






Automated Question Title Reformulation by Mining Modification Logs From Stack Overflow

Ke Liu , Xiang Chen , *Member, IEEE*, Chunyang Chen , Xiaofei Xie , and Zhanqi Cui 

Abstract—In Stack Overflow, developers may not clarify and summarize the critical problems in the question titles due to a lack of domain knowledge or poor writing skills. Previous studies mainly focused on automatically generating the question titles by analyzing the posts’ problem descriptions and code snippets. In this study, we aim to improve title quality from the perspective of question title reformulation and propose a novel approach QETRA motivated by the findings of our formative study. Specifically, by mining modification logs from Stack Overflow, we first extract title reformulation pairs containing the original title and the reformulated title. Then we resort to multi-task learning by formalizing title reformulation for each programming language as separate but related tasks. Later we adopt a pre-trained model T5 to automatically learn the title reformulation patterns. Automated evaluation and human study both show the competitiveness of QETRA after compared with six state-of-the-art baselines. Moreover, our ablation study results also confirm that our studied question title reformulation task is more practical than the direct question title generation task for generating high-quality titles. Finally, we develop a browser plugin based on QETRA to facilitate the developers to perform title reformulation. Our study provides a new perspective for studying the quality of post titles and can further generate high-quality titles.

Index Terms—Stack Overflow mining, question post quality assurance, question title reformulation, modification logs, deep learning.

I. INTRODUCTION

STACK Overflow is widely used for developers to seek solutions for programming-related problems. While the number of question posts in Stack Overflow has been growing rapidly, researchers [1], [2] found that the quality of a significant number of questions is not satisfactory. These low-quality question posts cannot attract timely attention from other developers in

Stack Overflow. Moreover, they can hinder the generation and sharing of programming knowledge [3], [4]. One of the essential reasons for the low quality of question posts is that developers cannot clarify and summarize the critical problems in the question titles [5], [6], [7] due to the lack of domain knowledge or good writing skills. This can also have a negative impact on some tasks related to Stack Overflow (such as post retrieval [8], [9], chatbot development [10], API method recommendation [11], [12]). Therefore, the automatic question title generation has become a valuable research problem in mining Stack Overflow.

In previous studies, researchers mainly focused on generating the question titles by analyzing the posts’ contents (such as problem descriptions and code snippets). For example, Gao et al. [4] were the first to automatically generate titles by analyzing the code snippets in the post body. They proposed a data-driven sequence-to-sequence approach Code2Que, which used an attention mechanism to better select content from the code snippet, a copy mechanism to solve the out-of-vocabulary problem, and a coverage mechanism to eliminate the word repetition problem. Then, we [13] further leveraged both the code snippets and the problem descriptions. We modeled this task as a multi-task learning problem and proposed a Transformer-based approach SOTitle.

One of the main reasons for the popularity of Stack Overflow is its ability to maintain high-quality posts on its platform. The community has implemented a set of mechanisms to ensure that the content posted on the platform is accurate, useful, and relevant. For example, after a question post is posted, the owner of the question post and other users can edit the content of the question post¹. Previous studies [14], [15] have also shown that editing question posts can help to improve the quality of question posts. Therefore, we can improve the title quality by learning title reformulation patterns by gathering the title modification information of developers in modification logs provided by Stack Overflow. Then given an unsatisfactory original question title, we can use the learned title reformulation knowledge to recommend reformulated titles with higher quality for the post editors. Notice previous studies [4], [13] mainly generate the title from scratch, while our investigated question title reformulation aims to polish the original crafted title by mining modification log. Therefore, our study can provide a new perspective to improve post title quality.

In our study, we first perform a formative study to investigate how developers reformulate the question titles by mining

This work was supported in part by the National Natural Science Foundation of China under Grants 61702041 and 61202006 and Jiangsu Provincial Frontier Leading Technology Fundamental Research Project under Grant BK20202001. Recommended for acceptance by T. Menzies. (Ke Liu and Xiang Chen are co-first authors.) (Corresponding author: Xiang Chen.)

Ke Liu and Xiang Chen are with the School of Information Science and Technology, Nantong University, Nantong 226019, China (e-mail: aurora.ke.liu@outlook.com; xchen@ntu.edu.cn).

Chunyang Chen is with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia (e-mail: chunyang.chen@monash.edu).

Xiaofei Xie is with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: xfxie@ntu.edu.sg).

Zhanqi Cui is with the Computer School, Beijing Information Science and Technology University, Beijing 100101, China (e-mail: czq@bistu.edu.cn).

Digital Object Identifier 10.1109/TSE.2023.3292399

¹<https://stackoverflow.com/help/editing>

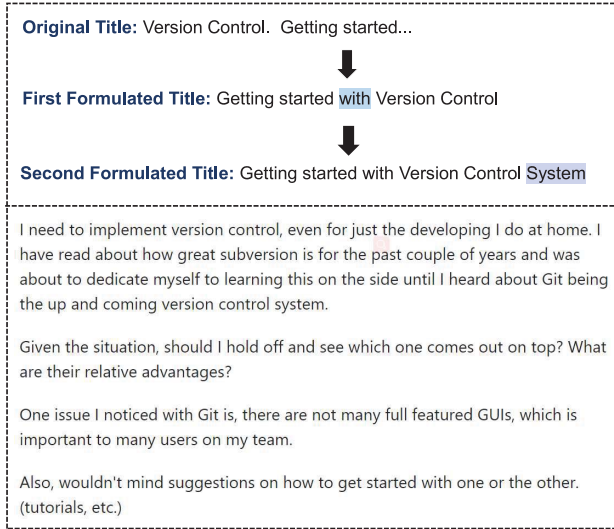


Fig. 1. The process of modifying the title of a question post in Stack Overflow.

modification logs. Based on the modification logs between July 2008 and March 2022, we extract 152,259 title reformulation pairs from 122,528 title reformulation threads. Fig. 1 shows a title reformulation thread. In this thread, the original title has been reformulated two times (The modified part is highlighted with shadows) and the post body is shown in the bottom half of this figure. After analyzing these title reformulation pairs, we find that question title reformulation has certain common patterns (detailed analysis is in Section III-C). For example, the developers may correct spelling errors, such as revising “*Undestanding*” to “*Understanding*.” The developers may simplify and refine the title into the most common expression ways, such as revising “*C# how do I write in the code this char—`*” to “*C# how do I write the escape char—` to code.*” The developers may add constraints (e.g., programming languages or platforms) in the title, such as revising “*how to convert a json to a php object?*” to “*how to convert a json to a php object in symfony2?*.” The developers may delete specific information in error messages or code snippets, such as revising “*MongoDb throws java.lang.IllegalArgumentException: Invalid BSON field name*” to “*MongoDb upsert exception invalid BSON field.*”

Since the question title reformulation task is tedious for developers, we propose a novel approach QETRA (QuEstion Title Reformulation) based on deep learning to automatically reformulate the original question title. Specifically, by mining modification logs from Stack Overflow, we first extract title reformulation pairs containing the original title and the reformulated title from the identified title reformulation threads. Then, based on the findings in our informative study (such as a large number of software-specific concepts exist in question posts for different programming languages, but the other contents in the post titles for different programming languages have a certain similarity), we formalize the question title reformulation for different programming languages as separate but related tasks and resort to multi-task learning [16], which can allow related tasks to improve each other’s performance

by using shared and complementary information. Finally, we adopt a pre-trained model T5 [17] to automatically learn the title reformulation patterns from the extracted title reformulation pairs. For the application phase, given the original title and the content of the question post body, the trained model can recommend candidate reformulated titles. We evaluate the quality of the reformulated titles of our approach with large-scale archival manual reformulation results in terms of both automatic evaluation and human study evaluation, respectively. Automatic evaluation results show that QETRA can generate higher-quality reformulated titles than six state-of-the-art baselines. Taking Python programming language as an example, in terms of metrics EM@1 and GLEU, our approach achieves 228.69% and 51.59% improvement compared with the baseline CodeBERT [18] and achieves 150.63% and 13.11% improvement compared with the baseline LanguageTool. The human study results also show that the quality of the reformulated titles by QETRA is higher than the baselines CodeBERT and LanguageTool. Finally, we find considering the original title and the post body information can achieve better performance than only considering the post body information in our ablation studies. Our empirical results show that reformulating on the basis of the original titles can better guarantee the quality of the titles.

The contributions of our study can be summarized as follows.

- We are the first to study the problem of automated question title reformulation by mining modification logs from Stack Overflow. Our study can provide a new direction for improving question title quality.
- We construct a high-quality large-scale dataset by mining modification logs of high-quality problem posts for six popular programming languages in Stack Overflow. This dataset includes 152,259 title reformulation pairs from 122,528 title reformulation threads.
- We perform a formative study to investigate how developers reformulate the question titles. Inspired by the findings of our formative study, we propose a novel approach QETRA based on deep learning to automatically reformulate the original question title, which is designed based on the pre-trained model T5 and multi-task learning.
- We evaluate the quality of the reformulated titles of QETRA by comparing state-of-the-art baselines based on automatic evaluation and human study evaluation. Moreover, to facilitate the formulation of the original title for developers, we develop a browser plugin based on QETRA.
- To facilitate other researchers to follow and advance our study, we have released the scripts, the dataset, and our developed plugin on our project homepage.²

The remaining part of this paper is organized as follows: Section II describes our process of collecting title reformulation pairs from modification logs. Section III illustrates the findings of our formative study. Section IV shows the details of our proposed approach QETRA. Section V introduces the experimental setup of our study. Section VI presents the empirical results for each research question. Section VII discusses the generalization

²<https://github.com/KeLiu97/QETRA>

TABLE I
ALL THE RECORDS FOR THE TITLE REFORMULATION (SHOWN IN FIG. 1) IN THE DATA DUMP *PostHistory*

Id	PostId	PostHistoryTypeId	CreationDate	UserId	Text
16757	2658	1	2008-08-05T18:29:14.283	406	Version Control. Getting started...
19367804	2658	4	2011-12-22T22:13:08.093	845306	Getting started with Version Control
264747877	2658	4	2022-02-23T12:27:23.230	5452687	Getting started with Version Control System

TABLE II
LENGTH STATISTICS OF THE ORIGINAL TITLE, REFORMULATED TITLE, AND POST BODY FOR DIFFERENT PROGRAMMING LANGUAGES

Language	Original Title Length				Reformulated Title Length				Question Body Length			
	Average	Mode	Median	<16	Average	Mode	Median	<16	Average	Mode	Median	<512
Java	8.48	7	8	95.64%	9.11	8	9	95.10%	135.75	49	92	97.34%
Python	8.63	7	8	95.87%	9.13	8	9	95.75%	131.27	50	92	97.89%
C#	8.70	7	8	95.00%	9.03	8	9	94.57%	137.40	41	98	97.50%
JavaScript	8.50	7	8	96.14%	9.07	8	9	95.75%	125.54	45	89	97.98%
PHP	8.31	7	8	96.54%	8.90	8	9	95.87%	127.05	52	87	97.89%
HTML	8.71	7	8	95.39%	9.26	8	9	95.18%	122.65	37	88	98.25%

and limitations of QETRA. Section VIII shows the novelty of our study by comparing related studies. Section IX concludes our study and shows potential future directions.

II. DATA COLLECTION

We gathered modification logs from two large-scale data dumps from Stack Overflow³. Stack Overflow releases data dumps of all its publicly available content roughly every three months via archive.org⁴ and the database schema documentation for these public data dumps can be found in the webpage⁵. The first data dump is *Posts*, which includes all question posts from July 2008 to March 2022. The second data dump is *PostHistory*, which includes the title reformulation history of all question posts from July 2008 to March 2022. In the data dump *PostHistory*, the modification of the post will be distributed to different records depending on the modified parts. For example, we show all the records for the title reformulation thread (shown in Fig. 1) in Table I. In this table, each historical record contains multiple attributes. Here we only show the meaning of some relevant attributes. Specifically, The attribute “*PostId*” represents the related post ID. The attribute “*PostHistoryTypeId*” represents the modification type according to the modification part (e.g., 1 denotes “Initial Title,” 4 denotes “Edit Title”). The attribute “*CreationDate*” represents the time when the modification occurred. The attribute “*UserId*” represents the user ID associated with the modification. The attribute “*Text*” represents the corresponding text. In this example, we can find the developer performed two successive title reformulations in the second row and the third row.

Due to many posts in Stack Overflow, we resort to popular tag statistical information provided by Stack Overflow⁶. After removing tags related to the system and framework (such as Android, and jquery), we finally select the top-six popular

programming languages. Specifically, we first use $\langle java \rangle$, $\langle c\# \rangle$, $\langle python \rangle$, $\langle javascript \rangle$, $\langle php \rangle$, and $\langle html \rangle$ tags to select corresponding posts from the data dump *Post*. Then, to guarantee the quality of our selected posts, we considered two heuristic selection rules based on the suggestions provided by previous studies [19], [20], [21], [22].

- **Rule 1:** The score of the selected question post is not smaller than 5.
- **Rule 2:** The selected question post should have the accepted answer.

We extracted related modification records for these selected posts according to “*PostId*” from the data dump *PostHistory*. Then we arranged related modification records chronologically for each post in a thread. In particular, supposing a thread t_1, t_2, \dots, t_n (Here t_1 denotes the original title, t_2 to t_n denotes the reformulated titles ordered in a chronological way), we can generate $n-1$ title reformulation pairs: $\langle t_1, t_n \rangle, \langle t_2, t_n \rangle, \dots, \langle t_{n-1}, t_n \rangle$, since we assume the quality of the title t_n is highest. Moreover, we only consider the post body’s latest content in this thread. After the above operations, we finally identified 122,528 title reformulation threads. From these threads, we extracted 152,259 title reformulation pairs. We show the length statistics of the original titles, the reformulated titles, and the contents of the post bodies in Table II. These statistics include average, mode, median, and the percentage of pairs less than the specified length. In this table, we can find that the contents of the Java and C# post bodies are longer than the other programming languages. On average, Java and C# post bodies contain 135.75 and 137.40 tokens respectively, while the post bodies of the other four programming languages contain around 125 tokens. Moreover, we find the original titles and the reformulated titles of all the programming languages almost contain the same number of tokens.

III. TITLE REFORMULATION PATTERN ANALYSIS

In this section, we conduct a formative study to understand the title reformulation patterns.

³<https://archive.org/download/stackexchange>, accessed in March 2022.

⁴<https://archive.org/details/stackexchange>

⁵<https://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede>

⁶<https://stackoverflow.com/tags>, accessed in March 2022.

TABLE III
THE TOP-10 MOST FREQUENT n -GRAMS IN THE TITLES OF THE JAVA POSTS AND THE PYTHON POSTS

Rank	1-gram		2-gram		3-gram		4-gram	
	Java	Python	Java	Python	Java	Python	Java	Python
1	java	python	how to	how to	how do i	how do i	is it possible to	object has no attribute
2	using	using	in java	in python	how can i	how can i	what is the difference	is it possible to
3	spring	pandas	can i	in a	what is the	what is the	is the difference between	is there a way
4	class	list	how do	how do	how to get	how to get	is there a way	there a way to
5	android	django	in a	can i	is there a	is there a	there a way to	what is the difference
6	method	file	what is	do i	how to use	how to use	how to get the	is the difference between
7	file	get	do i	how can	is it possible	a list of	how to create a	how to get the
8	use	string	how can	of a	the difference between	has no attribute	what is the best	how to create a
9	string	function	is the	way to	it possible to	object has no	how do i get	importerror no module named
10	get	dataframe	to use	a list	how to create	no module named	how to check if	how to check if

A. Who Edited Posts?

Stack Overflow allows users to edit three kinds of post information: post tags, the post title, and the post body. As of March 2022, there have been a total of 40,966,170 edited records of question posts. Among them, 3,373,872 (8.24%) are post title edits, 32,915,821 (80.35%) are post body edits, and 4,676,477 (11.41%) are post tag edits. Among all 3,373,872 post title edits, 1,097,760 (32.54%) are self-edits by the post owners, and 2,276,112 (67.46%) are edited by other experts. This statistical information shows that our proposed question title reformulation approach can be beneficial for both the post owners and other experts for title quality improvement.

B. What Are the Characteristics of Question Titles?

Question title content analysis. We analyzed question post titles to investigate the content of questions on Stack Overflow for our considered six programming languages. To achieve this goal, we first gathered all the words in the titles and performed standard text processing operations (such as punctuation removal, lowercase conversion, and stop word removal). After that, we identified the most common n -grams in these titles. Table III lists the top 10 most common 1-grams, 2-grams, 3-grams, and 4-grams in the titles of the Java posts and the Python posts (results of other programming languages can be found on our project homepage). In this table, we can find that programming languages (such as “Python” and “Java”), frameworks (such as “Spring” and “Django”), platforms (such as “Android”), data types (such as “String”), data structures (such as “Class” and “List”), and keywords (such as “Method” and “Function”) are the terms that appear most frequently in the title. Moreover, we also find that question words (such as “how to”) appear most in the titles, which appear in almost every 3-gram and 4-gram. Except for the question words, the most frequent terms are constraint words (such as “in python” and “in java.”), which can limit the question in a specific programming language.

Based on the title content analysis, we can find that a large number of terms related to the architecture (such as “Spring,” “Django,” and “Android”) are specific to programming language. However, except for these software-specific concepts, the high-frequency terms in the titles for different programming

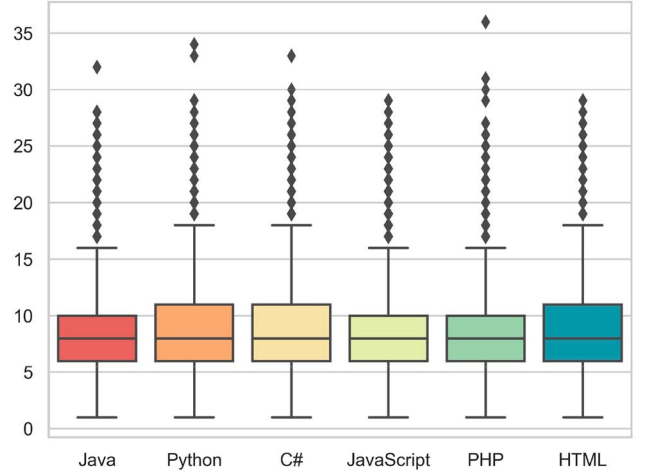


Fig. 2. The distribution of title length for different programming languages by using box plot.

languages also have a certain overlap. Therefore, it is reasonable to treat the question title reformulation for these six programming languages as separate but related tasks.

Question title length analysis. In the previous study, Baltadzhieva et al. [23] found that if the length of the question post titles is too long, these titles will have a negative effect on the quality of the question posts. While if the length of the question post titles is too short, these titles will not contain enough useful information. To investigate the title length range of our gathered posts, we compute the title length by using white space as a separator (i.e., word count). Notice we treat words satisfying CamelCase or underscore_case as a word since these words are identifiers in the code snippets and are often used as variable names, class names, or method names in the title.

We show the title length distribution via box plot in Fig. 2. In this figure, we can find the mean of the title length for different programming languages is around 8, which shows that developers do intentionally limit the length of titles to make the questions easier to answer. Moreover, there is a high consistency degree in the title length across six programming languages. However, we also find some outliers have larger lengths, with values between 18 and 36. These outlier posts may not satisfy the guidelines of writing perfect question

TABLE IV
CATEGORIES OF TITLE REFORMULATION PATTERNS AND THEIR SUB-CATEGORIES, CORRESPONDING EXAMPLES, AND THE PROPORTION

Category	Sub-category	Example	Proportion
Add	Software or platform	Original Title: how to convert a json to a php object? Reformulated Title: how to convert a json to a php object <u>in symfony2?</u>	3.48%
	Detailed requirement	Original Title: Can I pass arguments to the GWT compiler? Reformulated Title: Can I pass arguments (<u>deferred binding properties</u>) to the GWT compiler?	12.56%
Category Proportion			16.04%
Modify	Spelling and syntax check	Original Title: Java: How to get the scrolling method OS X Lion? Reformulated Title: Java: How to get the scrolling method <u>in</u> OS X Lion?	32.21%
	Simplify and refine	Original Title: C# how do I write <u>in the code this char — ‘\’</u> Reformulated Title: C# how do I write the escape char — ‘\’ <u>to code</u>	23.38%
	Large-scale modification	Original Title: Don’t work JSplitPane between JButton and Jtable Reformulated Title: <u>Can’t change size of block in BorderLayout</u>	10.45%
Category Proportion			66.04%
Delete	Detailed or unnecessary words	Original Title: What are the <u>current</u> options for making a simple static website multilingual? Reformulated Title: What are the options for making a simple static website multilingual?	12.81%
	Specific information in error message	Original Title: MongoDB <u>throws java.lang.IllegalArgumentException: Invalid BSON field name</u> Reformulated Title: MongoDB upsert exception invalid BSON field	0.25%
	Symbols or web links	Original Title: JavaScript Loops: for ... in vs for Reformulated Title: JavaScript Loops: for in vs for	3.48%
Category Proportion			16.54%
Others			1.37%

titles⁷ and can affect the timely reply of these posts [21]. Finally, We analyzed the posts with too short titles. After analyzing some randomly selected posts, we find most of these titles contain error logs or questionable API names (such as “*java.lang.UnsatisfiedLinkError*,” “*Boolean.hashCode()*”).

C. Why Are Question Titles Reformulated?

The developers may reformulate the question title for many reasons (such as misspelled words). In this subsection, we conduct a small-scale human study to investigate the title formulation reasons for different programming languages. In this human study, we hire two master students. Then we randomly select 300 reformulation pairs (50 reformulation pairs for each programming language). Later, for each selected reformulation pair, two students will categorize the reason for this pair. If two students disagree, they discussed it with each other until they reach a consensus.

Table IV shows our manual categorization results and corresponding examples. For these examples, we emphasize the reformulation part in the underlined manner. In this table, we can find that modifying the title is the most common title reformulation pattern, which accounts for 66.04% of all pairs, then comes adding more information to the title (16.04%), and deleting information from the title (16.54%). Notice we put other pairs that are difficult to classify into the Others category (1.37%). Specifically, in the Add category, we further divide it into two sub-categories: (1) adding constraint information, such as programming languages or platforms, and (2) adding

detailed requirements for the question. In the Modify category, we further divide it into three subcategories: (1) spelling and syntax checking, (2) simplifying and refining the title into the most commonly-used expression, and (3) large-scale title modification, which can better reflect the core content of the post. Compared with this sub-category, the modification scale of the first two sub-categories is much smaller. In the Delete category, we further divide it into three sub-categories: (1) deleting unnecessary words or words with little information, (2) deleting specific information in error messages or code snippets (such as file path, URL, function names, etc.), (3) deleting punctuation or mistyped symbols.

Based on the above categorization result, we can observe different types of question title reformulation patterns (such as the Add category, the Delete category, and the Modify category). Considering the diversity of title reformulation patterns, it is not practical to use a rule-based approach to solve this problem. Therefore, we propose a general post title reformulation approach based on deep learning, which can learn the knowledge required for title reformulation from our gathered large-scale dataset.

D. What is the Scale of Question Title Reformulation?

We use the character-level LCS (Longest Common Subsequence) [24] between the original title and the reformulated title to measure the similarity of this pair. The similarity can be computed as follows.

$$\text{similarity}(\text{original}, \text{reformulated}) = \frac{2 \times N_{\text{match}}}{N_{\text{total}}} \quad (1)$$

⁷<https://codeblog.jonskeet.uk/2010/08/29/writing-the-perfect-question/>

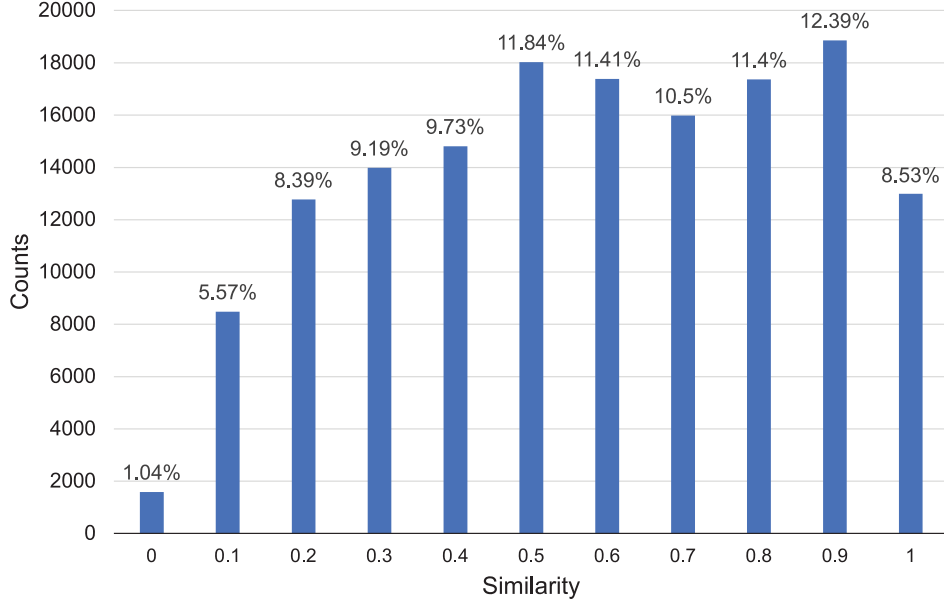


Fig. 3. The similarity score distribution of our gathered title reformulation pairs.

where N_{match} denotes the number of characters in the LCS and N_{total} denotes the sum of the *original* and *reformulated* characters. The similarity score is between 0 and 1. The higher the similarity score, the smaller the reformulation scale between the original title *original* and the reformulated title *reformulated*.

As shown in Fig. 3, among 152,259 title reformulation pairs, 42.82% of the pairs are of high similarity (i.e., the similarity score is larger than 0.6), 32.98% of the pairs are almost similar (i.e., the similarity score is between 0.4 and 0.6), and 24.19% of the pairs are of low similarity (i.e., the similarity score is not larger than 0.3). In this figure, we can find most of the title reformulation pairs are small-scale.

Our investigation results show that the scale of title reformulation is not uniform.

E. Implications of Our Informative Study

Since the previous studies [4], [13] on direct question post title generation were only able to assist question post owners in writing high-quality titles before posting question posts, it did not consider how to assist question post owners and other developers in reformulating titles when the posted posts already have the original titles. Therefore, we further propose a new task to complement the previous studies, i.e., to assist question post owners and other experts in reformulating the title after the question post is posted, which aims to further improve the quality of the question post title. Considering the diversity of title reformulation patterns, it is not practical to use a rule-based approach to solve this problem. According to previous research [25], the approach based on deep learning can automatically learn diverse patterns from large-scale data. Therefore, we can propose a general title reformulation approach based on deep learning. Moreover, given the large number of software-specific concepts in question posts for different languages, reformulating the question titles for different programming languages

cannot be simply regarded as a single task. However, except for these software-specific concepts, the other contents in the post titles for different programming languages have a certain similarity. Therefore, we can formalize the question title reformulation for each programming language as separate but related tasks, which can allow related tasks to improve each other's performance by using shared and complementary information via multi-task learning.

In summary, we think it is necessary and feasible to propose an automated post title reformulation approach based on deep learning and multi-task learning. Our study could benefit both developers and the Stack Overflow community. For example, the developers can use our proposed approach to refine their original post titles and then improve the quality of the post, which can help to get a timely reply from related developers.

IV. AUTOMATIC TITLE REFORMULATION APPROACH

The framework of our approach is shown in Fig. 4. In this figure, we can find QETRA contains three phases: corpus construction phase, model construction phase, and model application phase. Specifically, (1) in the corpus construction phase, we extracted the title edit records and the body of question posts from the data dump *PostHistory* and the data dump *Post* of Stack Overflow respectively. In our constructed dataset, we mainly focus on title edit records related to six popular programming languages. More details of the dataset construction process can be found in Section II. (2) In the model construction phase, we first concatenate the original title and the question body to model the multi-modal input. Then, we separate these two modalities by using the SentencePiece method [26] to solve the OOV (out of vocabulary) problem. Later, we model question post title reformulation for different programming languages as independent but related tasks. Then we use multi-task learning [16] to solve these tasks. Finally, we construct our model by fine-tuning a pre-trained Transformer model T5 [17]. (3) In the

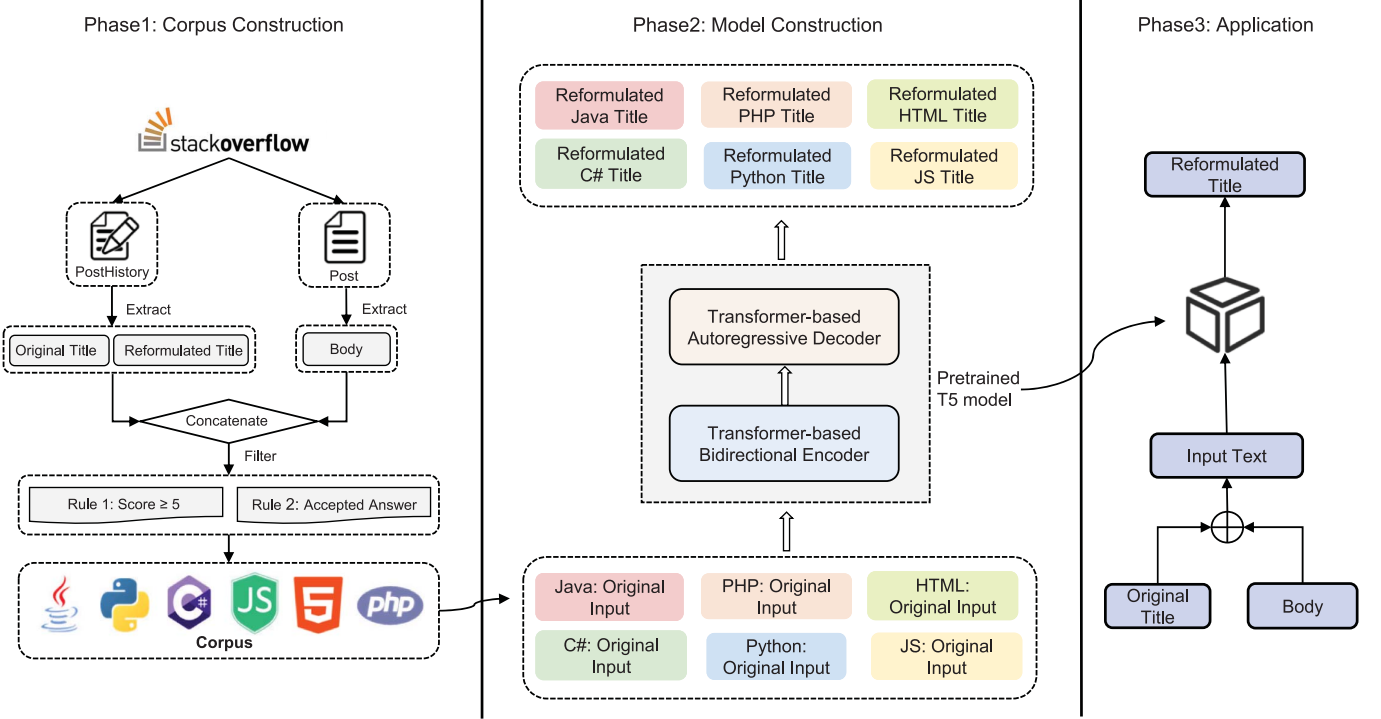


Fig. 4. Framework of our proposed approach QETRA.

model application phase, for a question post title that needs reformulation, we input its original title, related programming language, and the body of the question post to the trained model. Then the trained model can provide a set of candidate reformulated titles for selection through the beam search algorithm. In the rest of this section, we show the technical details of the model construction phase.

A. Encoder-Decoder Model Based on Transformer

Our encoder-decoder Model is based on T5 [17], which is similar to Transformer [27], but the difference is to remove the layer norm bias, place the layer normalization outside the residual path, and use a different position embedding scheme. When receiving an input sequence, we map the input sequence of tokens $X = (x_1, x_2, \dots, x_m)$ to an embedding sequence, which is then sent into the encoder. All of the encoders have the same structure and are made up of two subcomponents (i.e., a self-attention layer and a small feed-forward network). The calculation of self-attention [28] is based on queries (Q), keys (K), and values (V). Specifically, The dot product of the queries and keys is first computed. Then each is divided by $\sqrt{d_k}$ and the softmax function is used to get the weight of the corresponding value.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2)$$

The feed-forward neural network (FFN) is made up of two linear transformations with the Relu activation, which can provide a nonlinear transformation.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (3)$$

Layer normalization [29] is performed on the inputs of each child component. The layer normalization is a simplified version where the activations are only rescaled and no additive bias is applied. Following layer normalization, a residual skip connection [30] adds each child component's inputs to its output. Dropout [31] is applied within the feed-forward network, on the skip connection, on the attention weights, and at the input and output of the entire stack.

The decoder has a similar structure as the encoder. The distinction is that after each self-attended layer, it employs a standard attention mechanism to focus on the encoder output. The decoder's self-attention mechanism also employs a form of autoregressive or causal self-attention, which allows the model to only focus on previous outputs. To construct the output probabilities over the vocabulary, the output of the last decoder block is fed into a dense layer with a softmax output.

B. Multi-Task Learning

We represent the multi-modal input in our study by concatenating the original title with the body of the question post. Specifically, we concatenate the original title sequence $X_{\text{originalTitle}}$ and the post body sequence X_{body} via a special identifier ($\langle \text{body} \rangle$) to distinguish $X_{\text{originalTitle}}$ and X_{body} . We also investigate a multi-task learning scenario in which a shared model is trained on multiple tasks at the same time. Multi-task learning has been proven to increase model generalization capabilities in NL pre-training by reusing the majority of model weights for many tasks [16], [32], which can effectively reduce computational costs. As shown in Fig. 4, we prefixed the input X for each programming language with task-specific prefixes (e.g., the prefix "JS:" indicates the programming

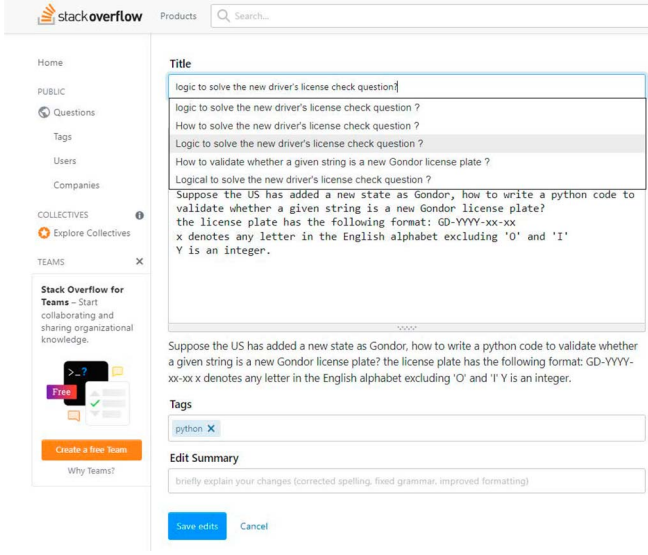


Fig. 5. The screenshot of our developed plugin based on QETRA.

language JavaScript) to allow the model to discriminate different tasks since we regard post title reformulation for the different programming languages as independent but related tasks. The input format is designed as follows.

$$X = \text{prefix} \oplus X_{\text{originalTitle}} \oplus \langle \text{body} \rangle \oplus X_{\text{body}} \quad (4)$$

C. SentencePiece

Since our study needs to deal with the OOV problem for different programming languages at the same time, we use the SentencePiece method [26] to process the inputs X . SentencePiece is a language-independent subword tokenizer and detokenizer for neural machine translation and other neural-based text processing tasks. Previous subword segmentation techniques presume that input is pre-tokenized into word sequences, whereas SentencePiece can train subword models straight from raw sentences, which makes it possible to work with a variety of programming languages and multi-modal inputs.

D. Beam Search

In QETRA, we use a heuristic search strategy (i.e., beam search) to find titles that have the least cost. Beam Search is improved on the greedy search by returning a list of the most likely output sequences. It scans through each step's title tokens one by one and chooses the k tokens with the lowest cost at each time step, where k denotes the beam width. After pruning any remaining branches, it continues to choose potential tokens for the subsequent tokens until it encounters the end-of-sequence sign. Finally, QETRA can return k candidate titles for each question post. We rank the generated candidate titles according to their average probabilities during the beam search procedure.

E. Tool Support

To make our proposed approach more practical, we developed a browser plugin based on QETRA and integrated it into

the Chrome browser. The screenshot of our developed tool is shown in Fig. 5. The usage process can be illustrated as follows. When a user edits a question title, this browser plugin will automatically analyze the original title and the question body and recommend the top-5 reformulated titles to the users for selection. Our plugin can be downloaded on our project homepage.

V. EXPERIMENTAL SETUP

A. Research Questions

In our empirical studies, we want to answer the following four research questions (RQs).

RQ1: Can our proposed approach QETRA generate higher-quality reformulated titles than the state-of-the-art baselines by automatic evaluation?

Motivation: In this RQ, we want to demonstrate the competitiveness of our proposed approach QETRA by comparing state-of-the-art baselines in an automatic way. First, we should select baselines from similar research problems since we are the first to study this problem to the best of our knowledge. Second, we should choose automatic evaluation metrics to compare the quality of the reformulated titles generated by different approaches from multiple perspectives.

RQ2: What is the effect of using the bi-modal information for QETRA?

Motivation: In this RQ, we want to analyze whether using the bi-modal information (i.e., original question title and question post body) can help to improve the performance of our proposed approach QETRA. Further, we can also analyze which modal information can provide more help for QETRA.

RQ3: What is the effect of using multi-task learning in QETRA?

Motivation: Using multi-task learning can help to augment the training data, which can enable QETRA to learn more useful information. However, using multi-task learning may also introduce noises and make it harder to train the models [16]. Therefore, we design this RQ to investigate whether using multi-task learning can improve the performance of QETRA.

RQ4: Can our proposed approach QETRA generate higher-quality reformulated titles than the state-of-the-art baselines by human study?

Motivation: Since evaluation metrics used in the automatic evaluation are measured based on the overlap between the ground truth titles and the generated titles, this cannot effectively reflect the semantic differences between different titles. Therefore, we conduct a human study to evaluate the effectiveness of our proposed approach in this RQ.

B. Dataset

As discussed in Section II, we identified 122,528 title reformulation threads after mining modification logs from Stack Overflow. From these threads, we extracted a total of 152,259 title reformulation pairs.

For the question post body, we perform a set of data preprocessing as follows. If the post body contains the code snippet,

TABLE V
STATISTICAL INFORMATION OF OUR GATHERED DATASET FOR SIX
PROGRAMMING LANGUAGES

Language	Training		Validation		Testing	
	Pair	Thread	Pair	Thread	Pair	Thread
Java	24,821	20,081	3,112	2,510	3,073	2,511
Python	24,602	19,601	3,050	2,450	3,011	2,451
C#	26,732	21,929	3,334	2,741	3,311	2,742
JavaScript	24,264	19,220	3,045	2,403	3,004	2,403
HTML	10,592	8,392	1,299	1,049	1,320	1,050
PHP	10,960	8,796	1,347	1,099	1,382	1,100
Total	121,971	98,019	15,187	12,252	15,101	12,257

we extract the code snippet by utilizing code tags (such as `<code>`, `<pre>`). Then we concatenate the extracted code snippet after the problem description by the tag `<code>`. Later we remove the tags and garbled parts in the problem description. Moreover, we notice the links in the problem description can cause the title length to be too long. Therefore, we also remove the links in the problem description.

As discussed in Section IV, the input of QETRA considers both title and corresponding post body. If we split the dataset in terms of pair granularity, the title reformulation pairs in the same thread may exist in different split parts. Since these pairs share the same post body, this split strategy may cause a data leakage problem. Therefore, we split the dataset in terms of thread granularity. Specifically, we split the dataset into the training set, the validation set, and the testing set in a ratio of 80%, 10%, and 10%. Finally, we randomly select 121,971 title reformulation pairs from 98,019 threads as the training set, select 15,187 pairs from 12,252 threads as the validation set for hyper-parameter optimization, and the remaining 15,101 pairs from 12,257 threads as the testing set to evaluate the performance of our proposed approach QETRA. The detailed statistical information for different programming languages can be found in Table V.

C. Baselines

As discussed in Section III, many title reformulations were performed due to grammatical errors. Therefore, we first consider a widely used grammatical error correction (GEC) tool as the first baseline.

LanguageTool. LanguageTool⁸ is an open-source proofreading tool that supports over 20 languages. The style and grammar checker in this tool is rule-based and has been under development for more than 10 years.

To the best of our knowledge, we are the first to study automated question title reformulation by mining reformulation logs from Stack Overflow. In our study, we regard the question title reformulation task as a neural translation problem (i.e., translating the original title into the reformulated title). Therefore, we further consider five baselines from the field of source code understanding, which also models the corresponding tasks as neural translation problems. For a fair comparison, these baselines also consider the same inputs as our proposed approach.

NMT. Jiang et al. [33] employed NMT (Neural machine translation) technology to automatically “translate” code changes into commit messages. We select NMT as a baseline since it can achieve promising results in generating commit messages for code changes.

BiLSTM-CC. This is an LSTM-based deep learning method to automatically generate titles for question posts from code snippets in Stack Overflow [4]. BiLSTM-CC is based on sequence-to-sequence architecture and enhanced with an attention mechanism to perform better content selection, a copy mechanism to handle the OOV problem within the input as well as a coverage mechanism to avoid meaningless repetitions.

Transformer. Transformer [27] is an attention-only deep learning model that uses positional encoding to better encode location information, which allows the network to capture long-term information and dependencies between sequential tokens through the self-attention mechanism. Recently, Transformer has been widely used in source code understanding tasks (such as source code summarization [34], [35], [36], pseudo-code generation [37], shellcode generation and summarization [38]).

BART. BART [39] is a denoising autoencoder for pretraining sequence-to-sequence models, which is trained by corrupting text with an arbitrary noising function and learning a model to reconstruct the original text. It uses a standard Transformer-based neural machine translation architecture which can be regarded as generalizing BERT due to the bidirectional encoder and GPT with the left-to-right decoder. When fine-tuned for neural machine translation, BART is very effective. BART is also used in source code understanding tasks (such as pull request title generation [40]).

CodeBERT. CodeBERT [18] is a Transformer-based bimodal pre-trained model for the programming language (PL) and the natural language (NL), trained with a hybrid objective function that incorporates the pre-training task of replaced token detection, which is to detect plausible alternatives sampled from generators. CodeBERT can learn general-purpose representations that support downstream NL-PL applications (such as code documentation generation [41], [42]), and achieve promising performance. Although CodeBERT does not use HTML and C# programming language corpus for pre-training, we still select it as our baseline due to its high generalization ability.

For the baselines that do not share their code scripts, we implement these baselines according to their original descriptions. The results of our re-implementations are very close to the results reported in the original studies. Specifically, we use the open-source framework OpenNMT⁹ to implement the baselines NMT, Transformer, and BiLSTM-CC. We use the open-source framework SimpleTransformer¹⁰ to implement the baseline BART. For the remaining baselines, we utilize the codes shared by the original studies. To ensure a fair comparison between QETRA and the baselines, we employed the same data split strategy and optimized the hyper-parameters in these baselines.

⁸<https://languagetool.org/>

⁹<https://opennmt.net/>

¹⁰<https://simpletransformers.ai/>

D. Evaluation Metrics

To compare the quality of the reformulated titles, we consider the following three different types of evaluation metrics from different perspectives.

Since we treat post title reformulation as a neural machine translation task, we first measure the quality of the reformulated titles based on three overlap-based metrics.

BLEU. BLEU (Bilingual Evaluation Understudy) metric [43] is commonly used to assess the quality of machine translation. It calculates the similarity between a reformulated title and a ground-truth title using a reformulated n -gram precision p_n . Intuitively, BLEU first computes the reformulated n -gram precision. Then, it uses n -grams up to length N and positive weights w_n . Finally, it adds the geometric average of p_n to one and is multiplied by an exponential shortness penalty factor. In our study, we set n to 4 to ensure the appropriate n -gram overlap.

ROUGE-L. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [44] is a measure based on Recall. It is used to calculate the length of the longest common subsequence between the reformulated title and the ground-truth title.

CIDEr. CIDEr (consensus-based image description evaluation) [45] mainly measures the quality of image title generation. It takes into account the frequency of occurrence of n -grams in ground-truth titles by calculating the TF-IDF value for each n -gram.

As analyzed in Section III, many post titles are reformulated to fix grammatical errors. Therefore, we also consider two evaluation metrics commonly used in grammatical error correction.

Exact Match. Exact Match (EM) assesses the probability of a perfect match between the reformulated title and the ground-truth title. Only when the two titles are identical, this metric can identify it as a positive case. QETRA can recommend multiple reformulations for an original title since it leverages beam search during decoding. Consequently, for different beam sizes, we may compute EM@1, EM@5, and EM@10, where EM@ n denotes that one case would be regarded positive if one of the n reformulation results generated by beam search fits the ground truth.

GLEU. GLEU (General Language Evaluation Understanding) [46] is a BLEU-customized metric. Because just a portion of the original title will be reformulated in the GEC task, which differs from the machine translation task, this motivates a slight tweak to BLEU that computes n -gram accuracy over the ground-truth title but gives greater weight to n -grams that have been successfully changed from the original title.

Finally, our approach employs the beam search algorithm, which can return multiple recommended reformulated titles. Therefore, we also evaluate the performance of our approach by calculating the rank of the correct reformulated title in the list of recommended reformulated titles. Notice this performance metric can evaluate the practicability of QETRA in retrieving posts in Stack Overflow.

MRR. MRR (Mean Reciprocal Rank) [47] is a general recommendation algorithm evaluation metric. For the automated

TABLE VI
HYPER-PARAMETERS AND THEIR VALUES USED BY OUR
PROPOSED APPROACH QETRA

Hyper-parameter Name	Hyper-parameter Value
encoder_layers	12
decoder_layers	12
max_input_length	512
max_output_length	48
hidden_size	768
dropout	0.1
beam search size	5

question title reformulation task, in the reformulated title list, the first reformulated title matching score is 1, the second matching score is 0.5, and the n -th matching score is $1/n$. If there is no matching title, the score is 0. Otherwise, the final score is the average of the sum of all scores.

For these metrics, the scores of BLEU, ROUGE-L, Exact Match, GLEU, and MRR are in the range of [0,1] and reported in percentages. CIDEr is in the range of [0,10] and reported in real values. The higher the value of these metrics, the better the performance of the corresponding approach.

E. Implementation Details and Running Platform

We implement our proposed approach based on the PyTorch framework¹¹. we use Transformers¹² to implement our proposed approach QETRA. The hyper-parameters and their values can be found in Table VI. The optimal values of these hyper-parameters are set according to our best practices in the empirical study. To further alleviate the overfitting problem, we employ the early stop strategy [48]. Finally, since the training steps for different tasks to achieve the best performance is different in multi-task learning, we choose the training steps with the best performance for different tasks.

We run all the experiments on a computer with an Intel(R) Xeon(R) Silver 4210 CPU and a GeForce RTX3090 GPU with 24 GB memory. The running OS platform is Windows OS.

VI. RESULT ANALYSIS

A. Result Analysis for RQ1

RQ1: Can our proposed approach QETRA generate higher-quality reformulated titles than the state-of-the-art baselines by automatic evaluation?

Table VII shows the performance of QETRA and baselines in terms of all the evaluation metrics for different programming languages. In this table, we highlight the best performance in boldface for each metric. Notice that since LanguageTool can only return one result, we cannot calculate performance values in terms of metrics EM@5, EM@10, and MRR. Moreover, we can find that for HTML, the score of the baseline NMT is 0 in terms of the metrics EM@1 and EM@5. This means there are no titles in the reformulated titles generated by NMT that can exactly match the ground truth.

¹¹<https://pytorch.org/>

¹²<https://github.com/huggingface/transformers>

TABLE VII
COMPARISON RESULTS BETWEEN OUR PROPOSED APPROACH AND STATE-OF-THE-ART BASELINES

Language	Approach	BLEU_4(%)	ROUGE_L(%)	CIDEr	EM@1(%)	EM@5(%)	EM@10(%)	GLEU(%)	MRR(%)
Python	NMT	17.81	17.98	1.43	0.13	0.53	0.80	20.40	0.40
	Transformer	20.04	20.06	1.74	0.40	1.23	1.76	22.87	0.73
	BiLSTM-CC	42.06	36.37	3.81	3.29	6.41	7.90	41.37	5.08
	BART	41.30	35.16	3.71	0.03	0.30	0.43	39.83	16.86
	CodeBERT	30.98	51.43	2.90	4.05	7.21	9.27	32.67	6.71
	LanguageTool	45.39	61.07	4.07	5.31	—	—	43.78	—
	QETRA	50.25	65.01	4.72	13.32	25.91	30.06	49.52	24.51
C#	NMT	21.66	20.06	1.69	0.42	0.91	1.18	22.84	0.63
	Transformer	21.51	21.42	1.84	0.82	1.51	1.84	23.78	1.28
	BiLSTM-CC	47.51	38.79	4.21	6.89	10.48	12.08	45.72	9.40
	BART	45.55	39.76	4.05	5.04	10.42	12.90	43.00	14.14
	CodeBERT	37.03	56.95	3.46	6.95	12.23	14.47	38.52	12.89
	LanguageTool	49.76	63.76	4.31	6.04	—	—	46.29	—
	QETRA	54.22	68.05	5.04	18.51	31.38	34.76	52.89	30.97
Java	NMT	18.08	18.33	1.41	0.26	0.55	0.62	20.40	0.40
	Transformer	16.15	17.92	1.36	0.29	0.65	0.78	19.15	0.59
	BiLSTM-CC	41.95	36.25	3.72	2.80	5.82	6.83	40.66	4.53
	BART	40.32	35.30	3.57	0.00	0.33	0.78	38.27	16.48
	CodeBERT	30.51	51.96	2.89	4.59	5.99	8.07	32.80	6.63
	LanguageTool	45.57	61.40	4.09	6.67	—	—	44.17	—
	QETRA	49.54	64.84	4.65	14.71	25.58	29.55	48.99	24.97
JavaScript	NMT	18.61	18.97	1.52	0.63	1.26	1.46	20.48	0.92
	Transformer	20.31	20.65	1.78	1.00	1.83	2.23	22.94	1.37
	BiLSTM-CC	43.99	37.17	3.86	4.26	7.59	9.09	42.01	6.37
	BART	42.35	35.71	3.71	0.07	2.23	4.13	39.44	17.29
	CodeBERT	30.55	51.53	2.89	4.96	7.22	9.22	32.74	6.24
	LanguageTool	47.15	61.83	4.17	8.46	—	—	44.83	—
	QETRA	50.88	65.20	4.72	15.31	26.00	30.13	49.69	25.99
HTML	NMT	8.35	13.34	0.72	0.00	0.00	0.08	12.15	0.00
	Transformer	17.28	19.23	1.62	0.91	1.36	1.89	20.90	4.85
	BiLSTM-CC	37.87	33.68	3.33	2.42	3.86	4.70	36.45	3.44
	BART	40.14	35.11	3.58	0.00	0.38	0.76	38.40	16.94
	CodeBERT	19.34	40.80	1.81	1.59	2.73	3.79	22.47	2.30
	LanguageTool	44.62	60.88	4.02	7.50	—	—	43.21	—
	QETRA	49.01	63.89	4.54	14.62	25.83	28.26	48.01	22.87
PHP	NMT	7.05	12.01	0.66	0.07	0.07	0.07	11.88	0.07
	Transformer	18.08	19.53	1.64	0.43	1.01	1.23	21.48	1.09
	BiLSTM-CC	40.03	35.87	3.60	2.89	5.50	6.01	39.18	4.27
	BART	40.32	36.51	3.58	0.14	1.37	2.24	38.29	15.49
	CodeBERT	19.12	41.42	1.82	1.59	2.39	3.04	22.54	2.94
	LanguageTool	46.69	61.90	4.13	7.74	—	—	44.56	—
	QETRA	50.32	65.36	4.68	14.83	26.77	29.31	49.50	25.06

Based on these comparison results, we can find QETRA outperforms all of our considered state-of-the-art baselines in terms of all metrics. From these tables, we can achieve the findings as follows.

Our approach achieves the best performance for all programming languages in terms of all the evaluation metrics. For these deep learning-based baselines, the baselines BiLSTM-CC, BART, and CodeBERT can achieve better performance than other baselines (i.e., NMT, Transformer). Taking the Python programming language as an example, compared to CodeBERT, QETRA can improve the performance by 62.22%, 26.39%, 62.81%, 228.69%, 259.45%, 224.37%, 51.59% and 264.94% for BLEU_4, ROUGE_L, CIDEr, EM@1, EM@5, EM@10, GLEU, and MRR respectively; Although T5, BART, and CodeBERT are all large-scale pre-trained models, our approach can achieve better performance than BART and CodeBERT. Motivated by the previous validity analysis of

pre-trained models on software engineering tasks [49], we summarize the potential reasons into two aspects. Firstly, in the corpus selection for pre-training, T5 and BART are pre-trained on general language corpora that contain various types of natural language texts (such as Wikipedia, news articles, academic papers, blog posts, and web pages). In contrast, the corpora used for CodeBERT's pre-training are obtained from CodeSearchNet datasets. These corpora were collected from open-source projects in GitHub and focused more on the structure and syntax of program code. As our title reformulating task requires processing both code and natural language and natural language texts play a more important role than the code (discussed in Section VI-B), T5 and BART can perform better than CodeBERT in most cases. Secondly, in the selection of pre-training tasks, T5's pre-training tasks include supervised generative tasks (such as machine translation and text summarization), which are highly matched with the question title reformulating task than BART.

Original Title :	Dictionaries of dictionaries merge	Reformulated Title :
Question Body:	<p>I need to merge multiple dictionaries, here's what I have for instance:</p> <pre>dict1 = {1:{"a":{A}}, 2:{"b":{B}}}</pre> <pre>dict2 = {2:{"c":{C}}, 3:{"d":{D}}}</pre> <p>With A B C and D being leaves of the tree, like {"info1":"value", "info2":"value2"}</p> <p>There is an unknown level(depth) of dictionaries, it could be {2:{"c":{"z":{"y":{"C}}}}}</p> <p>In my case it represents a directory/files structure with nodes being docs and leaves being files.</p> <p>I want to merge them to obtain:</p> <pre>dict3 = {1:{"a":{A}}, 2:{"b":{B},"c":{C}}, 3:{"d":{D}}}</pre> <p>I'm not sure how I could do that easily with Python.</p>	<p>Ground Truth : How to merge dictionaries of dictionaries?</p> <p>NMT : Use of dictionaries</p> <p>Transformer : Comparison of dictionaries</p> <p>BiLSTM-CC : How to merge a dictionaries merge</p> <p>BART : Dictionaries of dictionaries merge</p> <p>CodeBERT : Dictionary of dictionaries in Python?</p> <p>LanguageTool : Dictionaries of dictionaries merge</p> <p>QETRA : How to merge dictionaries of dictionaries?</p>

Fig. 6. The titles reformulated by QETRA and baselines for a question post related to the Python programming language.

Therefore, QETRA based on T5 can achieve better performance than BART.

For programming languages (i.e., Java, Python, JavaScript, and C#) with sufficient training data, the baseline BiLSTM-CC performs better than the baseline BART. However, for programming languages (i.e., HTML and PHP) with limited training data, BART performs better than BiLSTM-CC. The possible reason is that BiLSTM-CC can learn useful information when the training data is sufficient, while Bart is suitable for the limited training data by fine-tuning the pre-trained model. The baseline LanguageTool can achieve the best performance in terms of almost all metrics since our previous formative study showed that 32.21% of the title reformulations are related to spelling and syntax error modification (in Section III), LanguageTool can achieve high performance in terms of Rouge-L and GLEU metrics as it is a mature grammatical error correction tool. However, we find LanguageTool cannot achieve satisfactory performance in terms of the metric EM@n since this baseline cannot effectively identify software-specific concepts and identifiers of programming languages. For example, when the original title was “How do i change JPanel inside a JFrame on the fly?,” LanguageTool reformulated it to “How do I change Panel inside a Frame on the fly?.” Although LanguageTool correctly modified “i” to “I,” it incorrectly modified the correct software-specific concepts “JPanel” and “JFrame” to “Panel” and “Frame.” Moreover, our proposed approach QETRA is specifically optimized for the question title reformulation task by mining modification logs from Stack Overflow, which is not practical to use a rule-based approach to solve this problem (Discussed in Section III). Therefore, QETRA can better capture the thought process and habits of developers of modifying titles, and thus generate titles that better match developers’ expectations.

By following the previous studies [50], [51], we also perform Wilcoxon signed-rank tests [52] at the confidence level of 95% to check whether the performance differences between QETRA and baselines are significant. Specifically, for each approach, we

consider the evaluation score for each reformulated title in the testing set in terms of a specific performance measure. Then we perform the Wilcoxon signed-rank test for each pair of QETRA and the baseline by considering these scores. All the p -values are smaller than 0.05, which means our proposed approach QETRA can significantly improve over the baselines. Later, we use Cliff’s delta (δ) [53], which is a non-parametric effect size measure, to quantify the amount of difference between the two approaches. We consider the values of the delta that are less than 0.147, between 0.147 and 0.33, between 0.33 and 0.474 and above 0.474 as “Negligible (N),” “Small (S),” “Medium (M),” and “Large (L)” performance difference, respectively by following the suggestion [53]. The results show that QETRA can outperform the next best baseline (i.e., LanguageTool) with a large improvement with respect to Cliff’s delta. For example, in terms of BLEU_4, the value of Cliff’s delta is at least 0.80 for different programming languages.

Fig. 6 shows an example question post for Python. The left part of this figure shows the original title and its question body. The right part of this figure shows the ground truth and the reformulated titles generated by QETRA and baselines. In this example, we can find that the titles reformulated by NMT and Transformer are not relevant to the question post. The title reformulated by BiLSTM-CC is not a correct sentence. The titles reformulated by LanguageTool and BART are unmodified. The title reformulated by CodeBERT has low readability and cannot convey the core content of the question body. In contrast, the title reformulated by QETRA can not only express the core content of the question post concisely and accurately but also has high readability.

Answer to RQ1: QETRA can achieve better performance than six state-of-the-art baselines (i.e., a baseline based on grammatical error correction and five baselines based on deep learning) in automatic evaluation based on three different types of evaluation metrics.

TABLE VIII
THE PERFORMANCE OF QETRA WITH DIFFERENT COMBINATIONS OF INPUT MODALITIES

Language	Approach	BIEU_4(%)	ROUGE_L(%)	CIDEr	EM@1(%)	EM@5(%)	EM@10(%)	GLEU(%)	MRR(%)
Python	QETRA _{title}	49.40	64.40	4.58	10.03	22.98	26.40	48.37	21.20
	QETRA _{body}	5.02	19.32	0.50	0.17	0.37	0.50	7.85	0.23
	QETRA	50.25	65.01	4.72	13.32	25.91	30.06	49.52	24.51
C#	QETRA _{title}	53.35	67.35	4.87	14.89	27.09	30.59	51.54	26.04
	QETRA _{body}	4.76	18.09	0.45	0.30	0.66	1.03	7.37	0.60
	QETRA	54.22	68.05	5.04	18.51	31.38	34.76	52.89	30.97
Java	QETRA _{title}	48.94	64.15	4.50	11.58	23.43	26.68	47.92	21.93
	QETRA _{body}	4.81	18.61	0.48	0.49	0.81	1.07	7.50	0.68
	QETRA	49.01	63.89	4.54	14.62	25.83	28.26	48.01	24.97
JavaScript	QETRA _{title}	50.04	64.15	4.54	11.05	23.77	27.30	48.02	21.04
	QETRA _{body}	5.12	18.78	0.53	0.37	0.93	1.00	7.81	0.60
	QETRA	49.54	64.84	4.65	14.71	25.58	29.55	48.99	25.99
HTML	QETRA _{title}	46.63	62.61	4.29	9.70	21.14	24.77	45.78	18.81
	QETRA _{body}	4.98	19.21	0.51	0.08	0.53	0.91	7.85	0.48
	QETRA	50.32	65.36	4.68	14.83	26.77	29.31	49.50	22.87
PHP	QETRA _{title}	49.20	64.00	4.48	12.08	23.44	25.98	47.74	22.67
	QETRA _{body}	4.43	18.45	0.50	0.43	0.58	0.80	7.48	0.68
	QETRA	50.88	65.20	4.72	15.31	26.00	30.13	49.69	25.06

B. Result Analysis for RQ2

RQ2: What is the effect of using the bi-modal information for QETRA?

In this RQ, we aim to investigate the contribution of different input modalities to the performance of QETRA. Here we use different subscripts to distinguish different control approaches and the meaning of these subscripts is illustrated as follows.

- **title.** The corresponding control approach only uses the original title as the input.
- **body.** The corresponding control approach only uses the question post body as the input.

Table VIII shows the performance of QETRA with different combinations of input modalities. Take the programming language Python as an example, compared to training models with only question post bodies, QETRA can improve the performance by 7920% and 530.78% in terms of EM@1 and GLEU respectively. Compared to training models with only original titles, QETRA can improve the performance by 32.78% and 2.38% in terms of EM@1 and GLEU respectively. Based on the above results, we can find the information from the original title and the question body have a specific complementary. That means using both input modalities can help to achieve the best performance for QETRA. Moreover, for these two kinds of input modalities, we find that the original title can make more contributions than the question body. This also shows that reformulating on the basis of the original title can better guarantee the quality of the title.

Answer to RQ2: Original title can provide more valuable information than the question body for the question title reformulation task. Moreover, considering two modalities together can help to achieve the best performance of QETRA.

C. Result Analysis for RQ3

RQ3: What is the effect of using multi-task learning in QETRA?

To investigate the effect of considering multi-task learning in QETRA, in this RQ, we separately perform model training for different programming languages and use QETRA_{sep} to denote this control approach. For each programming language, we compare QETRA_{sep} with QETRA on the same testing set related to this programming language. The comparison results are shown in Table IX. In this table, we can find that the performance of the models based on multi-task learning can outperform that of the models trained for each programming language separately in most cases. For example, in terms of EM@1, compared to QETRA_{sep} on Python, C#, Java, JavaScript, HTML and PHP, QETRA can improve the performance by 19.70%, 6.42%, 13.18%, 11.30%, 43.97% and 34.79%, respectively. We notice for some programming languages with sufficient training data (such as Java and JavaScript), training the model separately may lead to slightly worse results in some performance measures. The potential reason is that the large size of the training data may enable the model to better capture the specific features and structures of these languages. However, we find using multi-task learning can bring more performance improvements for low-resource programming languages (i.e., HTML and PHP). The reason is that only using the training data of the low-resource programming language can only learn a few pieces of information while using multi-task learning can help to learn more useful information from other programming languages with sufficient training data. We will conduct more experiments to verify further the generalization of QETRA for other programming languages in Section VII. In summary, considering multi-task learning in QETRA can guarantee the generalization of our approach by learning multiple tasks simultaneously.

TABLE IX
COMPARISON OF QETRA WITH QETRA_{seq} FOR SIX PROGRAMMING LANGUAGES

Language	Approach	BIEU_4(%)	ROUGE_L(%)	CIDEr	EM@1(%)	EM@5(%)	EM@10(%)	GLEU(%)	MRR(%)
Python	QETRA _{seq}	47.89	63.84	4.51	11.13	21.49	25.71	47.59	22.75
	QETRA	50.25	65.01	4.72	13.32	25.91	30.06	49.52	25.12
C#	QETRA _{seq}	53.55	67.71	4.97	17.40	29.69	33.68	52.27	29.74
	QETRA	54.22	68.05	5.04	18.51	31.38	34.76	52.89	31.42
Java	QETRA _{seq}	48.87	64.24	4.54	12.92	24.86	29.19	48.10	24.90
	QETRA	49.01	63.89	4.54	14.62	25.83	28.26	48.01	25.41
JavaScript	QETRA _{seq}	50.52	64.38	4.60	13.22	25.27	29.16	48.56	23.58
	QETRA	49.54	64.84	4.65	14.71	25.58	29.55	48.99	26.25
HTML	QETRA _{seq}	46.90	62.45	4.29	10.30	21.97	25.38	45.72	20.03
	QETRA	50.32	65.36	4.68	14.83	26.77	29.31	49.50	22.82
PHP	QETRA _{seq}	48.35	63.89	4.49	11.36	23.44	27.06	47.52	23.87
	QETRA	50.88	65.20	4.72	15.31	26.00	30.13	49.69	25.73

Original Title: PyQt5: How to detect the type of widget?		Ground Truth: How to detect the type of widget?	
Problem Description: This should be a stupid question. I am just curious and could not find the answer on my own. E.g. I define in PyQt5 some widgets: Is there any method to detect what kind of widget the self.lbl, self.btn, or self.txt are? I could imagine: by detecting the widget type, the input is self.lbl, the output should be QLabel... Or something like this. I have only found the isWidgetType() method. But it is not what I want to have.			
Code Snippet: <pre>self.lbl = QLabel("label") self.btn = QPushButton("click") self.txt = QLineEdit("text")</pre>			
Title 1: How to detect the type of widget in PyQt5? Improved <input type="radio"/> Y <input type="radio"/> N Similarity <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 Naturalness <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 Informativeness <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4	Title 2: PyQt5: How to detect the type of widget? Improved <input type="radio"/> Y <input type="radio"/> N Similarity <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 Naturalness <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 Informativeness <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4	Title 3: How to detect the type of widget? Improved <input type="radio"/> Y <input type="radio"/> N Similarity <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 Naturalness <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 Informativeness <input type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4	

Fig. 7. A questionnaire for a sample in our human study.

Answer to RQ3: Applying multi-task learning can help to improve the performance of QETRA, especially for questions related to low-resource programming languages.

D. Result Analysis for RQ4

RQ4: Can our proposed approach QETRA generate higher-quality reformulated titles than the state-of-the-art baselines by human study?

Automatic evaluation metrics can evaluate the quality of the reformulated titles based on the overlap degree with the ground-truth titles. However, automatic evaluation cannot reflect the semantic similarity between the reformulated titles and the ground-truth titles in a realistic way. Since BiLSTM-CC and LanguageTool can achieve the best performance in our considered two types of baselines respectively by automatic evaluation, we perform a human study to further evaluate the quality of the titles reformulated by BiLSTM-CC, LanguageTool, and QETRA.

In our human study, based on the characteristics of the title reformulation task, we first investigated whether the quality of the reformulated titles was improved over the original titles, and then we follow the methodology considered by the previous studies [54]. As shown in Fig. 7, we measure the quality of the reformulated titles in terms of three aspects:

- **Similarity.** This aspect measures the similarity between reformulated titles and ground-truth titles.
- **Naturalness.** This aspect measures the grammaticality and fluency of the reformulated titles.
- **Informativeness.** This aspect measures the amount of content carried over from the input body of the post to the reformulated titles, ignoring the fluency of the text.

We recruited six master students, who had more than five years of project development and were familiar with the usage of Stack Overflow. Then we randomly selected 20 samples for each programming language (i.e., 120 samples in total) in the testing set. For each sample, we gathered the ground truth and three reformulated titles generated by BiLSTM-CC, LanguageTool, and QETRA respectively. Later we divided 120 samples into three groups. Each group has 40 samples and is evaluated

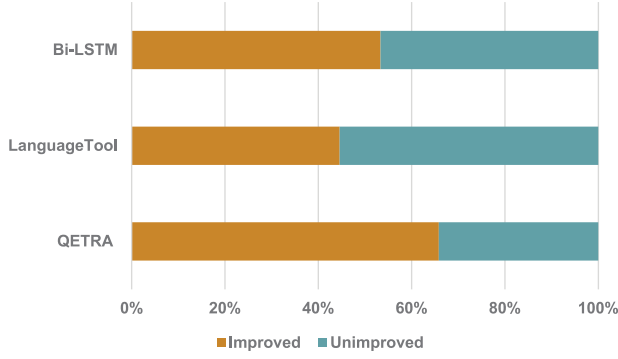


Fig. 8. Comparison of the reformulated titles with the original titles.

TABLE X
THE AVERAGE SCORE VALUE AND STANDARD DEVIATION (SHOWN IN PARENTHESES) OF OUR HUMAN STUDY RESULTS

Approach	Similarity	Naturalness	Informativeness
BiLSTM-CC	2.596 (0.991)	2.783 (0.963)	2.586 (1.002)
LanguageTool	2.792 (0.961)	3.038 (0.743)	2.762 (0.918)
QETRA	3.175 (0.72)	3.375 (0.592)	3.242 (0.713)

by two master students in terms of the above-mentioned three aspects. The value range of the score is between 0 and 4. The higher the value, the higher the quality of the reformulated title. Since a reformulated title will be evaluated twice for two students, we gather the average score value. A questionnaire for a sample can be found in Fig. 7. In this questionnaire, the reformulated titles are randomly ordered to ensure that master students do not know which approach the reformulated title is generated by. Moreover, the master students are free to use the Internet to look for related concepts that they are unfamiliar with. Finally, to guarantee the quality of our human study, we require master students to evaluate only 20 samples in half a day.

The results of our human study are shown in Fig. 8 and Table X. Fig. 8 illustrates a quality comparison between the reformulated titles and the original titles by the three methods, where "Improved" indicates that the reformulated title was rated by reviewers as having better quality than the original one, while "Unimproved" indicates the opposite. We observe that QETRA achieved a higher proportion of "Improved" compared to BiLSTM-CC and LanguageTool. To check whether this difference is statistically significant, we performed Fisher's exact test [55] and found the difference is statistically significant (p -value < 0.05). Moreover, Table X shows the average score value and standard deviation of all the samples when considering the similarity, naturalness, and informativeness of the generated reformulated titles. In this table, we can find QETRA outperforms BiLSTM-CC and LanguageTool in three perspectives, which can further verify the effectiveness of QETRA. Finally, we use Fleiss Kappa [56] to measure the agreement among these six students. The overall Kappa value based on the comparison results is 0.725, which indicates substantial agreement among these students.

TABLE XI
STATISTICAL INFORMATION OF OUR GATHERED DATASET FOR THE PROGRAMMING LANGUAGES GO AND RUBY

Language	Training		Validation		Testing	
	Pair	Thread	Pair	Thread	Pair	Thread
Go	1,878	1,511	231	189	246	190
Ruby	3,980	3,312	494	414	501	415
Total	5,858	4,823	725	603	747	605

Answer to RQ4: Our human study shows that QETRA can generate higher quality titles than the baselines BiLSTM-CC and LanguageTool in terms of similarity, naturalness, and informativeness.

VII. DISCUSSION

A. How Effective is QETRA for the Questions Related to Other Programming Languages

In this subsection, we want to evaluate the generalization of QETRA on posts related to other programming languages (i.e., Ruby and Go). To answer this RQ, we first gather the datasets by following the dataset gathering method introduced in Section II. The detailed statistical information for these two programming languages can be found in Table XI. In particular, for Go, we identified 1,890 title reformulation threads. From these threads, we extracted 2,355 title reformulation pairs. For Ruby, we identified 1,161 title reformulation threads. From these threads, we extracted 4,975 title reformulation pairs.

In this experiment, for QETRA, we directly used the trained model based on our previously considered six programming languages. For baselines, we trained the models based on the dataset related to Go or Ruby respectively. The comparison results can be found in Table XII. When considering Ruby, QETRA can achieve 10.38 and 46.82 in terms of EM@1 and GLEU. When considering Go, QETRA can achieve 18.70 and 52.73 in terms of EM@1 and GLEU. Taking the Go programming language as an example, QETRA can improve the performance by 8.24%, 318.18%, and 10.18% for BIEU_4, EM@1, and GLEU metrics respectively after comparing to LanguageTool, which can achieve the best performance in all the baselines. These results verify the generalization of the effectiveness of QETRA for posts related to other new programming languages.

B. Whether Using CodeT5 Can Improve the Performance of QETRA?

In this subsection, we consider CodeT5 [57] as the backbone model for our proposed approach QETRA and investigate whether this setting can improve the performance of QETRA. The comparison results can be found in Table XIII. In this table, we can find using T5 [17] can achieve better performance than using CodeT5 for QETRA. As discussed in Section VI-B,

TABLE XII
COMPARISON RESULTS BETWEEN QETRA AND BASELINES FOR POSTS RELATED TO OTHER NEW PROGRAMMING LANGUAGES (I.E., RUBY AND GO)

Language	Approach	BLEU_4(%)	ROUGE_L(%)	CIDEr	EM@1(%)	EM@5(%)	EM@10(%)	GLEU(%)	MRR(%)
Ruby	NMT	6.43	24.68	0.48	0.00	0.00	0.00	10.16	0.00
	Transformer	4.42	17.25	0.30	0.00	0.00	0.00	6.84	0.00
	BiLSTM-CC	32.68	50.20	2.92	1.40	2.99	3.99	32.12	2.60
	BART	39.50	53.49	3.56	0.00	0.00	0.40	37.51	15.69
	CodeBERT	7.36	23.38	0.61	0.00	0.00	0.00	10.24	0.00
	LanguageTool	46.05	60.52	4.06	4.59	—	—	43.24	—
	QETRA	48.38	62.92	4.45	10.38	22.55	27.15	46.82	21.70
Go	NMT	2.16	19.26	0.28	0.00	0.00	0.00	7.08	0.00
	Transformer	4.03	19.44	0.34	0.00	0.00	0.00	7.78	0.00
	BiLSTM-CC	42.73	59.00	3.97	4.88	9.35	10.57	41.71	7.41
	BART	47.42	62.47	4.38	0.00	1.63	4.07	46.20	24.98
	CodeBERT	1.86	14.84	0.21	0.00	0.00	0.00	5.46	0.00
	LanguageTool	50.45	66.84	4.61	4.47	—	—	47.86	—
	QETRA	54.60	67.55	5.12	18.70	35.77	42.28	52.73	28.60

TABLE XIII
COMPARISON RESULTS BETWEEN OUR PROPOSED APPROACH AND CODET5

Approach	BLEU_4(%)	ROUGE_L(%)	EM@1(%)	GLEU(%)
CodeT5	48.03	61.99	6.91	45.44
QETRA	52.29	66.42	16.74	51.14

the original title can provide more valuable information than the question body for the question title reformulation task. Moreover, in our previous study on Stack Overflow question title generation [13], the problem description can play a more important role in question title generation than the code snippet. Therefore whether we can learn high-quality semantic information from the text is more important to improve the performance of QETRA. Since T5 [17] is a general pre-trained model for various natural language processing tasks, while CodeT5 [57] is a specific pre-trained model for source code understanding and generation tasks, this is a possible reason for the effectiveness of using T5 in QETRA.

C. Statistically Significant Analysis with Different Dataset Splits

To investigate whether QETRA can statistically significantly outperform baselines with different dataset splits, we randomly split the dataset with different random seeds. Due to the high computational cost in model training with deep learning, we only split the dataset ten times. In this subsection, we consider four competitive baselines (i.e., BiLSTM-CC, BART, CodeBERT, and LanguageTool). The final comparison results are shown in Table XIV. In this table, we show the average value for these ten independent running results. Based on these results, we can still find that QETRA can improve the performance by 36.82%, 22.77%, 42.07%, 344.25%, and 36.49% in terms of BLEU_4, ROUGE_L, CIDEr, EM@1, and GLEU when compared to these baselines on average. Moreover, we perform the Wilcoxon signed-rank tests [52] at the confidence level of 95% to check whether the performance differences between QETRA and these baselines are significant. Final results show that QETRA statistically significantly outperforms these three baselines (i.e., p -value < 0.05). Finally, we further use Cliff's

TABLE XIV
COMPARISON RESULTS WITH DIFFERENT DATASET SPLITS

Approach	BLEU_4(%)	ROUGE_L(%)	CIDEr	EM@1(%)	GLEU(%)
BiLSTM-CC	43.72	58.81	3.88	3.74	42.30
BART	35.34	47.20	3.16	0.75	33.61
CodeBERT	27.27	48.27	2.50	3.31	29.08
LanguageTool	46.55	62.11	4.17	7.27	44.88
QETRA	52.29	66.42	4.87	16.74	51.14

TABLE XV
TRAINING AND INFERENCE TIME FOR DIFFERENT APPROACHES

Approach	Training Time (Hours)	Inference Time (Milliseconds/Post)
NMT	4.20	21.23
Transformer	3.81	15.19
Code2Que	5.12	28.12
BART	18.91	30.34
CodeBERT	20.23	36.27
LanguageTool	—	32.25
QETRA	18.20	34.24

delta [53] to quantify the amount of performance difference between different approaches. The results show that QETRA can outperform the next best baseline (i.e., LanguageTool) with a large improvement with respect to Cliff's delta. For example, in terms of BLEU_4, the value of Cliff's delta is at least 0.80 for different programming languages.

D. Time Efficiency

Table XV shows the time costs of model training and average inference for each question title reformulation. Notice LanguageTool is rule-based and does not need model training. Since other approaches are all deep learning-based, therefore, the model training cost is high (ranging from 18.20 hours to 20.23 hours). However, once models have been trained, it only takes a few milliseconds to generate the reformulated question title for the original title.

E. Challenging Types of Question Title Reformulation for QETRA

After automatic evaluation and human study, we can find QETRA can achieve the best performance. However, we also

notice that QETRA may generate the reformulated titles, which have low similarity with the ground-truth titles, in some cases. Due to the huge cost of manually analyzing all the failed cases, we use a simple random sample approach without replacement. Specifically, we randomly sampled 100 cases of this type and analyzed these cases in a manual way, which can help to identify the challenging types of question title reformulation for QETRA.

The first challenge type is the title reformulation due to different expression styles (such as from question sentence to declarative sentence, or from declarative sentence to question sentence.). For example, given the original title *“How do you select between two dates with Django,”* QETRA generates the reformulated title as *“How do you select between two dates with Django?”* However, the ground truth is *“Select between two dates with Django.”* Similarly, given the original title *“an expression for an infinite generator?”*, QETRA generates the reformulated title as *“An expression for an infinite generator?”* However, the ground truth is *“Is there an expression for an infinite iterator?”* Although the generated title and the ground truth have the same semantics, the different expression styles can result in a low score in terms of automatic evaluation metrics. A possible solution is to propose effective methods to unify expression styles for question titles.

The second challenge type is that the reformulated title and the ground truth have the same semantics but are expressed in different ways. For example, given the original title *“why builtins both module and dict”*, QETRA generates the reformulated title as *“Why is builtins both module and dict?”* However, the ground truth is *“why builtins is both module and dict.”* Similarly, given the original title *“Python—why compile?”* QETRA generates the reformulated title as *“Why would you compile a Python script?”* However, the ground truth is *“Why compile Python code?”* This kind of type can result in a low score in terms of previous automatic evaluation metrics and is also a challenging problem for evaluating source code summarization approaches [58]. A possible solution is to design semantic-based evaluation metrics, which can better evaluate the semantic similarity between the reformulated title and the ground truth.

The third challenge type is when the original title contains a tag (such as programming language) for search engine optimization purposes, the reformulated title and ground truth may be improved in different ways. For example, given the original title *“Python: Searching/reading binary data”*, QETRA generates the reformulated title as *“Searching/reading binary data.”* However, the ground truth is *“Searching/reading binary data in Python.”* Similarly, given the original title *“How to convert escaped characters in Python?”* QETRA generates the reformulated title as *“How to convert escaped characters in Python?”* However, the ground truth is *“How to convert escaped characters?”* The question title writing guidelines¹³ provided by Stack Overflow state that when the original title contains a tag, the tag can be removed directly or placed at the end of the sentence in regular English. Although the generated title and the ground truth are both officially recommended,

differences in improvement can result in a low score in terms of automatic evaluation metrics. A possible solution is to enhance the used evaluation metrics to handle this special case.

F. Threats to Validity

In this section, we mainly analyze the potential threats to the validity of our empirical study.

Threats to Internal Validity. The first internal threat is the potential faults in our approach’s implementation. To alleviate this threat, we carefully examined our implementation and used third-party mature libraries (such as PyTorch and transformers). The second internal threat is the implementation correctness of our considered baselines. To alleviate this threat, if the baselines shared the scripts, we directly used their shared scripts. Otherwise, we implemented these baselines according to their original descriptions.

Threats to External Validity. The main external threat is the datasets gathered by our study. To alleviate this threat, we gathered modification logs from recent data dumps from Stack Overflow. Then we select posts and related modification records from the six most popular programming languages (i.e., Java, Python, C#, JavaScript, PHP, HTML). Finally, we also verify the generalization of our proposed approach to two other low-resource programming languages (i.e., Ruby and Go).

Threats to Conclusion Validity. The first conclusion threat is the data leakage problem that existed in the dataset split process. To alleviate this threat, we split the dataset in terms of thread granularity. If we split the dataset in terms of pair granularity, the title reformulation pairs in the same thread may exist in different split parts. Since these pairs share the same post body, this split strategy may cause a data leakage problem. The second conclusion threat is that we assume the quality of the title t_n is highest in Section II. Users usually tend to reformulate their question post titles repeatedly to make them more accurate, concise, and clear. Therefore, it is common for the last version of a question title written by a user to be the best [14], [15]. However, we admit that there are exceptions, such as when a user may accidentally make the title worse, or when a title is already good from the start and does not need reformulation. In these cases, the last version of the title may not necessarily be the best. In the future, we can design more heuristic rules or manually analyze these titles to identify these exceptions to alleviate this threat.

Threats to Construct Validity. The main construct threat is related to evaluation metrics used in our automated evaluation. Since we treat the question title reformulation as a neural machine translation problem, we first consider evaluation metrics based on term overlap (such as BLEU, ROUGE-L, CIDEr), which were also used in previous studies on similar tasks on source code understanding [59], [4], [60], [13], [61]. Moreover, we also consider two evaluation metrics (such as Exact Match and GLEU) used in grammatical error correction by considering the characteristics of our studied problem. Finally, we consider information retrieval-based metrics (such as MRR), which can evaluate the practicability of QETRA in Stack Overflow post retrieval. Except for automatic evaluation, we also conducted a

¹³<https://stackoverflow.com/help/how-to-ask>

human study to further verify the effectiveness of our proposed approach, which is suggested in previous studies on similar software engineering tasks (such as source code summarization [33], [54]).

VIII. RELATED WORK

In this section, we first summarize related studies on post title quality assurance for Stack Overflow. Then we summarize related studies on query reformulation for software engineering. Finally, we emphasize the novelty of our study.

A. Post Title Quality Assurance for Stack Overflow

Stack Overflow is a popular questions and answers website for software developers. Quality assurance of content is crucial to maintaining a good user experience. Ponzanelli et al. [1] identified low-quality posts by analyzing both the content of the post and community-related aspects. Yao et al. [62] studies the relationship between the voting scores of questions and answers in Stack Overflow. Calefato et al. [22] provided guidelines for writing high-quality questions on Stack Overflow that developers can follow to increase the chance of getting technical help. Chen et al. [15] proposed an approach based on the convolutional neural network to learn mid-level editing patterns from historical post edits for predicting the need of editing a post. Wang et al. [14] investigated whether revision-related badges have a negative impact on the quality of revisions. Zhang et al. [63] investigated how the knowledge in answers becomes obsolete and then identified the characteristics of these obsolete answers. Liu et al. [64] investigated the broken links on Stack Overflow. Zhu et al. [65] conducted empirical studies on question discussions on Stack Overflow.

To our best knowledge, Gao et al. [4] were the first to propose a data-driven approach to generate question titles for a code snippet. This approach used an attention mechanism to perform content selection, a copy mechanism to handle the OOV problem, and a coverage mechanism to alleviate the word repetition problem. Liu et al. [13] proposed an approach SOTitle by leveraging the code snippets and the problem description. Then they formalized post title generation for each programming language as separate but related tasks and utilize multitask learning. Zhang et al. [66] also used the code snippets and the problem description. Their approach used CodeBERT to encode the question body into hidden representations, a decoder based on Transformer to generate post titles, and a copy attention layer to further refine the output distribution of the generated titles.

B. Query Reformulation for Software Engineering

Query reformulation plays an important role in many software engineering tasks (such as concept location, code search, and bug localization) [67].

In the concept location task, developers localized the source code to find the location to start the change. Gay et al. [68] performed query reformulation based on explicit relevance feedback. They expanded the initial query by marking the search results of the initial query as relevant or irrelevant. Hill et al.

[69] proposed a contextual search approach, which can quickly identify alternative words for query reformulation. Yang and Tan [70] discovered semantically related words by leveraging the context of words in comments and code in code bases. Sisman and Kak [71] proposed the framework, which can enrich a developer's query with specific additional terms. These specific additional terms were drawn from the highest-ranked artifacts retrieved by the initial query. Howard et al. [72] mined semantically similar words in the software context. Haiduc et al. [73] proposed an automated approach Refoqus based on machine learning. This approach trained a reformulation strategy recommendation model based on a sample of queries and relevant results. Rahman and Roy [74] proposed STRICT by identifying suitable search terms. They utilized two information retrieval techniques (i.e., TextRank and POSRank) to determine the term's importance based on both its co-occurrences and syntactic relationships with other important terms.

Existing code search methods require developers to provide high-quality queries to find relevant code snippets. Query reformulation is an effective method to solve this problem. Wang et al. [75] took the developers' opinions on the results from the code search engine as feedback. Then they used this feedback to reorder the code search results. Lu et al. [76] identified each term in the original query and extended this query with synonyms generated from WordNet [77]. Nie et al. [78] proposed the approach QECK to generate the expansion queries, which can identify software-specific expansion words by mining Stack Overflow. Rahman et al. [79], [80] exploited keyword-API associations from the crowdsourced knowledge of Stack Overflow. Rahman and Roy [81] proposed a novel technique that automatically identifies relevant API classes for a programming task written as a natural language query, and then reformulates the query using these API classes. Cao et al. [82] performed automated query reformulation based on query logs from Stack Overflow. Gao et al. [83] proposed a fully data-driven approach Que2Code to recommend the best code snippet in Stack Overflow by generating paraphrase questions for the input query.

Information retrieval (IR)-based bug localization relies on formulating an initial query based on the full text of a bug report. Chaparro et al. [84], [85] proposed and evaluated a set of query reformulation strategies by considering the selection of existing information in bug descriptions, and the removal of irrelevant parts from the original query. Rahman and Roy [86] proposed a novel technique BLIZZARD. This technique determined whether there exist excessive program entities in a bug report (i.e., query), and then applied appropriate query reformulations for bug localization. Florez et al. [87] investigated the benefits of different query reduction and query expansion strategies for bug localization and designed the approach QREX based on the most effective strategy.

C. Novelty of Our Study

Previous studies [4], [13], [66] mainly focused on automatically generating the question titles by analyzing the contents in the post bodies (such as problem descriptions and code

snippets). Different from previous studies, we consider a new scenario for improving question title quality. In this scenario, we can use our proposed approach QETRA to polish the original title by mining modification logs, which can better guarantee the quality of the Stack Overflow community. Moreover, our proposed approach QETRA is applicable not only to post owners but also to other experts (such as community moderators). Due to the characteristics of this new scenario, our proposed approach further considers the original question title when compared to the previously proposed approaches [4], [13], [66]. Therefore, the quality of generated titles may be improved and our experimental results also confirm this conjecture. Finally, our study and the previous studies [4], [13], [66] are not competitive work, but rather complementary. For example, we can use the previous studies (such as SOTitle [13]) to generate a draft title and then use QETRA to further polish the draft title.

IX. CONCLUSION

Writing a high-quality title that clarifies and summarizes the critical problems in the post is a challenging task for developers, especially for novices who lack domain knowledge or poor writing skills. To the best of our knowledge, we are the first to study the question title reformulation problem in this study. We first conducted an informative study to investigate the title reformulation patterns by mining modification logs provided by Stack Overflow. Motivated by the findings in our informative study, we propose a novel approach QETRA based on the pre-trained model T5 and multi-task learning and develop a plug-in to facilitate the use of developers. Based on our gathered datasets, we evaluate the effectiveness of QETRA by both automatic evaluation and human study. Finally, we also verify the generalization of QETRA for posts related to other new programming languages and the challenging types of question title reformulation for QETRA.

To further improve the performance of QETRA, we first want to consider more information from the posts (such as tags, and discussions). We second want to identify and remove the noises during the dataset gathering process as studied in neural code search [88]. We third want to consider more programming languages when training our models.

REFERENCES

- [1] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton, "Improving low quality stack overflow post detection," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Piscataway, NJ, USA: IEEE Press, 2014, pp. 541–544.
- [2] F. Calefato, F. Lanubile, M. C. Marasciulo, and N. Novielli, "Mining successful answers in stack overflow," in *Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 430–433.
- [3] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec, "Discovering value from community activity on focused question answering sites: A case study of stack overflow," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 850–858.
- [4] Z. Gao, X. Xia, J. Grundy, D. Lo, and Y.-F. Li, "Generating question titles for stack overflow from mined code snippets," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 4, pp. 1–37, 2020.
- [5] J. Trienes and K. Balog, "Identifying unclear questions in community question answering websites," in *Proc. 41st Eur. Conf. Inf. Retrieval (ECIR)*. Cologne, Germany: Springer, 2019, pp. 276–289.
- [6] S. Mondal, C. K. Saifullah, A. Bhattacharjee, M. M. Rahman, and C. K. Roy, "Early detection and guidelines to improve unanswered questions on stack overflow," in *Proc. 14th Innov. Softw. Eng. Conf. (formerly India Soft. Eng. Conf.)*, 2021, pp. 1–11.
- [7] M. Allamanis and C. Sutton, "Why, when, and what: Analyzing stack overflow questions by topic, type, and code," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, Piscataway, NJ, USA: IEEE Press, 2013, pp. 53–56.
- [8] R. Rubei, C. Di Sipio, P. T. Nguyen, J. Di Rocco, and D. Di Ruscio, "PostFinder: Mining stack overflow posts to support software developers," *Inf. Softw. Technol.*, vol. 127, Nov. 2020, Art. no. 106367.
- [9] G. Chen, C. Chen, Z. Xing, and B. Xu, "Learning a dual-language vector space for domain-specific cross-lingual question retrieval," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 744–755.
- [10] A. Abdellatif, D. Costa, K. Badran, R. Abdalkareem, and E. Shihab, "Challenges in chatbot development: A study of stack overflow posts," in *Proc. 17th Int. Conf. Mining Softw. Repositories*, 2020, pp. 174–185.
- [11] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "API method recommendation without worrying about the task-API knowledge gap," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 293–304.
- [12] M. Wei, N. S. Harzevili, Y. Huang, J. Wang, and S. Wang, "CLEAR: Contrastive learning for API recommendation," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 376–387.
- [13] K. Liu, G. Yang, X. Chen, and C. Yu, "SOTitle: A transformer-based post title generation approach for stack overflow," in *Proc. 29th IEEE Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, 2022, pp. 577–588.
- [14] S. Wang, T.-H. Chen, and A. E. Hassan, "How do users revise answers on technical Q&A websites? A case study on stack overflow," *IEEE Trans. Softw. Eng.*, vol. 46, no. 9, pp. 1024–1038, Sep. 2020.
- [15] C. Chen, X. Chen, J. Sun, Z. Xing, and G. Li, "Data-driven proactive policy assurance of post quality in community Q&A sites," *Proc. ACM Human-Comput. Interact.*, vol. 2, no. CSCW, pp. 1–22, 2018.
- [16] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 12, pp. 5586–5609, Dec. 2022.
- [17] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 5485–5551, 2020.
- [18] Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," in *Proc. Findings Assoc. Comput. Linguistics (EMNLP)*, 2020, pp. 1536–1547.
- [19] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 112–121.
- [20] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 510–520.
- [21] A. Y. Chua and S. Banerjee, "Answers or no answers: Studying question answerability in Stack Overflow," *J. Inf. Sci.*, vol. 41, no. 5, pp. 720–731, 2015.
- [22] F. Calefato, F. Lanubile, and N. Novielli, "How to ask for technical help? Evidence-based guidelines for writing questions on Stack Overflow," *Inf. Softw. Technol.*, vol. 94, pp. 186–207, Feb. 2018.
- [23] A. Baltadzhieva and G. Chrupala, "Predicting the quality of questions on stackoverflow," in *Proc. Int. Conf. Recent Adv. Natural Lang. Process.*, 2015, pp. 32–40.
- [24] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," in *Proc. 7th Int. Symp. String Process. Inf. Retrieval (SPIRE)*, Piscataway, NJ, USA: IEEE Press, 2000, pp. 39–48.
- [25] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.
- [26] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *Proc. Conf. Empirical Methods Natural Lang. Process., Syst. Demonstrations*, 2018, pp. 66–71.
- [27] A. Vaswani et al., "Attention is all you need," in *Proc. 30th Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
- [28] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Austin, TX, USA: Association for Computational Linguistics, 2016, pp. 551–561.

- [29] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.
- [30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 770–778.
- [31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [32] X. Liu, P. He, W. Chen, and J. Gao, "Multi-task deep neural networks for natural language understanding," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 4487–4496.
- [33] S. Jiang, A. Armaly, and C. Mcmillan, "Automatically generating commit messages from diffs using neural machine translation," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 135–146.
- [34] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "A transformer-based approach for source code summarization," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 4998–5007.
- [35] Z. Yang et al., "A multi-modal transformer-based code summarization approach for smart contracts," in *Proc. IEEE/ACM 29th Int. Conf. Program Comprehension (ICPC)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 1–12.
- [36] G. Yang et al., "ComFormer: Code comment generation via transformer and fusion method-based hybrid code representation," in *Proc. 8th Int. Conf. Dependable Syst. Their Appl. (DSA)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 30–41.
- [37] G. Yang, Y. Zhou, X. Chen, and C. Yu, "Fine-grained pseudo-code generation method via code feature extraction and transformer," in *Proc. 28th Asia-Pacific Softw. Eng. Conf. (APSEC)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 213–222.
- [38] G. Yang, X. Chen, Y. Zhou, and C. Yu, "DualSC: Automatic generation and summarization of shellcode via transformer and dual learning," in *Proc. 29th IEEE Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, 2022, pp. 361–372.
- [39] M. Lewis et al., "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 7871–7880.
- [40] T. Zhang, I. C. Irsan, F. Thung, D. Han, D. Lo, and L. Jiang, "Automatic pull request title generation," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 71–81.
- [41] G. Yang, K. Liu, X. Chen, Y. Zhou, C. Yu, and H. Lin, "CCGIR: Information retrieval-based code comment generation method for smart contracts," *Knowl.-Based Syst.*, vol. 237, Feb. 2022, Art. no. 107858.
- [42] C. Yu, G. Yang, X. Chen, K. Liu, and Y. Zhou, "BashExplainer: Retrieval-augmented bash code comment generation based on fine-tuned CodeBERT," in *Proc. 38th Int. Conf. Softw. Maintenance Evol. (ICSME)*, 2022, pp. 82–93.
- [43] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proc. 40th Annu. Meeting Assoc. Comput. Linguistics*, 2002, pp. 311–318.
- [44] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Proc. Workshop Text Summarization Branches Out*, 2004, pp. 74–81.
- [45] R. Vedantam, C. L. Zitnick, and D. Parikh, "CIDER: Consensus-based image description evaluation," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 4566–4575.
- [46] C. Napoles, K. Sakaguchi, M. Post, and J. Tetreault, "GLEU without tuning," 2016, *arXiv:1605.02592*.
- [47] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
- [48] L. Prechelt, "Early stopping—But when?" in *Neural Networks: Tricks of the Trade*. Berlin, Heidelberg: Springer, 1998, pp. 55–69.
- [49] J. Von Der Mosel, A. Trautsch, and S. Herbold, "On the validity of pre-trained transformers for natural language processing in the software engineering domain," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 1487–1507, Apr. 2023.
- [50] X. Hu, Z. Gao, X. Xia, D. Lo, and X. Yang, "Automating user notice generation for smart contract functions," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 5–17.
- [51] G. Yang, Y. Zhou, X. Chen, X. Zhang, T. Han, and T. Chen, "ExploitGen: Template-augmented exploit code generation based on CodeBERT," *J. Syst. Softw.*, vol. 197, Mar. 2023, Art. no. 111577.
- [52] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in Statistics: Methodology and Distribution*. New York, NY, USA: Springer, 1992, pp. 196–202.
- [53] N. Cliff, *Ordinal Methods for Behavioral Data Analysis*. New York, NY, USA: Psychology Press, 2014.
- [54] B. Wei, Y. Li, G. Li, X. Xia, and Z. Jin, "Retrieve and refine: Exemplar-based neural comment generation," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 349–360.
- [55] R. A. Fisher, "On the interpretation of χ^2 from contingency tables, and the calculation of P," *J. Roy. Statistical Soc.*, vol. 85, no. 1, pp. 87–94, 1922.
- [56] J. L. Fleiss, "Measuring nominal scale agreement among many raters," *Psychol. Bull.*, vol. 76, no. 5, pp. 378–382, 1971.
- [57] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2021, pp. 8696–8708.
- [58] S. Haque, Z. Eberhart, A. Bansal, and C. Mcmillan, "Semantic similarity metrics for evaluating source code summarization," in *Proc. 30th IEEE/ACM Int. Conf. Program Comprehension*, 2022, pp. 36–47.
- [59] Z. Li et al., "SeTransformer: A transformer-based code semantic parser for code comment generation," *IEEE Trans. Rel.*, vol. 72, no. 1, pp. 258–273, Mar. 2023.
- [60] Z. Li et al., "SeCNN: A semantic CNN parser for code comment generation," *J. Syst. Softw.*, vol. 181, Nov. 2021, Art. no. 111036.
- [61] H. Lin et al., "Gen-FL: Quality prediction-based filter for automated issue title generation," *J. Syst. Softw.*, vol. 195, Jan. 2023, Art. no. 111513.
- [62] Y. Yao, H. Tong, T. Xie, L. Akoglu, F. Xu, and J. Lu, "Detecting high-quality posts in community question answering sites," *J. Inf. Sci.*, vol. 302, pp. 70–82, May 2015.
- [63] H. Zhang, S. Wang, T.-H. Chen, Y. Zou, and A. E. Hassan, "An empirical study of obsolete answers on stack overflow," *IEEE Trans. Softw. Eng.*, vol. 47, no. 4, pp. 850–862, Apr. 2021.
- [64] J. Liu et al., "Broken external links on stack overflow," *IEEE Trans. Softw. Eng.*, vol. 48, no. 9, pp. 3242–3267, Sep. 2022.
- [65] W. Zhu, H. Zhang, A. E. Hassan, and M. W. Godfrey, "An empirical study of question discussions on stack overflow," *Empirical Softw. Eng.*, vol. 27, no. 6, 2022, Art. no. 148.
- [66] F. Zhang et al., "Improving Stack Overflow question title generation with copying enhanced CodeBERT model and bi-modal information," *Inf. Softw. Technol.*, vol. 148, Aug. 2022, Art. no. 106922.
- [67] M. M. Rahman and C. K. Roy, "A systematic literature review of automated query reformulations in source code search," 2021, *arXiv:2108.09646*.
- [68] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in IR-based concept location," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Piscataway, NJ, USA: IEEE Press, 2009, pp. 351–360.
- [69] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of NL-queries for software maintenance and reuse," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, Piscataway, NJ, USA: IEEE Press, 2009, pp. 232–242.
- [70] J. Yang and L. Tan, "Inferring semantically related words from software context," in *Proc. 9th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, Piscataway, NJ, USA: IEEE Press, 2012, pp. 161–170.
- [71] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, Piscataway, NJ, USA: IEEE Press, 2013, pp. 309–318.
- [72] M. J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker, "Automatically mining software-based, semantically-similar words from comment-code mappings," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, Piscataway, NJ, USA: IEEE Press, 2013, pp. 377–386.
- [73] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. D. Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2013, pp. 842–851.
- [74] M. M. Rahman and C. K. Roy, "Strict: Information retrieval based search term identification for concept location," in *Proc. IEEE 24th Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 79–90.

- [75] S. Wang, D. Lo, and L. Jiang, "Active code search: Incorporating user feedback to improve code search relevance," in *Proc. 29th ACM/IEEE Int. Conf. Automated Softw. Eng.*, 2014, pp. 677–682.
- [76] M. Lu, X. Sun, S. Wang, D. Lo, and Y. Duan, "Query expansion via WordNet for effective code search," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reengineering (SANER)*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 545–549.
- [77] G. A. Miller, "WordNet: A lexical database for English," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.
- [78] L. Nie, H. Jiang, Z. Ren, Z. Sun, and X. Li, "Query expansion based on crowd knowledge for code search," *IEEE Trans. Services Comput.*, vol. 9, no. 5, pp. 771–783, Sep./Oct. 2016.
- [79] M. M. Rahman, C. K. Roy, and D. Lo, "RACK: Automatic API recommendation using crowdsourced knowledge," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reengineering (SANER)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 349–359.
- [80] M. M. Rahman, C. K. Roy, and D. Lo, "Automatic query reformulation for code search using crowdsourced knowledge," *Empirical Softw. Eng.*, vol. 24, no. 4, pp. 1869–1924, 2019.
- [81] M. M. Rahman and C. Roy, "Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 473–484.
- [82] K. Cao, C. Chen, S. Baltes, C. Treude, and X. Chen, "Automated query reformulation for efficient search based on query logs from stack overflow," in *Proc. 2021 IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 1273–1285.
- [83] Z. Gao, X. Xia, D. Lo, J. Grundy, X. Zhang, and Z. Xing, "I know what you are searching for: Code snippet recommendation from stack overflow posts," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, pp. 1–42, 2023.
- [84] O. Chaparro, J. M. Florez, and A. Marcus, "Using observed behavior to reformulate queries during text retrieval-based bug localization," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 376–387.
- [85] O. Chaparro, J. M. Florez, and A. Marcus, "Using bug descriptions to reformulate queries during text-retrieval-based bug localization," *Empirical Softw. Eng.*, vol. 24, no. 5, pp. 2947–3007, 2019.
- [86] M. M. Rahman and C. K. Roy, "Improving IR-based bug localization with context-aware query reformulation," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 621–632.
- [87] J. M. Florez, O. Chaparro, C. Treude, and A. Marcus, "Combining query reduction and expansion for text-retrieval-based bug localization," in *Proc. IEEE Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 166–176.
- [88] Z. Sun, L. Li, Y. Liu, X. Du, and L. Li, "On the importance of building high-quality training datasets for neural code search," in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 1609–1620.



Ke Liu received the B.S. degree in computer science from Nantong University, Nantong, in 2020. She is currently pursuing the master's degree with the School of Information Science and Technology at Nantong University. Her research interest is in mining software repositories.



Xiang Chen (Member, IEEE) received the B.Sc. degree from the School of Management, Xi'an Jiaotong University, China, in 2002. Then, he received the M.Sc. and Ph.D. degrees in computer software and theory from Nanjing University, China, in 2008 and 2011, respectively. He is currently an Associate Professor with the School of Information Science and Technology, Nantong University, Nantong, China. His research interests include software engineering, particularly software testing and maintenance, software repository mining, and empirical

software engineering. He has authored or co-authored more than 90 papers in refereed journals or conferences, such as IEEE TRANSACTIONS ON SOFTWARE

ENGINEERING, *ACM Transactions on Software Engineering and Methodology*, *Information and Software Technology*, *Journal of Systems and Software*, IEEE TRANSACTIONS ON RELIABILITY, International Conference on Software Engineering, The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, International Conference Automated Software Engineering, International Conference on Software Maintenance and Evolution, and IEEE International Conference on Program Comprehension and International Conference on Software Analysis, Evolution and Reengineering. He received ACM SIGSOFT Distinguished Paper Awards in ICSE 2021. He is the Editorial Board Member of Information and Software Technology. More information about him can be found at: <https://smartse.github.io/index.html>.



Chunyang Chen is a Senior Lecturer (Associate Professor) with the Faculty of Information Technology, Monash University, Australia. His research focuses on software engineering, deep learning, and human-computer interaction. He has published over 50 papers in refereed journals or conferences, including IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, *ACM Transactions on Software Engineering and Methodology*, *Empirical Software Engineering*, International Conference on Software Engineering, The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, IEEE/ACM International Conference Automated Software Engineering, and ACM Conference on Human Factors in Computing Systems. He received ACM SIGSOFT Distinguished Paper Awards in ICSE 2021, ICSE 2020, and ASE 2018, the BEST PAPER AWARD in SANER 2016, and the BEST TOOL DEMO in ASE 2016. More information about him can be found at: <https://chunyang-chen.github.io/>.



Xiaofei Xie received the B.E., M.E., and Ph.D. degrees from Tianjin University. He is currently an Assistant Professor with Singapore Management University, Singapore. His research mainly focuses on program analysis, traditional software testing, and artificial intelligence quality assurance analysis. He has published over 60 papers in refereed journals or conferences, including IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, *ACM Transactions on Software Engineering and Methodology*, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, *Empirical Software Engineering*, International Conference on Software Engineering, The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, IEEE/ACM International Conference Automated Software Engineering, International Symposium on Software Testing and Analysis, International Conference on Machine Learning, International Conference on Computer Vision, and International Conference on Learning Representations. He won three ACM SIGSOFT Distinguished Paper Awards in ISSA'22, ASE'19, and FSE'16. More information about him can be found at: <https://xiaofeixie.bitbucket.io/>.



Zhanqi Cui received the B.E. degree in software engineering and the Ph.D. degree in computer software and theory in 2005 and 2011, respectively, from Nanjing University, Nanjing. He was a visiting Ph.D. student at the University of Virginia, Virginia, from Sep. 2009 to Sep. 2010. He is an Associate Professor at Beijing Information Science and Technology University, Beijing. His research interests include intelligent software engineering and trustworthy artificial intelligence. More information about him can be found at: <https://zqcui.github.io/>.