranking all answers to a question according to their user vote counts [1, 13, 14]. In this work, we contribute to this gap and follow such ranking approach.

In contrast to previous work [1, 13, 14] which uses post metadata in addition to their texts, we formulate our task as text-only. Our models are trained solely on question and answer texts. Such method has several advantages. First, it can be applied in situations when the answers are written by new users and cannot be distinguished by features related to the user profile. This problem is noted, for example, in [14]. Second, it does

[1] `https://github.com/SOBotics/FireAlarm`

not require any hand-crafted features which allows it to be applied off-the-shelf to other CQA domains beyond programming. Third, it reduces potential bias of the models as they do not rely on features like author reputation and do not have the possibility to disadvantage new users.

We work with the dataset used in [13, 14]. It is based on a 2012 Stack Exchange dump and contains SO questions and answers with user votes and additional metadata. We train a line of models based on LMs pretrained on SO data using two different ranking approaches and two encoder architectures. We compare their performance to the models presented in [13, 14] and our own baselines. Furthermore, we compare our best-performing model to models based on general-purpose LMs. We analyze the performance of our models depending on the question tags. Our main contributions are:

- we train a line of models to rank Stack Overflow answers based on the posts' texts using two different ranking approaches and two encoder architectures
- we find that fine-tuned models consistently outperform their counterparts based on frozen embeddings
- our best model outperforms the random forest model based on textual features from [14]
- we find that our ranking model based on an LM pretrained on SO data outperforms models based on general-purpose LMs of comparable sizes
- we find a set of question tags which cause statistically significant differences in the performance of our best model.

§2. Related works

2.1. Community question answering analysis. Stack Overflow belongs to the broader class of CQA platforms. We refer to the systematic survey of ML/DL methods for CQA in [37] and focus here on works that are relevant to our study.

A recurring theme in CQA research is that community feedback is informative but imperfect. [11] show that votes may reflect social effects and strategic behavior in addition to content quality, motivating approaches that do not treat raw votes as unbiased ground truth. Complementing this, [39] analyze the social role of voting on SO and argue that voting practices serve multiple purposes (curation, moderation, feedback), which helps explain why the same score can mix different signals.

Beyond SO, several works study answer selection and ranking primarily from text. [24] model answer selection via sentence-pair similarity and show gains from transformer encoders and fine-tuning pretrained models. [38] propose Support-BERT for assessing question–answer pairs using only text (without hand-crafted metadata), aligning with our text-only setting. [2] incorporate temporal/spatial relations among answers with recurrent architectures, while [40] combine BERT with TextRank-style compression to better handle long answers in ranking.

2.2. Stack Overflow.

2.2.1. *Stack Overflow downstream tasks.* A substantial body of work treats SO as a source of domain-specific language and benchmarks. [41] introduce a software-focused NER dataset and train BERTOverflow to improve entity extraction in programming text. [50] learn post representations from multiple post components (including code) using tag supervision and demonstrate improvements across several SO downstream tasks. [16] and [30] further argue that in-domain pretraining on large-scale SO data yields consistent improvements for SE tasks; we leverage such SO-pretrained encoders in our experiments.

Several papers address adjacent quality-related problems on SO. [42] identify questions with insufficient answer quality ("deficient" posts). [29] predict rejected edits to improve post maintenance. [53] study and predict questions without accepted answers. [4] propose moderation action prediction based on textual features. [27] focus on automatically updating answer posts using comment signals.

2.2.2. *Stack Overflow answers quality assessment.* Our work is closest to research that ranks answers within a question thread using learning to rank objectives. [14] frame answer ranking as regression on vote-derived scores and use rich feature sets (text, user, temporal, and interaction metadata). The authors take a broader look in [13] and propose a general machine learning framework for assessing the quality of collaboratively created content. They treat content quality as a multidimensional concept composed of independent views. Each view represents a group of features derived from different sources, such as text structure, edit history, or user metadata. In their SO experiments, the authors use eight views including structure, readability, style, edit history, user profile, and user graph-based indicators.

The authors propose two variants of the framework which combines different views. MVIEW trains per-view regressors and then a meta-regressor over their outputs, while MVIEW+F0 additionally feeds the meta-regressor the original features from all views (F0). The intuition is that the combination of high-level view scores and low-level features might further improve performance. However, in practice, MVIEW+F0 does not always outperform MVIEW. In some datasets, including SO, MVIEW without F0 achieves better results. This suggests that adding raw features to the meta-model may introduce redundancy or noise, especially when views already capture the most informative aspects.

These two studies provide strong metadata-based baselines, but they rely on signals that are unavailable or unstable at inference time for newly posted answers (e.g., user history, edits, comment activity). In contrast, our paper targets a text-only setting to improve portability and reduce reliance on potentially biased user-related features. Importantly, our evaluation is based on the same dataset that is used in these works which allows for direct comparison.

Other SO ranking work often uses binary relevance (accepted vs not accepted) or mixes text with metadata. For example, [45] build an SO dataset and predict acceptance using text plus numerical features (e.g., reputation, comments). [1] include SO forum answer ranking as a benchmark task (BLANCA), highlighting that domain-specific pretraining can substantially improve ranking quality. [3] propose combining quality prediction with content "recency" features to better surface up-to-date answers, emphasizing that usefulness is not captured by votes alone.

2.2.3. *Stack Overflow modeling.* SO-specific language models have been proposed to better represent the mix of natural language and code in posts. [41] pretrain BERTOverflow on a large corpus of SO posts and show strong gains for software NER. [50] learn post representations with explicit modeling of post structure and code snippets. [16] propose SOBERT (not to be confused with SOBert described below), initialized from CodeBERT and further pretrained on SO, and demonstrate improvements on multiple SO tasks. [30] introduce SOBertBase/Large as SO-pretrained encoders trained on full posts (including code and comments), showing consistent gains on quality/moderation-style tasks. Due to the fact that SOBert models are reported to outperform all the aforementioned ones on SO downstream tasks, we adopt them in our experiments.

2.2.4. *LLMs as alternatives to Stack Overflow.* Recent work studies how LLMs change developer help-seeking behavior and how their answers compare to SO. [19] compare ChatGPT answers with accepted SO answers and show that perceived quality can diverge from correctness, which motivates automated assessment beyond surface fluency. [22] and [23] study real-world use of AI assistants and their mixed productivity/security effects, suggesting that developers may need better signals for validating generated content. In academic settings, [48] highlight both the convenience of LLM assistance and the risks of subtle errors.

SO can also be used to mitigate LLM weaknesses. [31] propose SOSecure, a retrieval-augmented approach that uses relevant SO discussions to revise generated code, and relate this direction to vulnerability-focused RAG such as Vul-RAG [15].

2.3. Learning to rank. Learning to rank (L2R) optimizes an ordering of items rather than independent labels. Our task (ranking answers for a given question) naturally fits this paradigm.

2.3.1. *Approaches to ranking.* Point-wise approaches learn a relevance score per item (e.g., PRank [12]). Pair-wise approaches learn preferences between item pairs; classical examples include RankingSVM [17] and neural variants such as RankNet [6], LambdaRank [7], and the tree-based LambdaMART [8]. List-wise approaches optimize the whole permutation using losses aligned with ranking metrics; representative methods include ListNet [10], ListMLE [49], and SoftRank [43]. Our paper evaluates point-wise and pair-wise training regimes for transformer encoders in a text-only answer ranking setting.

2.4. Information retrieval. Answer ranking can be viewed as an information retrieval problem: a question acts as a query, and its candidate answers are the documents to be ordered.

2.4.1. *Encoder architectures for text retrieval.* Dense retrieval typically encodes texts with pretrained LMs and compares representations with a similarity function. [55] survey modern dense retrieval and distinguish bi-encoders and cross-encoders. Bi-encoders independently encode query and document, enabling precomputation and scalable retrieval, but they may miss fine-grained interactions. Cross-encoders jointly encode query–document pairs, usually improving accuracy at higher inference cost, and are therefore commonly used for re-ranking.

Between these extremes, late-interaction models aim to retain token-level matching while keeping most computations query-independent. ColBERT [21] encodes query and document separately but scores them via token-level similarity aggregation, providing a practical trade-off between quality and efficiency.

§3. Experimental setup

3.1. Data.

3.1.1. *Dataset description.* We work with the data used in [14] and [13]. Stack Exchange regularly releases data dumps[2], and the authors use a dump dated March 2012. They randomly sample 10 000 questions from this dump and filter only those that have at least 4 answers. They also remove questions that do not have rated answers. The resulting dataset contains 9 721 questions with 53 263 answers. The post creation dates range from August 2008 to March 2012.

[14] and [13] take answer rating as a target variable, which they define as the difference between user upvotes and downvotes. They also normalize the answer ratings by adding the minimum rating to avoid negative values in NDCG computation:

$$r_a = r'_a + r'_{\min}. \tag{1}$$

Here $r'_a = u_a - d_a$ is the difference between the number of upvotes u_a and downvotes d_a received by answer a, and r'_{min} is the minimum difference between upvotes and downvotes observed in the dataset. We follow this approach.

The original dump is a collection of posts, both questions and answers. Both types of posts have metadata features like creation date, owner user id, and comment count. The posts text is represented by HTML markup with `<code>` tags demarcating code snippets. Questions additionally have features like view count, one or multiple tags, and closed date. When transformed for the answer ranking task, the dataset's samples are answer posts with corresponding question posts.

To train and evaluate our models, we create train and test subsets of the dataset. We split them in a ratio of 0.9 to 0.1 which results in 7 776 questions and 42 700 answers in the train subset and 1 945 questions and 10 563 answers in the test subset. The questions in the subsets do not overlap.

[2] `https://data.stackexchange.com/`

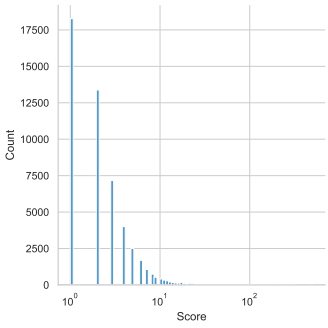


Figure 2. Answer score counts distribution in the dataset (answer scores are given on a logarithmic scale).

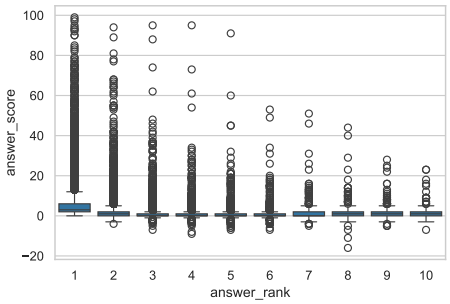


Figure 3. Distribution of answer scores not greater than 100 by answer rank.

Below we provide some statistics of the dataset we work with. In Figure 2, we plot the distribution of answer score counts against a logarithmic scale. It is clear that the distribution is rather skewed. [14] explain it by the fact that few answers attract a large audience to get many votes.

As mentioned above, [14] filter out questions that have less than 4 answers. In the resulting dataset, the median count of answers per question is 5, the 0.95 quantile is 10, and the maximum count is 92.

In Figure 3, we plot answer scores of 100 and less by their respective answer ranks. It can be seen that answers with rank 1 have slightly higher median scores, but the other ranks are less discernible.

We further analyze the length counts of tokenized answer and question texts. To do this, we use the tokenizer of SOBert models. The minimum question text length is 14 tokens, maximum — 11 625 tokens, median is 166 tokens, and the 0.95 quantile — 744 tokens. The minimum answer text length is 2 tokens, maximum — 12 990 tokens, median is 94 tokens, and the 0.95 quantile — 458 tokens. Our analysis shows that the majority of both question and answer posts fall in the range of 1 024 tokens.

3.1.2. *Data preprocessing.* To train our models, we apply preprocessing to the textual data — question text and answer text. We obtain question text by concatenating "question title" and "question body" fields through a

`\n` symbol. We apply the following preprocessing procedure to both question and answer text.

We clean HTML tags out of texts with `beautifulsoup` library. However, we keep `<code>` tags for SOBert models, following their pretraining setup.

To reduce the computational load for bi-encoder and cross-encoder model architectures, we truncate both question and answer posts. We set the limit to 1 024 tokens, following our analysis in Section 3.1.1, to preserve most information in the post texts.

3.2. Evaluation. Since we formulate our problem as a ranking task, we evaluate our results with metrics traditional for L2R tasks. Specifically, we follow [13, 14, 45] and use the NDCG metric [18]. The NDCG at rank k is defined as:

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k} \tag{2}$$

where DCG@k is the Discounted Cumulative Gain at rank k, and IDCG@k is the maximum possible DCG at k, obtained by sorting the documents in order of decreasing true relevance.

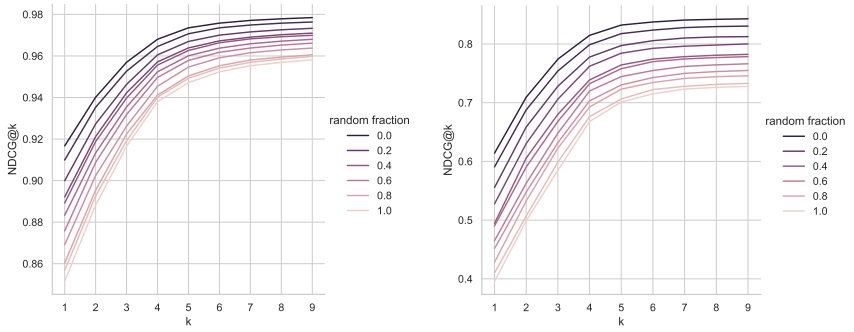
However, we find that the NDCG variant with linear gain function (see Equation 3) used in [13, 14] saturates too quickly and loses the power to distinguish between different models.

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{3}$$

See, for example, [20, 46] for an analysis of different NDCG gain and discount functions properties. We illustrate our finding by comparing the values of NDCG with linear gain and exponential gain (see Equation 4) for the same set of predictions.

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{4}$$

In Figure 4, we plot values of NDCG@k with different gain functions for a set of predictions with a growing fraction of randomness added. It can be seen that values of NDCG@k with linear gain (see Figure 4a) are grouped very tightly and span across hundredths, while NDCG@k with exponential gain (see Figure 4b) provides a wider span in tenths. Therefore, we further use the NDCG variant with an exponential gain function unless explicitly stated otherwise.



(a) NDCG@k with linear gain. (b) NDCG@k with exponential gain.

Figure 4. Values of NDCG@k with different gain functions for a set of predictions with a growing fraction of randomness added.

We also compute the Hit Rate@1 metric for our predictions:

$$\text{Hit Rate@1} = \mathbb{I}\left(\text{rank}^* = 1\right) \tag{5}$$

where rank^* is the rank of the answer with the highest score and $\mathbb{I}(\cdot)$ is the indicator function.

Additionally, we compute the Mean Absolute Error for our predictions in the point-wise approach:

$$\text{MAE} = |\hat{y} - y| \tag{6}$$

where y is the ground-truth answer score and $\hat{y}$ is the predicted answer score.

3.3. Our method.

3.3.1. *Models design.* To train models on the posts' texts, we create dense representations of them. We choose to work with the models introduced in [30] as they are reported to achieve SOTA performance on downstream tasks related to SO at the time of writing this current work. The authors train two models on the forum's data using the Megatron-LM toolkit: SOBertBase and SOBertLarge. Both of them are trained on 27B tokens of SO data with a context size of 2 048 tokens. SOBertBase has 125M parameters and a hidden size of 1 024, while SOBertLarge features 762M parameters and a hidden size of 1 536.

We train two architectures of encoders: bi-encoder and cross-encoder. For the bi-encoder, we create separate representations for question and answer texts. We collect embeddings for each token and then obtain the embedding of the whole text by mean pooling. We further concatenate the embeddings of the question and the answer. We build a scoring model on top of the concatenated representations with the following architecture:

- Linear($2 \cdot$ `hidden_size`, 512)
- ReLU
- Linear(512, 1).

This model outputs scores which are used to rank the answers.

For the cross-encoder, we obtain representations of question-answer pairs concatenated to a single sentence directly, using the model's attention mechanism. The score is then produced by a regression head.

3.3.2. *Training objectives.* We formulate our problem as a ranking task. Consequently, we address it in two common approaches: point-wise and pair-wise.

In the point-wise approach, the model is trained to output regression scores which should approximate the original answer scores. The model is optimized with MSE loss:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{7}$$

where N is the number of answers in the dataset, y_i is the ground-truth score for the i-th answer, and $\hat{y}_i$ is the predicted score for the i-th answer.

In the pair-wise approach, the model is trained to output scores which are used to rank answers to each single question. We optimize our models with a pairwise hinge loss (as implemented in pytorch's `MarginRankingLoss`):

$$\text{MarginRankingLoss} = \frac{1}{N}\sum_{i=1}^{N}\max\left(0, -y_i \cdot (x_{1,i} - x_{2,i}) + \text{margin}\right) \tag{8}$$

where $x_{1,i}$ and $x_{2,i}$ are the predicted scores for the two compared answers in pair i, $y_i \in \{-1, 1\}$ is the ground-truth label indicating the preferred item, margin is the margin hyperparameter, and N is the total number of training pairs. We set the margin equal to 1. During training, the model's input sample is two question-answer pairs where one of the answers should

be ranked higher. During inference, the model makes predictions for single question-answer pairs.

3.3.3. *Training parameters.* In this section, we describe the parameters of our training setup. We train all our DL models for 5 epochs with the AdamW optimizer [26], a learning rate of 2e−5, a linear schedule, and a weight decay of 0.01. The best model is then chosen by NDCG@1. We vary the batch size from 1 to 4 depending on the size of the model and apply gradient accumulation to increase the effective batch size to 4 if needed. We use gradient checkpointing and 8-bit training with large models due to memory constraints.

We further describe the pair building process for the pair-wise approach. By "pair" we denote a pair of answers to a single question where one of the answers has a higher score. We replicate the parameters used for training ranking models in the XGBoost library and implement two pair building strategies: `mean` and `topk`. In the `mean` strategy, N pairs are sampled randomly for each question. In the `topk` strategy, all possible pairs are constructed for each of the K answers with the highest scores within a single question. Both N and K are adjustable parameters.

3.3.4. *Baselines design.* To put the performance of our DL models in context, we also train a number of baseline models. First, we train models based on TF-IDF representations of the texts. For the point-wise approach, we concatenate question and answer representations and train a logistic regression model with default parameters in the sklearn library. We also experiment with removing the question representation, stop words, and `max_features` parameter. For the pair-wise approach, we train ranking models with the XGBoost library which uses the LambdaMART algorithm [8]. We experiment with the number of estimators in the model and the N parameter in the `mean` pair building strategy (see Section 3.3.3 for details).

Second, we train models based on embeddings produced by frozen SOBert backbone models. To obtain representations of the whole posts, we collect embeddings of each token and apply mean pooling. For the point-wise approach, we concatenate the representations of question and answer and produce scores using two methods: cosine similarity and logistic regression. For the pair-wise approach, we train ranking models using the same setup as described for TF-IDF baselines.

Third, we also add constant and random predictions as naive baselines. Constant baseline is a simple function that predicts the same number for

all questions. For the random baseline, we sample random answer scores from the train subset with replacement.

§4. Results

In this section, we describe the results we obtain with our models and baselines on the test subset. We report the metric values in tables and mark the best results within a table for each metric with underscore and the second best — with bold.

In Table 1, we display the results of our baselines. The TF-IDF pair-wise model trained on all possible pairs is the best by a margin in terms of NDCG. The next best results are demonstrated by TF-IDF pair-wise models trained on a limited number of pairs per question. We also note that limiting the number of features allows increasing the performance of the point-wise model.

In Table 2, we report the results of our point-wise models and compare them with the models presented in [13]. None of our models is able to outperform MVIEW; however, all fine-tuned encoders perform better than MVIEW+F0 in all metrics. Among them, the best result is achieved by SOBertLarge cross-encoder. All the other fine-tuned models perform similarly: the parameters count and encoder architecture do not have a significant effect. Regarding the embeddings models, logistic regression trained on top yields clearly better metrics than cosine similarity. We also note that all fine-tuned encoders outperform their counterpart models based on embeddings produced by frozen backbones.

In Table 3, we display the results of our pair-wise models and compare them with the models presented in [13]. None of the models manage to outperform MVIEW; however, all of the fine-tuned encoders perform on par or better than MVIEW+F0. The best NDCG@1 score among our models is achieved by the SOBertLarge cross-encoder trained in pair-wise approach. It can be seen that every fine-tuned encoder outperforms its counterpart model based on embeddings produced by the frozen backbone model. We note that limiting the number of pairs per question to 10 for the bi-encoder results in a slight increase in performance in terms of NDCG. For the models based on embeddings, increasing the number of pairs sampled per question from 1 to 10 raises metric scores. Training embeddings models on all possible pairs per question does not yield better results in our experiments, so we do not report these in the table.

baseline model	NDCG@1	NDCG@2	NDCG@3	NDCG@10	Hit Rate@1	MAE
TF-IDF pair-wise	0.465	0.573	0.655	0.768	**0.340**	-
TF-IDF pair-wise, n_estimators=100, n_pairs=10	0.423	0.527	0.610	0.742	0.337	-
TF-IDF pair-wise, n_estimators=500, n_pairs=10	**0.426**	**0.528**	0.611	**0.743**	0.338	-
TF-IDF point-wise	0.391	0.492	0.585	0.725	0.261	33.263
TF-IDF point-wise, no question text	0.404	0.506	0.592	0.731	0.276	21.630
TF-IDF point-wise, max_features=10000	0.410	0.523	**0.613**	0.741	0.280	11.110
TF-IDF point-wise, max_features=10000, no question text	**0.426**	0.522	**0.613**	**0.743**	0.298	**4.187**
Constant prediction	0.296	0.391	0.490	0.673	0.363	18.441
Random prediction	0.374	0.473	0.562	0.714	0.258	3.669

Table 1. Baseline models results.

model	NDCG@1	NDCG@2	NDCG@3	NDCG@10	Hit Rate@1	MAE
SOBertBase cross-encoder point-wise	0.521	0.622	0.699	0.794	0.404	2.247
SOBertLarge cross-encoder point-wise	**0.537**	**0.634**	**0.710**	**0.801**	**0.423**	2.209
SOBertBase bi-encoder point-wise	0.527	0.625	0.702	0.796	0.409	**2.212**
SOBertLarge bi-encoder point-wise	0.524	0.625	0.703	0.796	0.404	2.217
SOBertBase embeddings, regression	0.488	0.588	0.673	0.780	0.365	3.034
SOBertLarge embeddings, regression	0.479	0.584	0.667	0.775	0.357	3.339
SOBertBase embeddings, similarity	0.436	0.540	0.628	0.753	0.308	17.620
SOBertLarge embeddings, similarity	0.455	0.554	0.638	0.760	0.328	17.540
MVIEW+F0	0.520	0.601	0.676	0.790	0.394	-
MVIEW	0.614	0.709	0.774	0.843	0.510	-

Table 2. Point-wise models results.

In Figure 5, we plot the results of the best models in each combination of encoder architecture and ranking approach and compare them with the models presented in [13]. All the models clearly underperform when compared to MVIEW. The point-wise TF-IDF baseline is an obvious outsider, while the other models form a relatively tight group. Notably, the pair-wise SOBertLarge cross-encoder shows a slight edge over its competitors, particularly for lower ranks (e.g., NDCG@1–3), indicating its effectiveness in accurately identifying the top relevant answers.

§5. Discussion

5.1. Results analysis. In this section, we interpret the results reported above. Analyzing the performance of the baselines, we see that limiting the vocabulary of the TF-IDF representation and even removing the question text results in higher metrics. This implies that TF-IDF fails to model all the dependencies in the data.

model	NDCG@1	NDCG@2	NDCG@3	NDCG@10	Hit Rate@1	MAE
SOBertBase cross-encoder pair-wise	0.523	0.624	0.703	0.795	0.406	-
SOBertLarge cross-encoder pair-wise	**0.540**	**0.639**	0.710	**0.803**	**0.427**	-
SOBertBase bi-encoder pair-wise, n_pairs='all'	0.533	0.634	0.711	0.801	0.417	-
SOBertBase bi-encoder pair-wise, n_pairs=10	0.534	0.638	**0.712**	0.802	0.416	-
SOBertLarge bi-encoder pair-wise	0.519	0.626	0.704	0.796	0.403	-
SOBertBase embeddings pair-wise, n_estimators=500, n_pairs=10	0.512	0.609	0.687	0.790	0.394	-
SOBertBase embeddings pair-wise, n_estimators=100, n_pairs=10	0.508	0.603	0.681	0.786	0.388	-
SOBertBase embeddings pair-wise, n_estimators=100, n_pairs=1	0.494	0.601	0.679	0.783	0.370	-
SOBertLarge embeddings pair-wise, n_estimators=500, n_pairs=10	0.504	0.606	0.685	0.787	0.386	-
SOBertLarge embeddings pair-wise, n_estimators=100, n_pairs=10	0.506	0.604	0.680	0.786	0.389	-
SOBertLarge embeddings pair-wise, n_estimators=100, n_pairs=1	0.489	0.589	0.670	0.778	0.370	-
MVIEW+F0	0.520	0.601	0.676	0.790	0.394	-
MVIEW	0.614	0.709	0.774	0.843	0.510	-

Table 3. Pair-wise models results.

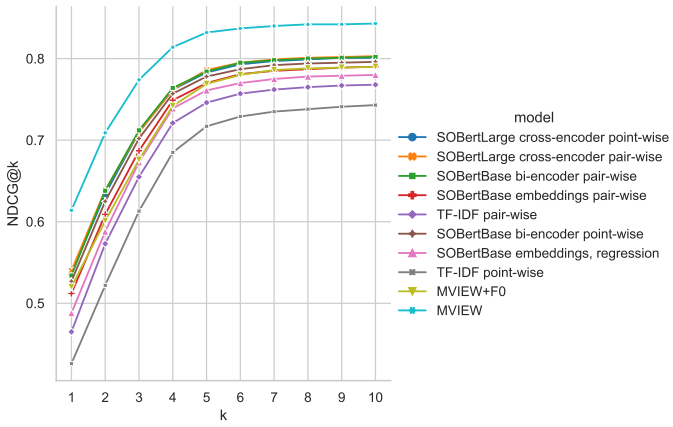


Figure 5. Best models' NDCG@k scores in each combination of encoder architecture and ranking approach.

Comparing the results in Tables 2 and 3, we find that the pair-wise approach almost consistently outperforms the point-wise approach (except

for SOBertLarge bi-encoder). This indicates that selecting a training objective that approximates the target ranking metric (NDCG, in this case) more closely is beneficial.

Another important parameter influencing the performance of the pair-wise approach turns out to be the number of pairs per question. Limiting it to 10 results in a substantial metrics increase for the embeddings models and slightly benefits the bi-encoder model. This suggests that not all pairs with different answer scores may bring useful information for the model. Given the answer score distribution displayed in Figure 2, we observe that the majority of answers have a score of 0 or 1, so a substantial number of pairs created for the pair-wise approach have a minimal score difference of 1. Experimenting with the number of pairs built per question or only creating pairs with score difference greater than a certain threshold may positively impact the performance in this approach. These hypotheses deserve further exploration.

When analyzing the experiment pairs by model size, we see that out of 9 cases SOBertBase performs on par with SOBertLarge in 2, outperforms it in 4 and the opposite happens in 3. [30] observe a similar trend when larger models perform equally with smaller ones and even worse. They explain it by the fact that training a classifier model with more parameters on top of an encoder with a larger hidden size provides more opportunities for overfitting on small datasets.

When comparing the performance of encoder architectures, we observe no clear winner. However, we note that the cross-encoder outperforms the bi-encoder only for the large model size. This may suggest that a greater parameter count may be important to benefit from the attention mechanism.

When analyzing the training setup, we observe that fine-tuned encoders consistently outperform models based on frozen embeddings by a substantial margin. This aligns with the common practice of unfreezing the entire LM during fine-tuning to improve performance. Our best-performing model across all metrics is the SOBertLarge cross-encoder trained using the pair-wise approach. This supports the general intuition that a larger model — benefiting from the attention mechanism and optimized with a ranking-specific objective — achieves superior results.

To compare our results with other models presented in [13, 14], we perform further analysis. As these works use the NDCG variant with a linear gain function, we calculate it in addition to the metrics reported above. In

Model	NDCG@				Hit Rate
	1	2	3	10	@1
SOBertLarge cross-encoder pair-wise	0.893	**0.923**	**0.943**	**0.972**	**0.427**
MVIEW+F0	**0.894**	0.918	0.939	0.971	0.394
MVIEW	0.916	0.940	0.956	0.978	0.510

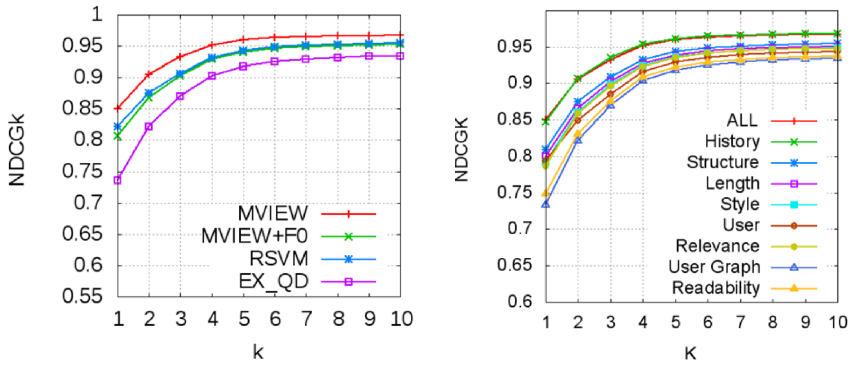
Table 4. Our best-performing model's results compared to MVIEW and MVIEW+F0 in NDCG@k with a *linear* gain function.

Table 4, we display values of NDCG with a linear gain function achieved by our best-performing model, MVIEW, and MVIEW+F0. MVIEW still yields the highest scores, while SOBert performs on par or slightly better than MVIEW+F0, although the difference is much less evident than with NDCG with an exponential gain function.

In Figure 6, we provide the NDCG@k values of models presented in [13] as reported in their work. Note that the exact values differ slightly from those we calculate ourselves and display in Table 4. We attribute this to the details of the NDCG metric implementation used by [13]. Since we have already shown that our best-performing model achieves the same or better scores as MVIEW+F0, we can count that they are represented by the same line in Figure 6a. Given that, we analyze Figure 6b and find that SOBertLarge cross-encoder trained in a pair-wise approach outperforms all individual views except for History and Structure.

In Figure 7, we give the results of the random forests model presented in [14] as reported by the authors. To compare with previous work, [13] use a RankingSVM model [17] instead of RF but state that "the RF results were qualitatively similar to the ones we obtained for SVM-RANK". Given that, we can count that our best model's performance is given by the line for the RF model with all features in Figure 7. Here, its NDCG@1 score aligns with the score of 0.894 we report for MVIEW+F0 in Table 4. Consequently, we can claim that our best text-based model outperforms RF models for textual-only features presented in [14].

To summarize, our fine-tuned LMs boost the performance of models based on frozen embeddings but underperform compared to MVIEW. It is important to note, however, that our best model achieves higher scores than any of the individual views used in MVIEW except for History and



(a) NDCG@k scores of models and baselines presented in [13].

(b) NDCG@k scores of MVIEW compared to individual views.

Figure 6. Results of models from [13] in NDCG@k with a *linear* gain function (images from [13]).

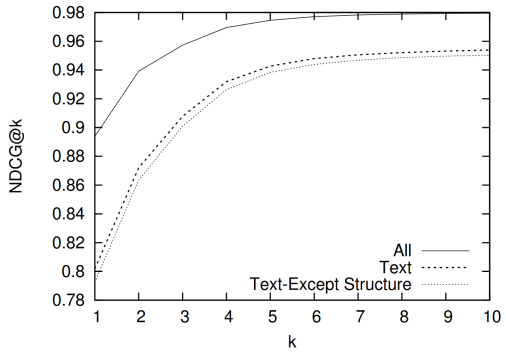


Figure 7. Results of the Random Forests model [14] for all and textual-only features in NDCG@k with a *linear* gain function (image from [14]).

Structure. The Edit History view includes features like the number of comments to the post or the number of times it has been edited. These features go beyond the text-only problem statement used in this work. In addition, such features may provide oracle information because, for example, a high

number of comments marks an answer which has attracted a large audience and consequently potential user votes. The Structure view includes features like section count or the number of links provided in the answer. Although [13] do not report the contribution of each individual view to the overall performance of MVIEW, the Structure view by itself achieves results comparable to our best-performing encoder model. This reinforces the question of to what extent modern encoders capture "visual" parameters of the text like its overall length or the number of sentences. Explicit inclusion of such features into the model deserves additional exploration. Overall, this comparison highlights the importance of metadata for predicting quality answers stated, for example, in [51].

Continuing the comparison with MVIEW, we highlight several advantages of our approach. First, our method does not rely on any hand-crafted features and can be applied off-the-shelf to domains beyond programming, which may require additional feature engineering. Second, the text-only nature of our problem statement reduces potential biases in the ranking model. MVIEW includes views like User which assumes that answers from users with high reputation are more likely to appear higher in the ranking. This may disadvantage new users and does not account for situations where an answer is submitted under another user's account. Our model is not subject to these problems.

5.2. Correlation with question tags. In this section, we analyze the correlation of our best model's performance with question tags. In Figure 8, we plot the distribution of NDCG@1 scores per question depending on the 10 most common question tags. Here, most distributions are bimodal and have the same median of around 0.5. The only tag with a substantially higher 0.25 quantile is `html`. It also has a larger number of questions concentrated around the median score of 0.5. It indicates that the `html` tag has more sets of answers that are ranked with mediocre quality in contrast to other tags which have two groups of answers ranked with high and low quality.

However, when we compare the performance of our model for questions having each of the 10 most common tags and all the remaining ones, we find no statistically significant difference in the scores. For details, see Table 7 in Appendix A.

When we extend our analysis to all the question tags present in the dataset, we find 26 tags which cause a statistically significant difference in the scores. However, 19 of them are represented by less than 10 questions

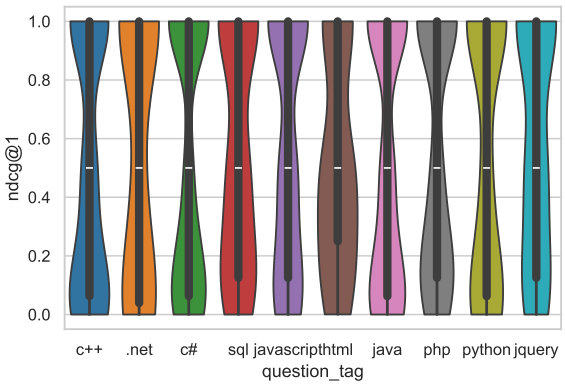


Figure 8. Distribution of NDCG@1 scores of our best-performing model depending on the 10 most common question tags in the dataset.

question tag	**mean NDCG@1**	
	with tag	**without tag**
asp.net	0.622	0.537
coding-style	0.213	0.542
django	0.221	0.541
linux	0.750	0.536
multithreading	0.353	0.542
regex	0.691	0.537
unit-testing	0.321	0.542

Table 5. NDCG@1 scores of our best-performing model for questions with and without tags represented by 10 or more questions in the test subset. The differences are statistically significant with the Mann-Whitney U test.

in the test subsets which prevents us from drawing reliable conclusions. In Table 5, we report NDCG@1 scores for questions with and without each of the 7 remaining tags. The differences for each pair are statistically significant with the Mann-Whitney U test.

Model	NDCG@				Hit Rate	MAE
	1	2	3	10	@1	
SOBertBase	0.521	0.622	0.699	0.794	**0.404**	**2.247**
Longformer-base	0.484	0.583	0.667	0.776	0.407	2.345
ModernBERT-base	**0.516**	**0.614**	**0.694**	**0.792**	0.401	2.220

Table 6. Comparison with general-purpose LMs (all models are point-wise cross-encoders).

5.3. Comparison with general-purpose LMs. In Table 6, we compare our SOBertBase point-wise cross-encoder to general-purpose LMs of similar sizes. We choose two long-context models: Longformer-base [5] (148M parameters) as a model pretrained on natural language only and ModernBERT-base [47] (150M parameters) as a model pretrained on a mixture of natural language and code. Longformer supports sequences up to 4 096 tokens in length, while ModernBERT works with sequences up to 8 192 tokens. We train these two models in the same setup as SOBert and truncate both question and answer text to 1 024 tokens.

Among the three models, SOBertBase shows the best results in terms of NDCG. Additionally, ModernBERT clearly outperforms Longformer in terms of NDCG, which is likely because of the code data used in its pretraining. This comparison supports the claim that pretraining on in-domain data increases the performance in downstream tasks stated, for example, in [30] itself.

5.4. Possible integrations. We see several possibilities to integrate models trained with our method to the Stack Overflow platform. First, one of the main applications of these models could be to produce preliminary rankings for sets of answers that have not yet received enough user votes. This could reduce the time users need to find relevant information for new questions, which could positively impact the platform's popularity.

Second, the models could be used to rerank answers which have been posted to questions which already have an accepted answer. Such questions attract a smaller audience and consequently have a lower chance to receive user votes. If they are scored by the model substantially higher than the accepted answer, they can be automatically ranked higher.

Finally, the models could be integrated into a plugin on the Stack Overflow website that provides users with instant feedback while they type answers to questions. The model's ranking score could be converted, for

example, to categorical grades and shown to the user as a measure of their answer's quality at the moment. Here, our text-only problem statement is crucial because the model's score does not depend on any features, including the user's profile, except for the text of the answer itself.

§6. Conclusion and future work

In this work, we address the task of assessing Stack Overflow answers quality. We formulate it as a ranking problem and aim to rank the answers according to their user votes. We work with an existing dataset presented in previous work, which solved an analogous task.

We use a text-only task formulation and train our models solely on question and answer texts. We train a line of ranking models using two ranking approaches – point-wise and pair-wise – and two encoder architectures – bi-encoder and cross-encoder. We evaluate our models with metrics traditional for learning to rank tasks: NDCG and Hit Rate.

We compare the fine-tuned models to models based on frozen embeddings, models presented in previous studies, and naive baselines. None of our models outperforms a meta-learning pair-wise model proposed in [13] which sees the answers through different "views" and uses features including metadata like edit history or user profiles. However, our best-performing model achieves better scores than a random forest model based on hand-crafted textual features from [14]. All our fine-tuned models consistently outperform their counterparts based on frozen embeddings.

We further investigate the performance of our models. We analyze the behaviour of our best model depending on the question tags. We find a set of question tags which cause statistically significant differences in the performance.

We compare our best model to models based on two general purpose LMs pretrained on natural language and a mixture of natural language and code. All three models are of comparable sizes. We find that our model achieves the highest results among them.

We propose several ways to integrate models trained with our approach into the Stack Overflow platform. First, they could be used to preliminarily rank new answers with an insufficient number of votes. Second, they could be applied to adjust the ranking of answers which are posted after a certain answer is accepted and have not attracted due user attention. Finally, a website plugin could be created which provides the user with real-time feedback regarding the quality of the answer they are writing. The ranking

scores from our models could be transformed and shown to the user as a categorical grade of answer quality.

Several directions for future research open up on the basis of this work. Given the high performance of the individual Structure view in MVIEW, we conjecture it is promising to explicitly include structural features like the length of the text or the number of sentences into the trained models.

It is of interest to experiment with the training objectives. First, models can be optimized with similarity loss for question and answer embeddings, following [1]. Second, one could compare the ranking obtained with our approach to the approach when the answers are ranked according to their possibility of being accepted.

Another research question could be the extent to which code snippets contribute to the answers' quality. Is it possible to achieve high-quality ranking using only the code or only textual descriptions in the answers? Code snippets could be encoded separately with models pretrained on extensive amounts of code data and then combined with the representations of the rest of the answer. Code quality can be assessed with external models and added as a feature to answer representations. [28] is one example of a study that addresses the quality of code snippets in SO posts.

Based on the results obtained in this study, it is beneficial to limit and further experiment with the number of pairs created per question for the pair-wise approach. Since we find that not all answer pairs may bring useful information for the model, pairs could be created only from answers with score differences greater than a certain threshold.

A natural extension of the bi-encoder setup would be to apply the ColBERT approach. It is shown to outperform traditional bi-encoder models based on a similarity function [21].

Finally, it is worth exploring list-wise approaches to ranking. They tend to outperform point-wise and pair-wise approaches by more closely approximating evaluation functions like NDCG [43].

Appendix A. Analysis of performance-affecting question tags

This appendix presents a detailed analysis of the question tags that significantly affect ranking performance. Table 7 reports the NDCG@1 scores of our best-performing model for questions with and without each of the ten most common tags. Tables 8, 9, and 10 summarise the distinguishing

question tag	mean NDCG@1	
	with tag	without tag
.net	0.523	0.541
c#	0.533	0.541
c++	0.497	0.544
html	0.513	0.541
java	0.527	0.542
javascript	0.563	0.538
jquery	0.577	0.538
php	0.562	0.538
python	0.543	0.540
sql	0.552	0.539

Table 7. NDCG@1 scores of our best-performing model for questions with and without each of the 10 most common question tags in the dataset. The differences are not statistically significant with the Mann-Whitney U test.

tag	NDCG@1		Length (tokens)		# code snippets		Code pct. (characters)	
	tag	no tag	tag	no tag	tag	no tag	tag	no tag
asp.net	0.622	0.537	312.770	266.130	1.405	1.445	16.678	20.654
coding-style	0.213	0.542	155.454	268.544	1.090	1.446	21.350	20.498
django	0.221	0.541	217.200	268.166	1.400	1.444	16.377	20.524
linux	0.750	0.536	**147.156**	269.924	**0.625**	1.457	**7.161**	20.726
multithreading	0.353	0.542	539.300	265.085	1.200	1.446	17.965	20.529
regex	0.691	0.537	244.609	268.406	**2.512**	1.421	**23.429**	20.440
unit-testing	0.321	0.542	279.470	267.802	**0.529**	1.452	**7.822**	20.615

Table 8. Features of questions with performance-affecting tags related to question text. Differences significant with the Mann-Whitney U test are marked with bold.

features of questions that carry performance-affecting tags, grouped respectively by question-text, answer-score, and answer-text characteristics.

tag	NDCG@1		Scores range		1st score gap pct.		Scores entropy	
	tag	no tag	tag	no tag	tag	no tag	tag	no tag
asp.net	0.622	0.537	**3.229**	6.897	**0.090**	0.177	0.638	0.481
coding-style	0.213	0.542	**9.363**	6.743	0.204	0.173	0.316	0.488
django	0.221	0.541	**11.800**	6.732	0.312	0.173	**0.285**	0.488
linux	0.750	0.536	5.718	6.775	0.198	0.173	0.580	0.486
multithreading	0.353	0.542	9.300	6.731	0.195	0.173	0.393	0.488
regex	0.691	0.537	5.170	6.792	0.124	0.174	0.486	0.487
unit-testing	0.321	0.542	11.117	6.719	0.215	0.173	0.358	0.488

Table 9. Features of questions with performance-affecting tags related to answer scores. Differences significant with the Mann-Whitney U test are marked with bold.

tag	NDCG@1		Length (tokens)		# code snippets		Code pct. (characters)	
	tag	no tag	tag	no tag	tag	no tag	tag	no tag
asp.net	0.622	0.537	**159.597**	156.761	**0.768**	1.323	**16.200**	21.369
coding-style	0.213	0.542	106.000	157.333	**0.690**	1.309	**9.017**	21.299
django	0.221	0.541	181.180	156.721	0.852	1.306	21.876	21.182
linux	0.750	0.536	**122.025**	157.387	1.197	1.305	16.151	21.262
multithreading	0.353	0.542	163.837	156.780	**0.748**	1.310	**10.636**	21.310
regex	0.691	0.537	167.167	156.648	**2.181**	1.285	**32.710**	20.946
unit-testing	0.321	0.542	171.488	156.672	**0.333**	1.316	**5.750**	21.386

Table 10. Features of questions with performance-affecting tags related to answer text. Differences significant with the Mann-Whitney U test are marked with bold.

References

1. I. Abdelaziz, J. Dolby, J. McCusker, and K. Srinivas, *Can Machines Read Coding Manuals Yet? – A Benchmark for Building Better Language Models for Code Understanding*, ArXiv preprint arXiv:2109.07452 (2021).
2. M. Ahmed, H. U. Khan, M. A. Khan, U. Tariq, and S. Kadry, *Context-aware Answer Selection in Community Question Answering Exploiting Spatial Temporal Bidirectional Long Short-Term Memory*, ACM Trans. Asian Low-Resour. Lang. Inf. Process. (2023).
3. L. Amancio, C. F. Dorneles, and D. H. Dalip, *Recency and quality-based ranking question in CQAs: A Stack Overflow case study*, Information Processing & Management **58** (2021), 102552.

4. I. Annamoradnejad, J. Habibi, and M. Fazli, *Multi-view approach to suggest moderation actions in community question answering sites*, Information Sciences **600** (2022), 144–154.
5. I. Beltagy, M. E. Peters, and A. Cohan, *Longformer: The Long-Document Transformer*, ArXiv preprint arXiv:2004.05150 (2020).
6. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, *Learning to rank using gradient descent*, in: Proceedings of the 22nd international conference on Machine learning, 2005, pp. 89–96.
7. C. Burges, R. Ragno, and Q. Le, *Learning to rank with nonsmooth cost functions*, Advances in neural information processing systems **19** (2006).
8. C. J. Burges, *From ranknet to lambdarank to lambdamart: An overview*, Learning **11** (2010), 81.
9. F. Calefato, F. Lanubile, M. C. Marasciulo, and N. Novielli, *Mining Successful Answers in Stack Overflow*, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 430–433.
10. Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, *Learning to rank: from pairwise approach to listwise approach*, in: Proceedings of the 24th international conference on Machine learning, 2007, pp. 129–136.
11. B.-C. Chen, A. Dasgupta, X. Wang, and J. Yang, *Vote calibration in community question-answering systems*, in: Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval, 2012, pp. 781–790.
12. K. Crammer and Y. Singer, *Pranking with ranking*, Advances in neural information processing systems **14** (2001).
13. D. H. Dalip, M. A. Gonçalves, M. Cristo, and P. Calado, *A general multiview framework for assessing the quality of collaboratively created content on web 2.0*, J. Assoc. Inf. Sci. Technol. **68** (2017), 286–308.
14. D. H. Dalip, M. A. Gonçalves, M. Cristo, and P. Calado, *Exploiting user feedback to learn to rank answers in q&a forums: a case study with stack overflow*, in: Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval, 2013, pp. 543–552.
15. X. Du, G. Zheng, K. Wang, J. Feng, W. Deng, M. Liu, B. Chen, X. Peng, T. Ma, and Y. Lou, *Vul-RAG: Enhancing LLM-based vulnerability detection via knowledge-level RAG*, ArXiv preprint arXiv:2406.11147 (2024).
16. J. He, X. Zhou, B. Xu, T. Zhang, K. Kim, Z. Yang, F. Thung, I. C. Irsan, and D. Lo, *Representation Learning for Stack Overflow Posts: How Far Are We?*, ACM Trans. Softw. Eng. Methodol. **33** (2024), 69:1–69:24.
17. T. Joachims, *Optimizing search engines using clickthrough data*, in: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 2002, pp. 133–142.
18. K. Järvelin and J. Kekäläinen, *Cumulated gain-based evaluation of IR techniques*, ACM Transactions on Information Systems **20** (2002), 422–446.
19. S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang, *Is Stack Overflow Obsolete? An Empirical Study of the Characteristics of ChatGPT Answers to Stack Overflow Questions*, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–17.

20. E. Kanoulas and J. A. Aslam, *Empirical justification of the gain and discount function for nDCG*, in: Proceedings of the 18th ACM conference on Information and knowledge management, 2009, pp. 611–620.
21. O. Khattab and M. Zaharia, *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*, ArXiv preprint arXiv:2004.12832 (2020).
22. J. H. Klemmer, S. A. Horstmann, N. Patnaik, C. Ludden, C. B. Jr, C. Powers, F. Massacci, A. Rahman, D. Votipka, H. R. Lipford, A. Rashid, A. Naiakshina, and S. Fahl, *Using AI Assistants in Software Development: A Qualitative Study on Security Practices and Concerns*, in: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, 2024, pp. 2726–2740.
23. D. Kreitmeir and P. A. Raschky, *The Heterogeneous Productivity Effects of Generative AI*, ArXiv preprint arXiv:2403.01964 (2024).
24. M. T. R. Laskar, J. X. Huang, and E. Hoque, *Contextualized Embeddings based Transformer Encoder for Sentence Similarity Modeling in Answer Selection Task*, in: Proceedings of the Twelfth Language Resources and Evaluation Conference, 2020, pp. 5505–5514.
25. W. Ledbetter and J. Springer, *Answer Comments As Reviews: Predicting Acceptance By Measuring Valence On Stack Exchange*, in: 2022 IEEE International Conference on Big Data (Big Data), 2022, pp. 4782–4791.
26. I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, ArXiv preprint arXiv:1711.05101 (2019).
27. Y. Mai, Z. Gao, H. Wang, T. Bi, X. Hu, X. Xia, and J. Sun, *Towards Better Answers: Automated Stack Overflow Post Updating*, ArXiv preprint arXiv:2408.09095 (2024).
28. S. Meldrum, S. A. Licorish, C. A. Owen, and B. T. R. Savarimuthu, *Understanding stack overflow code quality: A recommendation of caution*, Science of Computer Programming **199** (2020), 102516.
29. S. Mondal, G. Uddin, and C. Roy, *Automatic Prediction of Rejected Edits in Stack Overflow*, ArXiv preprint arXiv:2210.03281 (2022).
30. M. Mukherjee and V. J. Hellendoorn, *"Medium" LMs of Code in the Era of LLMs: Lessons From StackOverflow*, ArXiv preprint arXiv:2306.03268 (2024).
31. M. Mukherjee and V. J. Hellendoorn, *SOSecure: Safer Code Generation with RAG and StackOverflow Discussions*, ArXiv preprint arXiv:2503.13654 (2025).
32. M. Neshati, *On early detection of high voted Q&A on Stack Overflow*, Information Processing & Management **53** (2017), 780–798.
33. O. P. Omondiagbe, S. A. Licorish, and S. G. Macdonell, *Evaluating Simple and Complex Models' Performance When Predicting Accepted Answers on Stack Overflow*, 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (2022), 29–38.
34. O. P. Omondiagbe, S. A. Licorish, and S. G. MacDonell, *Features that Predict the Acceptability of Java and JavaScript Answers on Stack Overflow*, ArXiv preprint arXiv:2101.02830 (2023).
35. G. Orosz, *Are LLMs making StackOverflow irrelevant?*, 2025.
36. G. Orosz, *The Pulse #134: Stack overflow is almost dead*, 2025.

37. P. K. Roy, S. Saumya, J. P. Singh, S. Banerjee, and A. Gutub, *Analysis of community question-answering issues via machine learning and deep learning: State-of-the-art review*, CAAI Transactions on Intelligence Technology **8** (2023), 95–117.
38. B. Sen, N. Gopal, and X. Xue, *Support-BERT: Predicting Quality of Question-Answer Pairs in MSDN using Deep Bidirectional Transformer*, ArXiv preprint arXiv:2005.08294 (2020).
39. A. Seredko and T. Hillman, *What Does a Downvote Do? Performing Complementary and Competing Knowledge Practices on an Online Platform*, Proc. ACM Hum.-Comput. Interact. **8** (2024), 201:1–201:28.
40. S. Sun, Y. Wang, J. Cheng, Z. Xiao, D. Zheng, and X. Hao, *Improving the Performance of Answers Ranking in Q&A Communities: A Long-Text Matching Technique and Pre-Trained Model*, IEEE Access **13** (2025), 4188–4200.
41. J. Tabassum, M. Maddela, W. Xu, and A. Ritter, *Code and Named Entity Recognition in StackOverflow*, ArXiv preprint arXiv:2005.01634 (2020).
42. M. Tavakoli, M. Izadi, and A. Heydarnoori, *Improving Quality of a Post's Set of Answers in Stack Overflow*, in: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020, pp. 504–512.
43. M. Taylor, J. Guiver, S. Robertson, and T. Minka, *SoftRank: optimizing non-smooth rank metrics*, in: Proceedings of the 2008 International Conference on Web Search and Data Mining, 2008, pp. 77–86.
44. Q. Tian, P. Zhang, and B. Li, *Towards Predicting the Best Answers in Community-based Question-Answering Services*, Proceedings of the International AAAI Conference on Web and Social Media **7** (2013), 725–728.
45. L. Valentin, *Answer ranking in Community Question Answering: a deep learning approach*, ArXiv preprint arXiv:2212.01218 (2022).
46. Y. Wang, L. Wang, Y. Li, D. He, T.-Y. Liu, and W. Chen, *A Theoretical Analysis of NDCG Type Ranking Measures*, ArXiv preprint arXiv:1304.6480 (2013).
47. B. Warner, A. Chaffin, B. Clavié, O. Weller, O. Hallström, S. Taghadouini, A. Gallagher, R. Biswas, F. Ladhak, T. Aarsen, N. Cooper, G. Adams, J. Howard, and I. Poli, *Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference*, ArXiv preprint arXiv:2412.13663 (2024).
48. S. Wills, S. T. S. Poon, A. Salili-James, and B. Scott, *The use of generative AI for coding in academia*, Methods in Ecology and Evolution **15** (2024), 2189–2191.
49. F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, *Listwise approach to learning to rank: theory and algorithm*, in: Proceedings of the 25th international conference on Machine learning, 2008, pp. 1192–1199.
50. B. Xu, T. Hoang, A. Sharma, C. Yang, X. Xia, and D. Lo, *Post2Vec: Learning Distributed Representations of Stack Overflow Posts*, IEEE Transactions on Software Engineering **48** (2022), 3423–3441.
51. S. Xu, A. Bennett, D. Hoogeveen, J. H. Lau, and T. Baldwin, *Preferred Answer Selection in Stack Overflow: Better Text Representations ... and Metadata, Metadata, Metadata*, in: Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text, 2018, pp. 137–147.

52. Y. Yang, W. He, C. Gao, Z. Xu, X. Xia, and C. Liu, *TopicAns: Topic-informed Architecture for Answer Recommendation on Technical Q&A Site*, ACM Trans. Softw. Eng. Methodol. **33** (2023), 20:1–20:25.
53. M. Yazdaninia, D. Lo, and A. Sami, *Characterization and Prediction of Questions without Accepted Answers on Stack Overflow*, in: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), 2021, pp. 59–70.
54. H. Zhang, S. Wang, T.-H. Chen, and A. E. Hassan, *Reading answers on stack overflow: Not enough!*, IEEE Transactions on Software Engineering **47** (2021), 2520–2533.
55. W. X. Zhao, J. Liu, R. Ren, and J.-R. Wen, *Dense Text Retrieval based on Pretrained Language Models: A Survey*, ArXiv preprint arXiv:2211.14876 (2022).

AI Talent Hub,
ITMO University, Saint Petersburg, Russia;
Slavic-Greek-Latin Academy,
Moscow, Russia
E-mail: eugene200020@gmail.com

ITMO University,
St. Petersburg, Russia
E-mail: valentin.malykh@phystech.edu

Поступило 4 февраля 2026 г.