



Dialog summarization for software collaborative platform via tuning pre-trained models[☆]

Guodong Fan^a, Shizhan Chen^a, Hongyue Wu^a, Cuiyun Gao^{b,*}, Jianmao Xiao^c, Xiao Xue^a, Zhiyong Feng^a

^a College of Intelligence and Computing, Tianjin University, Tianjin, China

^b School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China

^c School of Software, Jiangxi Normal University, Nanchang, China

ARTICLE INFO

Article history:

Received 27 October 2022

Received in revised form 20 May 2023

Accepted 23 May 2023

Available online 27 May 2023

Keywords:

Software maintenance

Text summarization

Pre-trained language model

Prompt tuning

Live chat

ABSTRACT

Software collaborative platforms, e.g., Gitter live chat and GitHub Discussions, are essential in software maintenance. Summarizing the live chat logs is useful for extracting, retrieving, and sharing knowledge for software developers. Automatic text summarization has been studied in many areas such as code summarization, and title generation. However, the previous studies rely on rich collected labeled data for model training which are absent for the noisy interleaved dialogs, resulting in poor performance in the few-shot scenario. To tackle the issue, we propose a novel Automatic Dialog Summarization Approach based on pre-trained models, named *ADSum*. To alleviate the high-cost problem of the from-scratch manual annotation, *ADSum* finetunes the Text-To-Text Transfer Transformer (T5) model by exploiting the discussion posts on GitHub, and then recommends summaries for an annotator. To solve the poor performance in the few-shot scenario, we propose to employ the prompt tuning paradigm for tuning the T5 model by exploiting the disentangled dialog data on Gitter. Meanwhile, the soft prompt is used to avoid the manual effort of designing appropriate prompt templates. To verify the effectiveness of our approach, we extract 38,964 high-quality discussion posts from GitHub and manually annotate 3,039 dialog summarizations from Gitter. Experimental results show our approach achieves state-of-the-art performance in terms of three performance metrics. In particular, our proposed method outperformed the Transformer-based and other pre-training models by 39% and 14%, respectively, on the GitHub dataset regarding the Rouge-L metric. The experiments of handling data scarcity and a human evaluation also confirm the effectiveness of *ADSum*.

© 2023 Elsevier Inc. All rights reserved.

1. Introduction

In the development of open-source software, collaboration communication among developers is essential in software maintenance. The communication is posted for many purposes (Xie et al., 2021), such as bug reports (Song and Chaparro, 2020), feature requests (Iacob and Harrison, 2013), and question answers (Shi et al., 2021b). Software Collaborative Platform, such as GitHub Discussions,¹ Gitter,² Slack,³ and Discord,⁴ play an

important role in team communications and collaboration. Different from the instant messaging systems (Shi et al., 2021a), GitHub Discussions is a choice to better share the knowledge of developers and facilitate future code reuse (Hata et al., 2022).

The existing literature has extensively researched on live chat rooms (Shi et al., 2021a,b; Pan et al., 2021; Jiang et al., 2021; Silva et al., 2022; Chatterjee et al., 2020). For example, Shi et al. (2021a) manually study the communication profile, communication structure, discussion topic, and iteration pattern of Gitter chat. They find that the average time consumption for responding to a question in live chat is 50% faster than that in Stack Overflow. To automatically mine the live chat data, disentangling the dialog is needed (Jiang et al., 2021). Based on the disentangled dialog, Pan et al. (2021) propose an automated classification method using handcrafted non-textual features and deep textual features.

However, it is challenging for developers to mine useful knowledge due to the noisy dialogs. To address the challenge, Shi et al. (2021b) propose an automatic issue-solution pair extraction method. After disentangling live chat logs, they detect discussion issues, extract appropriate utterances, and combine them as corresponding solutions. Different from GitHub Issues (Chen et al.,

[☆] Editor: Alexander Serebrenik.

* Corresponding author.

E-mail addresses: guodongfan@tju.edu.cn (G. Fan), shizhan@tju.edu.cn (S. Chen), hongyue.wu@tju.edu.cn (H. Wu), gaocuiyun@hit.edu.cn (C. Gao), jm_xiao@jxnu.edu.cn (J. Xiao), jzxuexiao@tju.edu.cn (X. Xue), zyfeng@tju.edu.cn (Z. Feng).

URL: <https://www.guodongfan.com> (G. Fan).

¹ <https://support.github.com/features/discussions/>.

² <https://gitter.im/>.

³ <https://slack.com/>.

⁴ <https://discord.com/>.

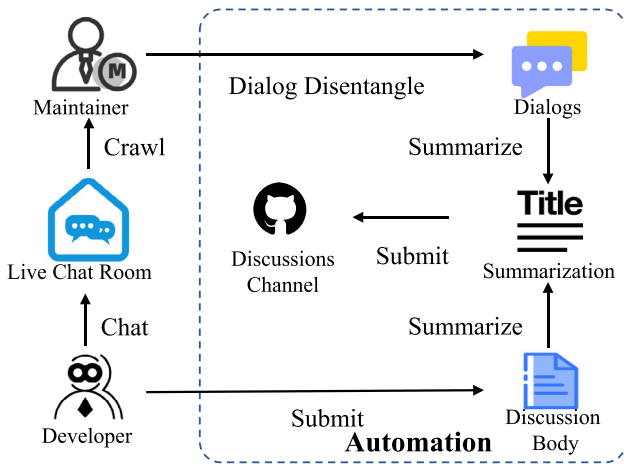


Fig. 1. The workflow for Maintainers/Developers submitting information in software collaborative platform.

2020) defining pieces of software maintenance work (GitHub, 2022), GitHub Discussions as a community knowledge base (Hata et al., 2022) shares questions and ideas that require team communication to make decisions.

For convenient retrieval of knowledge exchanged by developers, project maintainers move and summarize live chat dialogs to the GitHub Discussions channel, as shown in Fig. 1. For example, in a real case of the NocoDB project on GitHub,⁵ software maintainers have moved relevant discussions from their live chat on Discord to GitHub Discussions. These discussions contain valuable information and insights that can be used by developers to enhance the project. By moving these threads (dialogs) to a more visible and structured platform like GitHub Discussions, the maintainers ensure that important knowledge and ideas are not lost in the transient nature of live chat. Additionally, this allows developers who may not have been present during the live chat to have access to the discussions and benefit from the knowledge shared. This supports the claim that moving threads from live chat to Discussions can be a useful approach for maintaining knowledge and promoting collaboration within a software collaborative platform. With the development of natural language processing (NLP), automatic summarization has made a great success in many areas such as bug reports (Chen et al., 2020), Stack Overflow (Liu et al., 2022), and code snippets (Gao et al., 2020). However, several limitations of the aforementioned methods still exist. First, different from these methods (Chen et al., 2020; Liu et al., 2022; Gao et al., 2020) which train on enough data, the Gitter only has scarce data, which is not only due to the noisy nature of the entangled dialogs but requires manual annotation of the summarization of the utterances in the live chat room. Therefore, the existing methods may perform ineffectively in the few-shot scenario. Second, the cost of manual annotation is extremely high, so it is very important to improve the efficiency of manual annotation and reduce the number of annotations.

To solve the above challenges for summarization, in this paper, we propose an Automatic Dialog Summarization approach via tuning pre-trained models named *ADSum*. *ADSum* can better use the knowledge of pre-trained language models (PLMs), e.g., Text-To-Text Transfer Transformer (T5) (Raffel et al., 2020), for the downstream summarization task. There are four main stages as follows. The first stage is preprocessing, which mainly aims to disentangle the noisy interleaved dialogs in the live

chat logs into disentangled dialogs using a feedforward neural network. Then, to alleviate the high-cost problem of the from-scratch manual annotation, *ADSum* fine-tunes T5 by exploiting the GitHub Discussion posts and generates top-*k* summarization recommendations for Gitter dialog data using the beam search method. Next, based on the recommendations, an annotation tool is developed for manual annotation, in which four operations are defined for convenient annotation e.g., select recommendations, edit by oneself, select & edit, and pass. Finally, to solve the poor performance of existing few-shot learning for approaches, we use the prompt-tuning technique to fully explore the knowledge of the pre-trained model T5, by leveraging the few-shot of annotated Gitter data to generate the summarization of the dialog. Our main contributions are shown as follows.

(1) **Technique.** We formulate the problem of dialog summarization for live chat dialog and discussion. Then, to solve the problems of the high cost of from-scratch manual annotation and limited performance in the few-shot scenario, we propose *ADSum* to exploit the knowledge of the pre-trained models for the summary tasks effectively.

(2) **Experiment.** We conduct an extensive evaluation of the performance of *ADSum*. Furthermore, we conduct a human evaluation to confirm the grammatical fluency, relevance, and accuracy of the summaries generated by *ADSum*.

(3) **Tool.** To help manual dialogs annotation, we develop a semi-automatic annotation tool for annotators. In addition, we share our code, dataset, and tools on our project homepage⁶ to facilitate the replication of our study.

The rest of this paper is organized as follows. Section 2 describes the motivation and problem formulation. Section 3 introduces our approach. Experiments setup is presented in Section 4 and the experiment results are described in Section 5. Section 6 discusses the threats to the validity of our approach. After the review of related work in Section 7, we conclude this paper and show possible future directions.

2. Motivation and problem formulation

In this section, we introduce the **Motivating Examples** and **Problem Formulation** of this paper.

2.1. Motivating examples

Fig. 2 shows motivating examples of our approach. The left one displays dialogs in the live chat room, while the right one shows a specific discussion in GitHub Discussions. The workflow of the two examples is shown in Fig. 1.

For the left one, we can see that there are two threads in the live chat log, with different colors for demonstration. The special issue for the live chat dialog is as follows: (1) The dialogs are entangled with each other which leads the developers to be confused about what the chat is talking about. (2) There are no ground truth summarizations, so it is impossible to train or fine-tune an automatic summarization model. Thus, in the paper, we first disentangle the log into two dialogs. Then, we need to summarize each dialog with a title for developers to better understand. It is also useful to move threads from live chat to Discussions from time to time. **In order to reduce the burden of the above process for software maintainers, it is necessary to provide an automated way to summarize dialogs in the live chat log.**

For the one on the right, when the developer writes a discussion, it is useful to help the developer automatically generate titles. In addition, dialogs in the live chat log can also be extracted

⁵ <https://github.com/nocodb/nocodb/discussions/1944>.

⁶ <https://github.com/guodongfan/ADSum>.

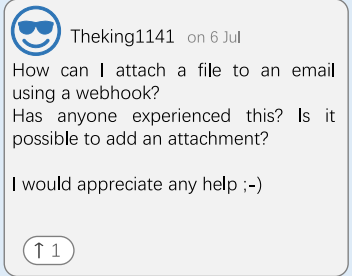
① Entangled Dialogs			② Discussion
Time	ID	Textural Message	Attach file to webhook #2589
[10:40:28]	D1	Can someone explain the reasoning why TS works like this? [link] I would assume that all of those should be valid or invalid, but not a weird mixture like this.	Unanswered theking1141 asked this question in Q&A
[10:42:30]	D2	Is there a way to force the types of an object without having to create a type beforehand? [code]	
[10:42:51]	D3	Where do you want to apply the type? initialVideoPlayerState will have an implicit type already and you could reuse that by saying...[code]	
[10:44:51]	D4	@D1 microsoft/TypeScript#48014	
[10:46:41]	D1	Ah thanks!	
[10:47:01]	D2	how can I allow the HTMLVideoElement to have an optional extra prop called playIndex?	

Fig. 2. Examples. The left one denotes the dialogs in the live chat log, and the right one denotes a submission in the GitHub discussion channel.

and summarized to GitHub Discussions for convenient sharing development knowledge. However, different from other summarization tasks (Chen et al., 2020; Gao et al., 2020), the GitHub Discussions is announced nearly in 2020, and lots of projects do not use it, and it is difficult to manually annotate a lot of summarizations for the live chat dialogs. In another word, the data is far from enough to train a model from scratch. **Therefore, it is helpful and meaningful to investigate an automatic title generation method exploiting the pre-trained models in few-shot scenarios.**

2.2. Problem formulation

We introduce the three concepts in live chat including utterance, chat log, and dialog. In a live chat room, there is a sequence of utterances one by one denoted as *log*. For each utterance u in a live chat room, we denote the identifying number of the utterance as *id*, the timestamp of the utterance as *time*, the developer who sends the utterance as *developer*, and the content of the utterance as *text*. Our task is to automatically disentangle the chat *log* into a series of *dialogs*, and then summarize the *dialog* to a title. We formulate the chat disentangling process as follows:

$$\begin{aligned} \log &= \{u_1, u_2, \dots, u_n\} \\ \text{dialog}_i &= \{u_i | S(u_i) = s, u_i \in \log\} \\ u &= \langle id, time, developer, text \rangle \end{aligned}$$

where s indicates the subject of the current utterance u_i , and S represents the subject function. For GitHub Discussions, as shown in Fig. 11, we structure each discussion as question body and title pairs shown in Eq. (1), which is much simpler than dealing with the dialog in live chat.

$$\text{discussion} = \langle \text{body}, \text{title} \rangle \quad (1)$$

Overall, the problem is formulated as:

$$\text{title} = f(\text{dialog}/\text{body}) \quad (2)$$

where *dialog* and *body* represent a disentangled dialog and a discussion content body, respectively. The symbol f denotes the generation model, and *title* denotes the summarization result.

3. Our approach

In this section, we first introduce an overview of our approach. Then we describe the four key steps in detail, including **dialog disentanglement** for the live chat log, the process of **T5 fine-tuning for summarization recommendations**, the **manual annotation of dialogs**, and **few-shot learning for dialog summarization**, as shown in Fig. 3.

3.1. Overview

Our approach consists of four main steps as shown in Fig. 3. The first step is preprocessing which aims to make a preparation for model training and inferring and mainly includes dialog disentanglement after data crawling and cleaning. The second step is training a recommendation model using GitHub Discussions posts which aims to recommend titles for the disentangled dialogs in the live chat log. The third step uses our developed annotation tool to assist manual annotation combined with the recommended titles, which can improve the efficiency of the annotator and the effect of annotation. Finally, using the manual annotation of the few-shot data, prompt tuning learns to generate dialog summarizations. We present each step in detail in the following subsections respectively.

3.2. Pre-processing

We first normalize non-ASCII characters like emojis to standard ASCII strings. Then, we use a custom dictionary wordMapper (Phong et al., 2015), which contains common misspellings, abbreviations, and their corresponding correct forms to reduce useless information. Some low-frequency tokens such as email addresses, and version numbers contribute little to the generation task, so we replace them with specific tokens <EMAIL> and <ID>, respectively. For the code snippets, we denote them using tags such as <CODE> in the HTML attribute and drop the code snippets at the back of the message. We utilize NLTK⁷ to tokenize sentences into terms and perform lemmatization and lowercasing on terms with NLTK to alleviate the influence of word morphology.

⁷ <http://www.nltk.org/>.

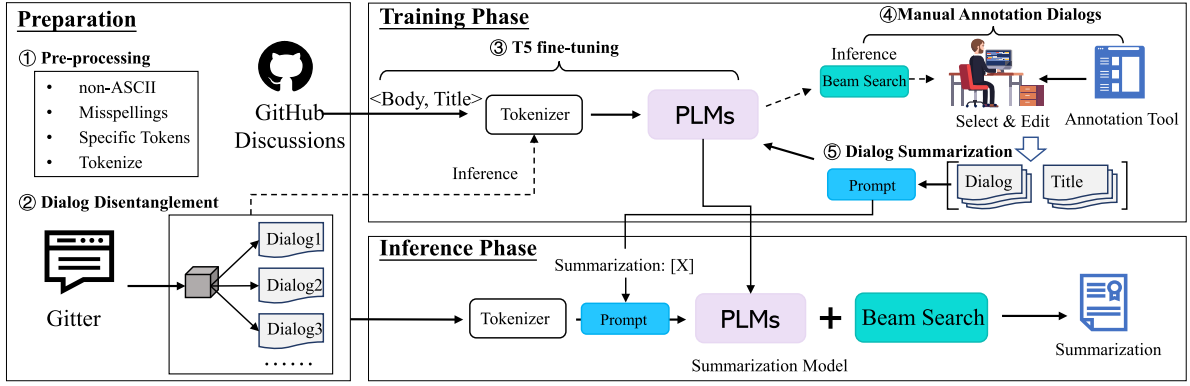


Fig. 3. Overview of our approach.

3.3. Dialog disentanglement

In this step, we focus on disentangling the interleaved dialogs into a series of separate dialogs. There are two types of methods to disentangle the live chat log, classified by the number of stages. The most commonly used method is two-stage dialog disentanglement with a feed-forward model (Kummerfeld et al., 2019). The method first involves link prediction to model “reply-to” relation between two utterances. Then, they use a clustering step to utilize the results from link prediction to construct the individual conversation threads. Another one is an end-to-end online framework for dialog disentanglement (Yu and Joty, 2020), which embeds timestamp, speaker, and message text, and learns the reply-to relationship and pairwise relationship jointly. According to Jiang et al. (2021)’s study, the feed-forward model is reported to achieve relatively good performance, so we use the feed-forward model to disentangle the dialogs. Then, we concatenate each of the utterances in the disentangled dialog following the Eq. (3), and X_{dialog} is used as the original input of the prompt template.

$$X_{dialog} = \langle \text{SEP} \rangle \oplus u_1 \oplus \langle \text{SEP} \rangle \oplus u_2 \oplus \langle \text{SEP} \rangle \oplus u_i \dots \quad (3)$$

3.4. T5 fine-tuning for summarization recommendations

To alleviate the high-cost problem of the from-scratch manual annotation, we use the discussion posts to train a recommendation model based on the fine-tuning paradigm (Liu et al., 2022). Fine-tuning using a PLM for downstream tasks is a prevalent paradigm in the NLP field (Houlsby et al., 2019). It aims at exploiting the knowledge learned by PLMs without learning from scratch and can be regarded as a way of applying transfer learning. The GitHub discussions have relatively rich supervisory signals, as each discussion text corresponds to a title, so the fine-tuning paradigm is suitable for training the recommendation model. The discussion content does not only contain text descriptions but code snippets. Therefore, we use a `<code>` identifier to distinguish input text description X_{text} and input code snippet X_{code} (Liu et al., 2022). The format of the input of fine-tuning is shown as follows.

$$X_{discussion} = X_{text} \oplus \langle \text{code} \rangle \oplus X_{code} \quad (4)$$

Then, a Transformer-based pre-trained language model T5 (Raffel et al., 2020) is fine-tuned exploiting the formatted discussion posts. To adapt PLMs to downstream tasks, fine-tuning trains the model in a supervised way. Specifically, given a dataset that consists of task-specific samples X and corresponding labels Y , fine-tuning aims to find a set of parameters θ to minimize the

negative log-likelihood of the target text Y conditioned on a given input text X , that can be denoted as :

$$Y = \arg \min_{\theta} P(Y|X; \theta) \quad (5)$$

Semi-automated methods can improve the efficiency and quality of annotation (Sun et al., 2018). Therefore, before manual annotation, the recommendation model is used to generate reference titles Ref_{title} for the disentanglement dialogs on Gitter. To generate more choices and to avoid bias (Ravaut et al., 2022), we use beam search (Wiseman and Rush, 2016), denoted as BS , to obtain a list of the top- k candidates (Gong et al., 2022) for the annotators, which is shown in Eq. (6).

$$[Ref_{title_1}, Ref_{title_2}, \dots, Ref_{title_k}] = BS(PLM, X_{dialog}) \quad (6)$$

3.5. Manual annotation of dialogs

The process of manual annotation mainly consists of two steps, including consistent assurance, and annotation processes. We randomly select dialogs and assign two annotators to summarize and label topics. After a rule specification and consistency check, a third annotator reviews and resolves conflicts.

To ensure the consistency of our annotation, based on the results of the recommendation shown in Eq. (6), we first develop a data annotation tool with four operations to obtain titles(select, edit, select & edit, and pass). Then, according to an open-coding scheme (Shen et al., 2021; Islam et al., 2019), we measure the inter-rater agreement among them using the Cohen’s Kappa coefficient (Fleiss, 1971), according to three metrics e.g. consistent, medium, and inconsistent which are defined to manually assess the similarity of the annotations between the two annotators. The results of the first experiment conducted by the two annotators illustrate that Cohen’s Kappa coefficient is nearly 32% for the top 25% of the annotation results in the 1000 samples. Hence, we conduct a session on data annotation and established rules for data annotation. After that, the two annotators annotate 50% of the data(including the previous 25% of the data), and Cohen’s Kappa coefficient achieved 61%. After further discussion of differences, Cohen’s Kappa coefficient exceeds 83% in subsequent data annotation processes.

The established rules are as follows:

- It is recommended to skip the entangled dialogs to avoid confusion.
- Utilize keywords from the original text to summarize effectively.
- When multiple turns questions are present in a dialog, we consider using the first one as the main question.

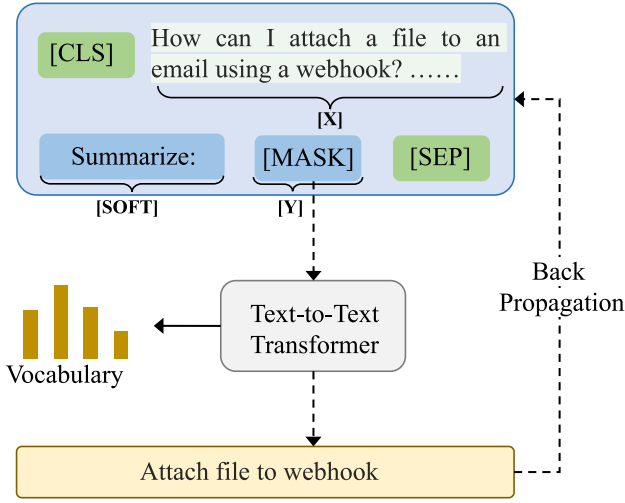


Fig. 4. Diagram of prompt tuning.

In the annotation process, due to the limitation of cross-domain performance (Wang et al., 2022b) which trains using discussion posts and inferences using dialogs on live chat log, we invite three graduate students with at least one year of programming experience, to manually check and annotate the dialogs disentangled in Section 3.3. The disentanglement dialogs are randomly selected by our annotation tool. Specifically, two annotators annotate each dialog with a title according to the reference titles respectively. Then, a third annotator according to the annotation titles makes a final decision, because the third annotator has more experience and communication skills. Finally, we get a series of <dialog, title> pair data that is used to tune the PLMs for generating the summarizations of the dialogs.

$$\text{Pair} = \langle X_{\text{dialog}}, Y_{\text{title}} \rangle \quad (7)$$

3.6. Few-shot learning for dialog summarization

The encoder-decoder structure of T5 allows the model to comprehend the context of the input text and generate a summary that accurately captures the key points (Liu et al., 2022; Ding et al., 2021). T5's encoder-decoder structure is particularly suited for text-to-text summarization, as it enables the model to comprehend the input text's context and produce a summary that effectively captures the main semantics. The effectiveness of T5 has been demonstrated in Vulnerability repair, and text generation fields (Fu et al., 2022; Yu et al., 2022).

To solve the poor performance in the few-shot scenario, prompt tuning learns to generate the dialog summarizations exploiting the manual annotation data. Prompt function $f_{\text{prompt}}(\cdot)$ is applied to modify the input text x into a prompt $x' = f_{\text{prompt}}(x)$. It consists of two steps as shown in Fig. 4: (1) apply a template, which is initiated by a textual string that has two slots: an input slot [X] for input text x and an answer slot [Y] for a generated summarization text y that will be mapped into it later. (2) Fill slot [X] with the input text x .

According to research proposed by Wang et al. (2022b), prefix soft prompt, vanilla soft prompt, and hard prompt can generate comparable performance. However, the hard prompt needs more effort to be well-designed. *ADSum* uses soft approach (Lester et al., 2021) and initialized by predefined tokens e.g., "Summarization:", shown as follows.

$$[\text{SOFT: Text: }][X][\text{SOFT: Summarization: }][Y][\text{SEP}] \quad (8)$$

In the training phase, given a series of tokens, x_0, x_1, \dots, x_n , the pre-trained T5 model first tokenizes and embeds the tokens, forming a matrix $X_e \in \mathbb{R}^{n \times e}$ where n is the length of the tokens, and e is the dimension of the embedding space. Soft prompts are represented as a parameter $P_e \in \mathbb{R}^{p \times e}$, where p is the length of the prompt. The prompt can be concatenated to the embedded input forming a single matrix $[P_e; X_e] \in \mathbb{R}^{(p+n) \times e}$ which then flows through the encoder-decoder as its normal. The new conditional generation is $P_{p\theta; \theta p}(Y|[P_e; X_e])$, where $p\theta$ represents the T5' parameters and θp represents the parameters of prompt. The prompt model is trained to maximize the probability of Y , but only the prompt parameters P_e are updated. The loss function of prompt tuning is shown in Eq. (9).

$$\mathcal{L}_{\text{prompt}}(\theta|\mathcal{Y}, \mathcal{X}, \mathcal{S}, \mathcal{M}) = - \sum_i^N \log P_{\mathcal{M}}(y_i | s_{1:p}, X, Y_{<i}), \quad (9)$$

where θ denotes the trainable parameters of the prompt, \mathcal{Y} denotes the target tokens, \mathcal{X} denotes the source tokens, \mathcal{S} represents the soft prompt vector, and \mathcal{M} represents the frozen parameters of a PLM.

The inference phase aims to disentangle the dialogs and generate concise and useful summarizations for the live chat log. As shown in Fig. 3, given a live chat log, *ADSum* first disentangles log into a series of dialogs, then tokenizes and embeds the tokens. Finally, the prompt-tuned T5 model with beam search is used to obtain the corresponding summarization.

4. Experiments setup

In this section, we investigate three research questions, introduce the dataset of our experiments, list the details of our implementation, and finally present the metrics and baselines.

4.1. Research questions

We conduct quantitative analysis to evaluate the effectiveness of our approach and aim at answering the following research questions (RQs).

- **How effective is our approach in the summarization tasks?** This RQ aims to investigate the summarization accuracy of *ADSum*. To answer this question, we compare the quality of the summarization generated by *ADSum* and baseline methods.
- **How capable is prompt tuning to handle data scarcity in our scenario?** This RQ aims to validate the effectiveness of prompt tuning and our annotation tool in the few-shot scenario. We answer the questions from three perspectives, including performance evaluation, choice of the number of annotations, and tool efficiency.
- **What is the performance of our approach under human evaluation?** This RQ aims to evaluate the performance of our approach in a human evaluation manner.

4.2. Dataset

We first select projects with more than 1000 stars through the GitHub graphql⁸ API until June 10, 2022. Then we crawl the Discussion module in each project separately. Finally, we obtain 82,041 pieces of Discussion information in 339 projects. According to the rules proposed by Chen et al. (2020), we further filter Discussion posts, obtaining 31,017 posts and title pairs as training data, 2971 pairs as validating data, and 3976 pairs as test data respectively.

The GitHub Discussions are filtered using the rules followed by Chen et al. (2020).

⁸ <https://docs.github.com/en/graphql/overview/explorer>.

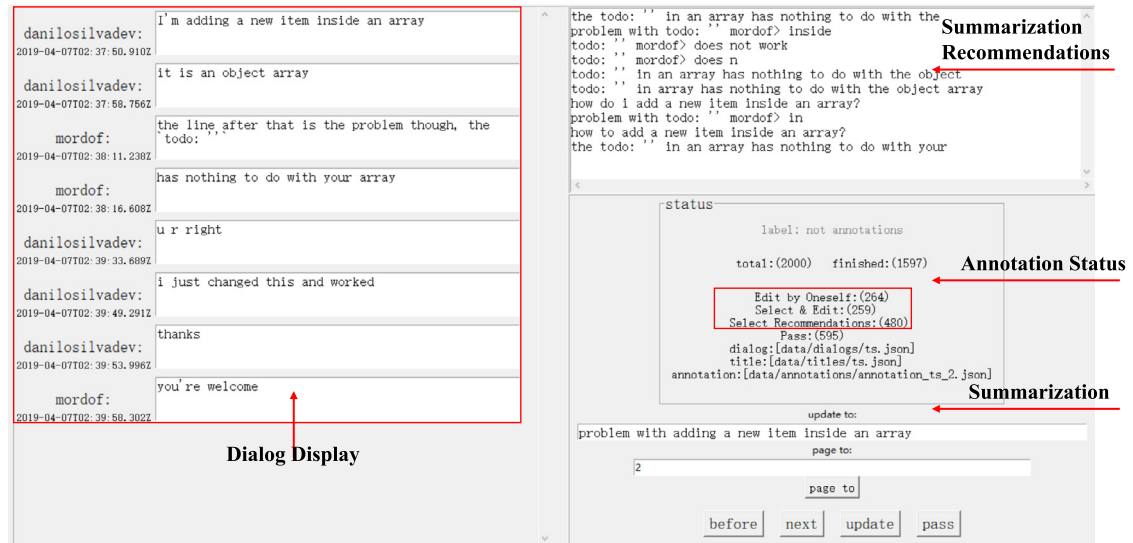


Fig. 5. Screenshot of the annotation tool.

Table 1

Statistical information of dataset. The number in brackets denotes the number of messages involved in the dataset.

Datasets	Projects	Data size	Avg. Title Len.	Avg. Body Len.
Discussion	339	37,964	139.0	7.7
Gitter	Appium	1033 (13,961)	164.6	10.2
	TypeScript	1002 (11,959)	156.3	10.1
	Ethereum	1004 (14,681)	183.6	10.1

- Discussion titles that have less than 3 words, or more than 15 words, are filtered.
- Discussion titles that have more than 70% words missing in the bodies are not considered suitable for our context and are discarded.
- Discussion with titles that has a sub-sequence that can exactly match a particular part of the body, in which the length of the sub-sequence is over 70% of the title length in tokens, are discarded.

For dialog information, because of the easy access through the Gitter API,⁹ we can easily crawl all the data in the live chat room (Parra et al., 2020). We use the following rules to filter dialog data:

- Dialogs with fewer than two participants or less than two messages are filtered.
- If the length of the longest message in the dialog has less than 20 words, the dialog will be discarded.

Then, we manually annotate the corresponding summarization titles for 3039 pairs of dialogs from selected three projects which are TypeScript, Appium, and Ethereum, of which 1033 for Appium, 1004 for TypeScript, and 1002 for Ethereum. Data statistics are presented in Table 1. For GitHub data, the training set is created by randomly selecting 70% of the data, while the remaining 30% is equally split between the validation and test sets. For Gitter data, we divide the dataset into training, validation, and testing sets based on the project. The screenshot of our annotation tool is shown in Fig. 5, which mainly contains four components, “Dialog Display”, “Summarization Recommendations”, “Annotation Status”, and the final “Summarization”.

4.3. Implementation details

We train our model using a machine with an Intel Xeon Processor (Skylake, IBRS) with 16 cores, 32G memory, and 1 NVIDIA Tesla V100 with 16 GB memory. The running OS platform is Ubuntu Linux. The machine learning models are implemented with the PyTorch library. The PLMs such as T5-base, T5-small, and the corresponding tokenizer are downloaded from Huggingface. We use OpenPrompt¹⁰ as the implementation of prompt tuning. We set the maximum source length and target length as 256 and 20 respectively. The model is trained using the batch size of 10 for 20 epochs. More details can be found in our shared code.

4.4. Evaluation metrics

To automatically evaluate the performance of our approach, we use three metrics, i.e. Rouge, BLEU, and Meteor. These three performance metrics are widely used in text and code summarization. The details of these metrics are shown as follows:

Rouge (Lin and Och, 2004), a recall-based metric, is used to calculate the lexical overlap between generated titles and reference titles. Particularly, Rouge-1 and Rouge-2 are based on 1-gram and 2-gram respectively. Rouge-L is based on the longest common subsequence. The Longer the length, the higher the Rouge-L score.

BLEU (Papineni et al., 2002) is a performance metric to evaluate the degree of coincidence of n-grams in generated titles and reference titles. Corresponding to Rouge, the BLEU is calculated based on precision. BLEU-1 and BLEU-2 correspond to the scores of unigram, and 2-grams respectively.

Meteor (Banerjee and Lavie, 2005) is an automatic metric for machine translation evaluation, and it can the capability to incorporate knowledge from WordNet to expand the synset when computing the F-value. Meteor is designed to address the limitations of BLEU which relies on direct word-to-word matching.

4.5. Baselines

We compare our proposed approach with four baselines to prove its effectiveness, including the information retrieval approach and generation approach.

⁹ <https://developer.gitter.im/docs/third-party-api-wrappers>.

¹⁰ <https://thunlp.github.io/OpenPrompt/>.

- Information Retrieval (Yang et al., 2022) is suitable in code clones scenario. CCGIR uses semantic similarity, lexical similarity, and syntactic similarity to determine the retrieval results. We modify the CCGIR by deleting the syntactic and lexical similarity because the natural language does not have abstract syntax tree information and the length of the body content is long. As this model outperforms the generated model in several scenarios, we select it as our baseline to assess the applicability of the different approaches.
- NMT (Bahdanau et al., 2015) is first used in the neural machine translation area and it can automatically translate the code commit to text summarization. We use NMT as the baseline because it is one of the most basic generative models, which enables us to assess the performance of the basic model in our scenario.
- Transformer (Vaswani et al., 2017) makes it possible for the model to have more parameters and therefore achieve more accurate results. Recently, Transformer has shown effective in different generation tasks such as product descriptions generation (Chen et al., 2019), opinion summarization (Suhara et al., 2020), and query reformulation (Cao et al., 2021).
- iTAPE (Chen et al., 2020) is an RNN-based approach adopting the Copy mechanism with human-named token tagging. We choose iTAPE as a baseline because it can achieve promising performance in the generation of bug report titles which is similar to ours. Therefore, we choose this approach as one of our baseline methods.

5. Experiment results

In this section, we perform a result analysis for the three research questions and provide case examples for qualitative comparison.

5.1. RQ1: How effective is our approach in the summarization tasks?

Before evaluating the summarization, we compare other methods for dialog disentanglement on Gitter. We compare the FF model proposed by Kummerfeld et al. (2019) against a rule-based method proposed by Ehsan et al. (2020). Followed by Jiang et al. (2021)'s work, we make comparisons using the three metrics, e.g. recall, precision, and F1. From the experimental results, the rule-based model has achieved a better recall value, whereas the FF model has higher accuracy. Furthermore, the F1 of the FF model is 35% higher than the rule-based method on average (see Table 2). This indicates that using the FF model as our foundation is a good choice.

Then, we proceed to compare the performance of the summarization tasks. To make the comparison fair, we adopt the same form dataset and the same random seed for the models. To avoid the result differences due to different implementation versions of these performance measures, we utilize these models to generate an output file, and then we use the same evaluation metric program for comparison.

The comparison results between our approach and baselines introduced in Section 4.5 are shown in Table 3. We can see that T5 Fine-tuning achieves the best performance with 32.21% Rouge-1 metric, and the T5 Prompt Tuning achieves the second performance with 29.01% Rouge-1 metric. Compared with iTAPE, our approach achieves a 39% performance improvement with the Rouge-1 metric. Due to the lack of data, Transformer does not work well in our scenario. Compared to the T5-small model, T5-base also shows significant improvement. For instance, with prompt tuning, T5(-base) achieves a 12% improvement over T5-small in terms of the Rouge-L metric. The Information Retrieval

approach works the worst, because there are not many duplicates or similar entries in the discussion posts which are different from the code clone scenario.

There is not enough labeled data available for live chat data on Gitter compared to GitHub Discussion data. The comparison results are shown in Table 3. We can see the T5 prompt tuning approach achieves the best performance under the 51.52 Rouge-L metric, and the T5 fine-tuning approach achieves the second best performance under the 50.40 Rouge-L metric. Compared to the T5-small model, T5-base also shows significant improvement. For instance, with prompt tuning, T5(-base) achieves a 12% improvement over T5-small in terms of the Rouge-L metric. The hard prompt T5 approach has poor performance because without supervised signals it is difficult to find the optimal hard prompt template. The experimental results indicate that T5 prompt tuning is more effective in data-scarce scenarios, such as Gitter, compared to GitHub Discussion, which has relatively abundant data.

Answer to RQ1: *ADSum* can outperform baselines in terms of three metrics for the summarization task. Specially, the T5 Fine-tuning approach is good at training with abundant data, and the T5 prompt tuning is particularly useful when the data are scarce.

5.2. RQ2: How capable is tuning to handle data scarcity in our scenario?

Considering the performance of fine-tuning is known to strongly rely on the amount of downstream data, while scenarios with scarce data or data without labeling in software engineering are common. We focus on three sub-research questions for the data scarcity settings: (1) what is the performance of our approach under different amounts of data? (2) Is manual annotation of 1000-shot enough? and (3) Can annotation tools help developers effectively?

5.2.1. RQ 2.1: What is the performance of our approach under different amounts of data?

For the summarization task on Gitter, we choose 10, 50, 100, 500, and 1000-shot to train the PLMs using fine-tuning or prompt tuning as shown in Fig. 6. We adopt k-Folder cross-validation to build the model and verify the model parameters. More specially, we split our dataset according to the project. Our model is trained using TypeScript data and validated on Ethereum data, and tested on Appium data, as shown in Fig. 6.1. Then, our model is trained using Appium data, validated on TypeScript data, and tested on Ethereum data, as shown in Fig. 6.2. Finally, our model is trained using Ethereum data, validated on Appium data, and tested on TypeScript data, as shown in Fig. 6.3. From the experimental results, we can see that prompt tuning performs better than fine-tuning from 10-shot to 1000-shot. when it comes to 50-shot as shown in Fig. 6.3, the performance of prompt tuning slightly falls behind fine-tuning. It is because prompt tuning introduces additional parameters, and fewer samples may lead to overfitting and poor results. From Fig. 6, we can see that the performance of our approach from 10-shot to 500-shot improves rapidly, especially from 10-shot to 50-shot, and then slows down after 500-shot.

5.2.2. RQ 2.2: Is manual annotation of 1000-shot enough?

We study the training effect of prompt tuning on different amounts of data to justify our manual annotation of 1000 dialogs for each project. To this end, we set x-shot for comparative experiments, ranging from 50 to 4000-shot, using discussion posts, as shown in Table 4. We conduct x-shot trials on the dataset

Table 2
Results of different approaches for dialog disentanglement.

Model	Appium			TypeScript			Ethereum		
	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
FF model	0.68	0.87	0.69	0.70	0.87	0.69	0.75	0.91	0.77
Rule-based method	0.83	0.48	0.53	0.87	0.35	0.42	0.87	0.68	0.69

Table 3
Results of our approach and State-of-The-Art Baselines.

Dataset	Model	Rouge-1	Rouge-2	Rouge-L	BLEU-1	BLEU-2	Meteor
GitHub	Information retrieval	8.07	2.75	1.28	8.07	2.75	3.43
	NMT	19.85	4.28	18.46	18.74	8.75	8.31
	Transformer	20.70	3.74	19.26	19.51	8.25	8.01
	iTAPE	23.01	6.13	21.33	22.00	11.37	10.26
	Hard Prompt T5	17.25	5.10	16.14	14.41	7.65	11.11
	T5-small Fine-tuning	27.53	10.23	25.95	24.82	15.12	14.68
	T5-small Prompt tuning	26.93	9.98	25.45	24.01	14.70	14.38
	T5 Fine-tuning	32.21	12.43	29.77	28.88	18.01	15.64
	T5 Prompt tuning	29.01	11.44	27.05	24.90	15.76	15.31
Gitter	Information retrieval	7.72	1.24	7.62	7.98	3.10	3.19
	NMT	12.80	1.97	12.27	12.06	5.01	5.27
	Transformer	24.29	2.04	22.37	16.85	5.44	5.25
	iTAPE	27.60	9.27	26.27	25.25	15.03	11.31
	Hard Prompt T5	26.47	16.60	25.72	26.49	21.02	15.84
	T5-small Fine-tuning	43.87	32.76	43.33	42.24	36.53	25.42
	T5-small Prompt tuning	46.26	34.50	45.79	46.87	40.73	26.80
	T5 Fine-tuning	50.94	39.23	50.40	47.80	41.99	29.43
	T5 Prompt tuning	52.09	40.61	51.52	49.77	43.92	29.81

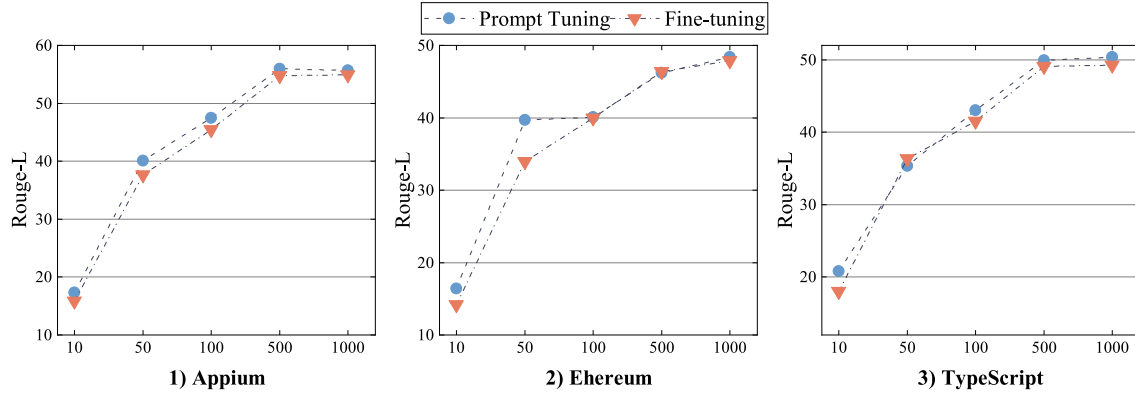


Fig. 6. The performance of ADSSum under different amounts of data.

constructed by 339 GitHub projects, as introduced in Section 4.2. To ensure diversity, we randomly select N pieces of data from the dataset. We can see that from 50-shot to 500-shot, the Rouge metrics increase rapidly. However, when it comes to over 500, the performance shows a slow rising process. For the Rouge-2 metric, the 1000-shot achieves the best performance, which is slightly over 2000-shot and 3000-shot. Therefore, we compromise the performance of difficult annotation and few-shot setting, which proves that the annotation of 1000-shot is reasonable.

The results of our experiments demonstrate that prompt tuning is effective for generating high-quality summaries even with limited data, and 1000-shot is a reasonable compromise between difficult annotation and few-shot settings for our task.

5.2.3. RQ 2.3: Can our annotation tool help developers effectively?

To help manual dialogs annotation effectively, we design the annotation tool with four operations, which are “select recommendations”, “edit by oneself”, “select & edit”, and “pass”. The “select recommendations” operation refers to the developers selecting the recommended titles directly by our tools which is the most efficient way. The “edit by oneself” operation refers to the developers editing the titles by themselves because they are not

Table 4
Results of Few-shot settings.

x-shot	Rouge-1	Rouge-2	Rouge-L	BLEU-1	BLEU-2	Meteor
50	21.92	7.93	20.99	19.96	12.11	12.40
100	25.03	9.44	23.84	22.12	13.64	13.15
500	26.32	9.99	24.98	23.09	14.26	14.00
1000	27.16	10.57	25.76	24.47	15.30	14.33
2000	27.34	10.53	25.88	23.99	15.76	14.93
3000	27.51	10.56	26.02	24.74	15.37	14.93
4000	27.78	10.80	26.26	24.20	15.13	14.63

satisfied with the recommendations. The “select & edit” operation means the developers first select one title recommended by our tool, and then based on the recommendation, they make some changes to the content. The “pass” operation means the dialog is confusing or unable to summarize.

Thus, we have statistics on the operations taken by the three annotators introduced in Section 3.5 when they annotate the dialog summarizations, as shown in Table 5. These annotators have all passed CET-6, have at least one year of programming experience, and are familiar with some programming terms. We can see that almost 50% of annotations can be only selected using

Table 5
Statistic of different operations for manual annotation.

	TypeScript	Ethereum	Appium
Select Recmds	479(47.8%)	531(52.9%)	514(49.7%)
Edit by oneself	264(26.3%)	129(12.8%)	168(16.3%)
Select & Edit	259(25.8%)	344(34.3%)	351(34.0%)

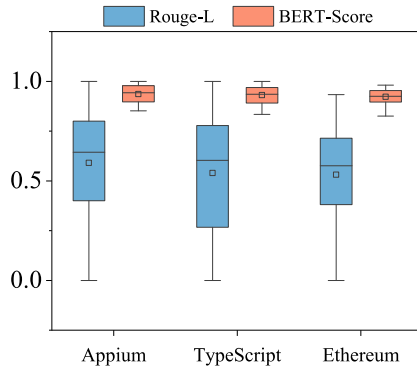


Fig. 7. Bias experiment of manual annotation.

the recommended titles, a quarter of the data can be further modified by the developer's choice, and only less than a quarter of the data needs to be edited by the developer himself. In summary, our tool can effectively reduce developers' manual effort.

In addition, we recruit an annotator to annotate without showing recommendations to prove the usefulness of the recommended titles with as little bias as possible. For each project, we randomly selected 50 dialogs for experimentation. The experimental results are shown in Fig. 7. The Rouge-L score was around 0.6, indicating a certain difference between the annotations made with and without recommended titles. However, the average value of BERT-Score is above 0.9, which proves that the semantic level of annotation is very similar. It can be explained to a certain extent that the bias we recommend for annotation is relatively small in semantic level.

Answer to RQ2: As the data increase, the performance of Prompt tuning gets better, but the increasing trend becomes slower. We show that 1000-shot is enough for manual annotation. Besides, our annotation tool can help developers effectively.

5.3. RQ3: What is the performance of our approach under human evaluation?

We conduct a human evaluation to compare the outputs of our approach with baseline methods. We randomly select 50 dialogs of the live chat log on Gitter and 50 discussion posts on GitHub. For discussion posts, we compare the ground truth titles and those generated by our approach and iTAPE. Due to the data scarcity limitation, dialogs of the live chat log are only compared between manually labeled data and titles generated by our approach. Each summarization is evaluated separately by four students and the statistical results are shown in Figs. 8 and 9.

Inspired by Santhanam and Shaikh (2019) and Santhanam et al. (2020), all the summarizations are evaluated considering three aspects "grammatical fluency", "relevance", and "accuracy". The metric "grammatical fluency" is the degree to measure whether a text is easy to understand. The metric "relevance" relates to the extent of topical relevance between the dialog and title. The metric "accuracy" estimates the degree to which the

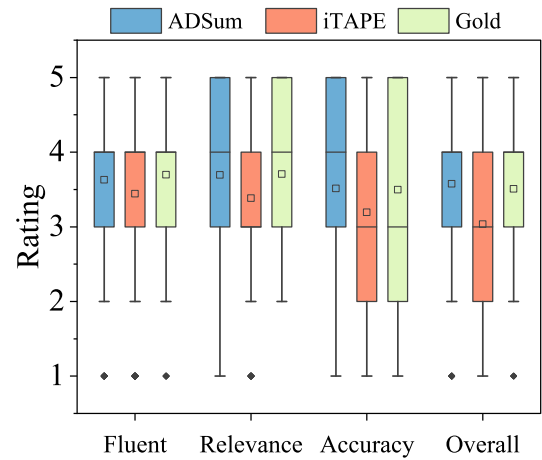


Fig. 8. Manual results for GitHub discussion.

title accurately summarizes a dialog. All metrics are rated on a 1–5 scale, e.g., 5 represents the participant totally satisfying the rating scheme, on the opposite 1 represents they are not satisfying the rating scheme. Besides the above three metrics, each participant is asked to give an overall rating in a similar way.

The experimental results indicate that our approach, *ADSum*, performs similar to the ground truth for discussion posts, and even slightly better in terms of the "relevance" metric shown in Fig. 8. This is because the generated results are more consistent than manual ground truth results. What is more, *ADSum* significantly exceeds the baseline method, which demonstrates the effectiveness of our approach. The performance of *ADSum* across the above four metrics, grammatical fluency, relevance, accuracy, and an overall score, in the Gitter dialogs falls short of the manual annotation results, as demonstrated in Fig. 9. This inadequacy can be attributed to the bottleneck in the few-shot learning performance. In addition, the ratings on the Gitter are higher than those on GitHub, because the manual annotation for dialog summarization ensures consistency, making the results produced by the model more uniform. Our experimental findings demonstrate that *ADSum* outperforms baseline methods and achieves performance similar to that of manual annotation for discussion posts. We believe that *ADSum* has the potential to be a useful approach in this scenario. Our approach could potentially supplement human annotation efforts by providing a quick and consistent summary of a large number of dialogs. However, we recognize that the accuracy of our approach may not be high enough to fully replace human annotation. In future work, we plan to explore how our approach could be integrated with human annotation efforts to improve the overall efficiency and quality of dialog summarization, for example, dialogs that are difficult for machines to summarize are handled by humans.

Answer to RQ3: Through human evaluation, we prove that our approach is better than baseline methods, and even in the few-shot scenario, our approach can generate high-quality titles.

5.4. Qualitative comparison

In this section, we have added four concrete examples to demonstrate the usefulness of our approach. Three examples were collected from Gitter, representing discussions related to TypeScript, Appium, and Ethereum, respectively. The fourth example was obtained from GitHub. These case studies showcase

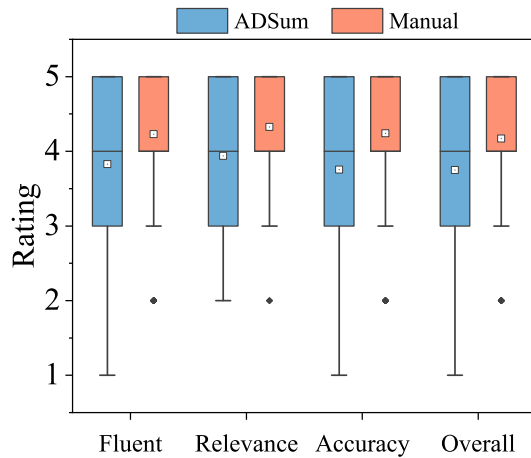


Fig. 9. Manual results for Gitter chats.

our approach's ability to generate informative and concise summarizations.

From the examples on Gitter, we can see that *ADSum* can better capture topics of the disentangled dialogs for summarizing. For instance, Figs. 10 and 12 show *ADSum* extracts more topic keywords in the dialog, e.g., “existing” and “new option”. However, many domain concept problems are not done well, e.g., between “bodyOrFilters” and “bodyOrFilters”, shown in Fig. 11. While our approach has shown effectiveness in generating an informative and concise summary, it is important to note that it tends to summarize what is already present in the input text. *ADSum* cannot capture the related keywords outside the content of the developer's discussions, which can be richer and encompass other related concepts. For instance, in the case of discussion related to “Next.js”, as shown in Fig. 13, a summary generated by our approach may miss certain nuances or connections to the relevant topics that a developer would recognize. Therefore, in the area of dialog summarization, one potential direction is integrating related domain concepts into the model.

6. Threats to validity

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

6.1. Internal threats

Three graduate students with at least one year of programming experience are recruited for data labeling. Two annotate the data separately, and the third who is the leader combines their results for the final label. If there are significant discrepancies, the final result is discussed.

One of the main limitations of annotated data is that it can be expensive and time-consuming to obtain. Pre-trained models can be fine-tuned on small annotated datasets to improve their performance on specific tasks. In contrast, not pre-trained models must leverage larger amounts of data to achieve comparable performance. Therefore, we compare their performance on a larger labeled dataset on GitHub, demonstrating the effectiveness of the pre-trained models.

Another potential threat is bias in the manual annotation. For live chat data on Gitter, there are no ground truth data and manual annotation brings bias due to the annotator's preferences. To solve this problem, we first fine-tuned with GitHub Discussions data, then generate recommendation titles for the live chat dialogs. The annotators inspect and modify titles based

on the recommendations, which also can improve the efficiency of annotators.

The third threat is human evaluation. The potential for construct bias exists as there is an overlap between the students responsible for evaluating and annotating the data. However, we believe that this issue can be mitigated because of the long interval between annotation and evaluation, and the inclusion of new students during the evaluation process. To be fair, the order of titles generated by different methods is shuffled each time throughout the rating process, so no students know in advance which title is generated by which method or whether it is written by developers.

In addition, our configuration of T5, such as the maximum source length of 256 characters and maximum target length of 20 characters, may impact the quality of generated titles due to potential input truncation. However, we believe that this value is appropriate as the average length of our statistical text is no more than 200. We plan to conduct more analysis in future work.

6.2. External threats

The external threats involve dialog disentanglement. Because the performance of our results is based on the accuracy of disentangled dialog, we employed the state-of-the-art FF model proposed by Kummerfeld et al. (2019). We compared it with the classical method and find that the FF has higher performance. Therefore, we believe this can make a good basis for our study as much as possible.

Another potential threat to our study is related to model comparisons, as our baseline includes models built from scratch. To mitigate this risk, we have taken several steps, such as comparing our approach with baselines using two types of data. Specifically, we compared our approach against information retrieval, multiple pre-trained models, and various prompt paradigms.

This project targets software developers and maintainers. The goal is to enable automated title generation for discussions in software chat rooms, with the generated titles uploaded to GitHub for storage. This can save time and effort in manually creating and organizing discussion topics. As for implications for researchers and practitioners, this paper demonstrates the potential of natural language processing and machine learning in automating certain aspects of software development and maintenance. However, to fully automate the process, it is necessary to develop the ability to automatically access each chat room and use the GitHub API for uploading the results. It is worth noting that this approach has its limitations, particularly in terms of potential privacy and security concerns to accessing the content.

7. Related work

In this section, we discuss the related work including software collaborative platforms, knowledge-enhanced generation, and PLMs as a knowledge base for generation for software engineering. This is a subset of deep learning applied to software engineering (Yang et al., 2020).

7.1. Software collaborative platforms

Software collaborative platforms, such as Gitter, Slack, and Github Discussions play a crucial role in software maintenance (Chatterjee et al., 2020; Subash et al., 2022; Chatterjee et al., 2021; Ehsan et al., 2020). These platforms enable developers to ask questions and receive answers from their peers. However, the dialogs on these platforms become entangled, as is often the case in Gitter and Slack chats, and disentanglement techniques become necessary. Researchers typically follow the

```

<engrhxn>      guys i need help
<gromer89>      yes
<engrhxn>      i setup a new ethereum wallet on my laptop
<engrhxn>      bought ethereum coins from coin base
<engrhxn>      and sent them to my wallet
<gromer89>      you dont see it ?
<engrhxn>      but my wallet shows nothing
<engrhxn>      i think i lost my money :(
<gromer89>      go download Agama Wallet and import your existing wallet to it.
                https://www.agamavault.io/
<gromer89>      then you will see the coins, once this wallet not have to sync.
<engrhxn>      my wallet stuck on 2 % synchronization
<gromer89>      yeah it will take for you 2-3 days to full sync
<gromer89>      or just get Agama light wallet, then you dont have to wait
<gromer89>      and can you please guide me how to import existing vault

[Manual] unable to synchronize ethereum wallet with agama
[ADSum] how to import existing vault from ethereum wallet to agama
Wallet

```

Fig. 10. Example in the Ethereum chat room on Gitter.

```

<ZBAGI>        oh wait i know how i can distinguished between 'bodyOrFilters'
<ZBAGI>        If first param is 'object' then 'bodyOrFilters' has to be 'filters'
<ZBAGI>        if first param is 'string' then 'bodyOrFilters' is 'body'
<keithlayne>    Seems like it would be way easier if you just put body as the first
                param
<ZBAGI>        oh yeah then they would be always the same
<ZBAGI>        100% better
<ZBAGI>        ts protected async postRequest(body: any, ...filters: Filter[]):
                Promise<unknown>;\n  protected async postRequest(body: any,
                path: string, ...filters: Filter[]): Promise<unknown>; .....
<ZBAGI>        Awesome, thanks @keithlayne

[Manual] how to distinguish between bodyOrFilters?
[ADSum] how can i distinguish between bodyOrFilters and
bodyOrFilters?

```

Fig. 11. Example in the Typescript chat room on Gitter.

pipeline that dialog disentanglement and classification to analyze and understand these dialogs.

A dialog log in a live chat room can be represented as a graph G composed of utterances. The goal of the Feedforward (FF) model is to predict whether there is an edge E between the utterance u_i and u_j , where the utterances are nodes and the edges represent that one utterance u_j is a response to another u_i , and $i < j$, as shown in Fig. 14.

$$G = \{V, E\}, V = \{u_1, u_2, \dots, u_n\}, E = \langle u_i, u_j \rangle, i < j$$

After training multiple FF models with different random seeds, the union strategy is leveraged to combine their output by keeping all edges predicted. Then, they use a clustering step to utilize the results from link prediction to construct the individual conversation threads used for downstream tasks.

To understand the usage of software collaborative platforms, Chatterjee et al. (2021) find Opinion Q&A has a more strong presence on Gitter. They develop an automatic identification of opinion-asking questions and extraction of participants' answers from public online developer chats. Similarly, Silva et al. (2022) analysis of 87 chat rooms on Gitter and identify 47 themes, where

<Alex-DG> Hi, is someone use the latest Appium version and the inspector ? Because since I've updated my Appium version to the latest I'm no more able to select my UI elements through the app overview on the right side of the inspector. Is there a **new option** to enable this functionality ?

<Simon-Kaz> its a known issue, lots of people logged bugs for it

<Alex-DG> ok fine, we just have to wait for a fix then. Thanks for the information

<Simon-Kaz> np

<Simon-Kaz> you could revert to 1.3.5 too

<Alex-DG> yep

[Manual] unable to select UI elements through app overview on the right side of the inspector

[ADSum] is there a **new option** to enable the selection of UI elements through the app overview

Fig. 12. Example in the Appium chat room on Gitter.

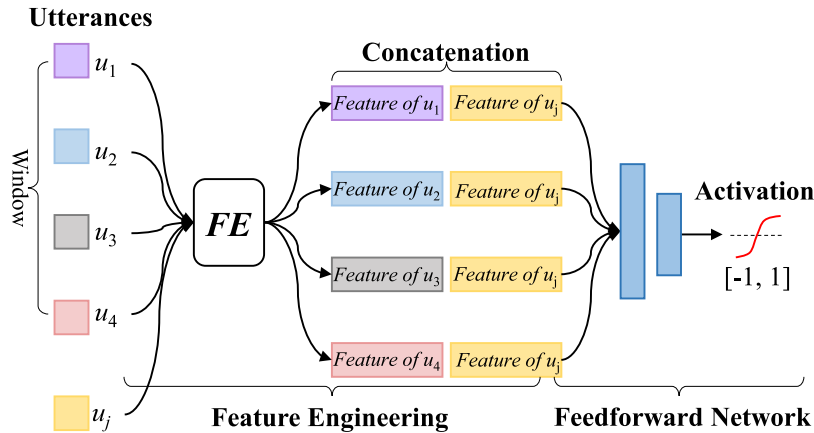
I have read in the docs that a custom express server disables automatic static optimization . After doing some research i was unfortunately still not able to find out whether this means that every page is server-side rendered . Or to put it in a different way : is it possible to **serve static pages** with a custom server ? Would appreciate any help to clarify my confusion .

[Ground Truth] serving static pages **with a next.js** custom express server

[iTAPE] how to serve pages from a custom express server ?

[ADSum] is it possible to **serve static pages** with a custom express server?

Fig. 13. Example in the GitHub discussion.

Fig. 14. Feedforward model for dialog disentanglement. The figure denotes the utterance u_j predicts within-dialog relation with utterances ahead in a window, e.g., u_1, u_2, u_3 , and u_4 .

they present themes in a hierarchical way. Their findings reveal that chat rooms are primarily focused on software development and practices, web development, and code quality themes. In addition, [Tian et al. \(2022\)](#) find that it is critical to help developers generate high-quality messages, and they define what is a good commit message.

Developers make decisions by rationale such as Discussing issues, considering alternatives, and during software development. Rationale is a crucial aspect of decision-making during software development, which helps in documenting and reusing development knowledge. [Alkadhi et al. \(2017\)](#) examine the frequency and completeness of rationale in chat messages, and present automatic techniques for rationale extraction. [Alkadhi et al. \(2018\)](#) conduct an empirical study to understand how OSS developers discuss rationale in IRC (Internet Relay Chat) channels and explore the possibility of automatic extraction of rationale elements by analyzing the IRC messages.

Moreover, there have been attempts to automate classification and detection in chat messages. [Pan et al. \(2021\)](#) propose an automated classification method using handcrafted non-textual features and deep textual features based on the disentangled dialog. [Shi et al. \(2020\)](#) recast the traditional text classification task into the task of determining whether two dialogues are similar or not via a deep Siamese network to detect feature requests from dialogues for few-shot learning.

These studies shed light on various aspects of online developer chats and provide insights into dialog disentanglement, rationale extraction, and automated classification methods. While existing approaches for chat mining concentrate on formatting and classifying the content of the dialog, our approach diverges by tackling the problem of summarizing the dialog to facilitate efficient reuse.

7.2. Knowledge-enhanced generation

Recently, [Yu et al. \(2022\)](#) propose a survey of the knowledge-enhanced generation. They classify the knowledge into internal and external and review the general methods, architectures, and applications for integrating knowledge. We mainly introduce the knowledge-enhanced generation for software engineering.

[Gao et al. \(2019\)](#) propose an NMT-based model named RRGGen which incorporates review attributes and keywords to help developers automatically reply to user reviews. The keywords can indicate review topics and are helpful to recognize the semantics of review and response. To generate commit messages for code changes, [Dong et al. \(2022\)](#) propose FIRA which represents code changes via fine-grained graphs, and then learn to generate commit messages. Meanwhile, they explicitly describe the code edit operations between the old version and the new version. Pull requests description is helpful for a developer to gain a quick understanding, but 34% of pull requests description is empty. Therefore, [Liu et al.](#) propose a pointer generator network method to automatically generate pull request descriptions, where reinforcement learning is used to solve the gap between the training loss and evaluation metric. [Liu et al. \(2022\)](#) propose a transformer-based post-title generation approach for Stack Overflow by leveraging the code snippets and the problem description. Similar to our scenario, [Chen et al. \(2020\)](#) propose an automatic bug reports summarization method named iTAPE, and they propose three rules to refine the dataset and incorporate a copy mechanism to process human-named tokens. To solve the semantic gap between the query and the code, [Wang et al. \(2022a\)](#) propose a reinforcement learning approach to generate semantic enriched queries. Then the enriched queries are employed for code search and achieve significant improvement. In addition, retrieval-based methods also used in generation tasks, e.g., [Zhang et al. \(2022\)](#) propose ShellFusion to

recommend a list of relevant shell commands for programmers, which integrates knowledge mined from Q&A posts in Stack Exchange, Ubuntu MPs, and TLDR tutorials. Different from retrieval from knowledge directly, [Sabour et al. \(2022\)](#) adopt a pre-trained GPT-2 named COMET which is fine-tuned on triplets (*event, relation type, inferred knowledge*) dataset to generate related knowledge from ATOMIC and produces a response generation with the commonsense understanding. Our approach falls under the category of knowledge-enhanced methods as it leverages the existing knowledge encoded in the pre-trained model.

7.3. PLMs and Its usage

With the development of deep learning, the attention technique has made great success in numerous applications, such as image recognition, recommendation system, and natural language processing ([Vaswani et al., 2017](#)).

Based on the encoder of the transformer, BERT ([Kenton and Toutanova, 2019](#)) is proposed as a PLM which can be used in many downstream tasks such as text similarity and text classification. In the contrast, GPT-2 ([Radford et al., 2019](#)) based on the decoder of the transformer architecture also makes it successful in natural language generation problems. Then, T5 ([Raffel et al., 2020](#)), a uniform text-to-text PLM, is proposed by Google, and it makes all the tasks to be generation problems including classification and generation. These methods are also trained in code datasets and adopted for code understanding and generation, e.g., CodeBERT ([Feng et al., 2020](#)), CodeGPT ([Lu et al., 2021](#)), and CodeT5 ([Wang et al., 2021](#)).

In the process of forming prompt tuning, [Petroni et al. \(2019\)](#) find that BERT contains relational knowledge and factual knowledge even without fine-tuning. [Liu et al. \(2021\)](#) survey prompting methods in NLP, and they propose “pre-train, fine-tune” procedure is replaced by “pre-train, prompt, and predict”. In the prompt paradigm, instead of adapting PLMs to downstream tasks via objective engineering, downstream tasks are reformulated, just like training the original PLMs with the help of a textual prompt. Aiming to better use the new prompt paradigm, Tsinghua NLP Lab release a unified easy-to-use toolkit to conduct prompt learning over PLMs called OpenPrompt ([Ding et al., 2021](#)). More recently prompt tuning is used in code intelligence tasks ([Wang et al., 2022b](#)) such as defect detection, code translation, and code summarization, which shows great potential in low-resource scenarios.

Different from these approaches, in this paper, we focus on generating with few-shot learning and exploiting the knowledge of pre-trained models. Specifically, we leverage prompt tuning and fine-tuning paradigms based on the T5 model to address the problem of dialog summarizations in data scarcity scenarios including discussion posts on GitHub and live chat logs on Gitter, which are real-world scenarios.

8. Conclusion

In this paper, we proposed an automatic dialog summarization by disentangling the dialogs and tuning the Text-To-Text Transfer Transformer (T5) model to fully exploit pre-trained model knowledge. This addresses the critical concern of software maintainers reusing knowledge in live chat logs, in which dialogs are entangled in nature and without ground truth titles. We demonstrated the effectiveness of our approach through experiments on two datasets, where we achieved high ROUGE scores and outperformed existing state-of-the-art approaches. To further enhance the quality of the dataset and alleviate the high-cost problem of from-scratch annotation, we developed an annotation tool to facilitate annotation efficiency for annotators, which consists of

four operations, e.g., select recommendations, edit by oneself, and select & edit. The tool can significantly improve the efficiency of the annotation process. Future works may not only consider generating the summarization but also further formatting the data to better extract knowledge from the dialog.

CRediT authorship contribution statement

Guodong Fan: Methodology, Software, Writing – original draft. **Shizhan Chen:** Conceptualization, Methodology, Supervision. **Hongyue Wu:** Data curation, Validation. **Cuiyun Gao:** Data curation, Writing – review & editing, Supervision. **Jianmao Xiao:** Data curation. **Xiao Xue:** Validation. **Zhiyong Feng:** Conceptualization.

Declaration of competing interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the National Natural Science Key Foundation of China grant No. 61832014 and No. 62032016, and the National Natural Science Foundation of China grant No. 61972276, No. 62102281, and No. 62002084, as well as the Jiangxi Provincial Natural Science Foundation under Grant No. 20224BAB212015 and the Foundation of Jiangxi Educational Committee under Grant No. GJJ210338.

References

- Alkadhi, R., Lata, T., Guzman, E., Bruegge, B., 2017. Rationale in development chat messages: an exploratory study. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories. MSR, IEEE, pp. 436–446.
- Alkadhi, R., Nonnenmacher, M., Guzman, E., Bruegge, B., 2018. How do developers discuss rationale? In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, pp. 357–369.
- Bahdanau, D., Cho, K.H., Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate. In: 3rd International Conference on Learning Representations, ICLR 2015.
- Banerjee, S., Lavie, A., 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In: Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization. pp. 65–72.
- Cao, K., Chen, C., Baltes, S., Treude, C., Chen, X., 2021. Automated query reformulation for efficient search based on query logs from stack overflow. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, IEEE, pp. 1273–1285.
- Chatterjee, P., Damevski, K., Kraft, N.A., Pollock, L., 2020. Software-related slack chats with disentangled conversations. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 588–592.
- Chatterjee, P., Damevski, K., Pollock, L., 2021. Automatic extraction of opinion-based Q&A from online developer chats. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, IEEE, pp. 1260–1272.
- Chen, Q., Lin, J., Zhang, Y., Yang, H., Zhou, J., Tang, J., 2019. Towards knowledge-based personalized product description generation in e-commerce. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 3040–3050.
- Chen, S., Xie, X., Yin, B., Ji, Y., Chen, L., Xu, B., 2020. Stay professional and efficient: Automatically generate titles for your bug reports. In: 2020 35th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 385–397.

- Ding, N., Hu, S., Zhao, W., Chen, Y., Liu, Z., Zheng, H.-T., Sun, M., 2021. OpenPrompt: An open-source framework for prompt-learning. arXiv preprint arXiv:2111.01998.
- Dong, J., Lou, Y., Zhu, Q., Sun, Z., Li, Z., Zhang, W., Hao, D., 2022. FIRA: Fine-grained graph-based code change representation for automated commit message generation. In: 2022 IEEE/ACM 44th International Conference on Software Engineering. ICSE, IEEE.
- Ehsan, O., Hassan, S., Mezouar, M.E., Zou, Y., 2020. An empirical study of developer discussions in the gitter platform. ACM Trans. Softw. Eng. Methodol. (TOSEM) 30 (1), 1–39.
- Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al., 2020. Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155.
- Fleiss, J.L., 1971. Measuring nominal scale agreement among many raters. Psychol. Bull. 76 (5), 378.
- Fu, M., Tantithamthavorn, C., Le, T., Nguyen, V., Phung, D., 2022. VulRepair: a T5-based automated software vulnerability repair. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 935–947.
- Gao, Z., Xia, X., Grundy, J., Lo, D., Li, Y.-F., 2020. Generating question titles for stack overflow from mined code snippets. ACM Trans. Softw. Eng. Methodol. (TOSEM) 29 (4), 1–37.
- Gao, C., Zeng, J., Xia, X., Lo, D., Lyu, M.R., King, I., 2019. Automating app review response generation. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 163–175.
- GitHub, 2022. What is GitHub Discussions? A complete guide. URL: <https://resources.github.com/devops/process/planning/discussions/>.
- Gong, S., Li, M., Feng, J., Wu, Z., Kong, L., 2022. Diffuseq: Sequence to sequence text generation with diffusion models. arXiv preprint arXiv:2210.08933.
- Hata, H., Novelli, N., Baltes, S., Kula, R.G., Treude, C., 2022. GitHub Discussions: An exploratory study of early adoption. Empir. Softw. Eng. 27 (1), 1–32.
- Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., Gelly, S., 2019. Parameter-efficient transfer learning for NLP. In: International Conference on Machine Learning. PMLR, pp. 2790–2799.
- Iacob, C., Harrison, R., 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In: 2013 10th Working Conference on Mining Software Repositories. MSR, IEEE, pp. 41–44.
- Islam, M.J., Nguyen, G., Pan, R., Rajan, H., 2019. A comprehensive study on deep learning bug characteristics. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 510–520.
- Jiang, Z., Shi, L., Chen, C., Hu, J., Wang, Q., 2021. Dialogue disentanglement in software engineering: How far are we? arXiv preprint arXiv:2105.08887.
- Kenton, J.D.M.-W.C., Toutanova, L.K., 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT. pp. 4171–4186.
- Kummerfeld, J.K., Gouravajhala, S.R., Peper, J.J., Athreya, V., Gunasekara, C., Ganhotra, J., Patel, S.S., Polymenakos, L.C., Lasecki, W., 2019. A large-scale corpus for conversation disentanglement. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. pp. 3846–3856.
- Lester, B., Al-Rfou, R., Constant, N., 2021. The power of scale for parameter-efficient prompt tuning. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 3045–3059.
- Lin, C.-Y., Och, F., 2004. Looking for a few good metrics: ROUGE and its evaluation. In: Ntcc Workshop.
- Liu, K., Yang, G., Chen, X., Yu, C., 2022. SOTitle: A transformer-based post title generation approach for stack overflow. arXiv preprint arXiv:2202.09789.
- Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., Neubig, G., 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. arXiv preprint arXiv:2107.13586.
- Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., et al., 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. arXiv preprint arXiv:2102.04664.
- Pan, S., Bao, L., Ren, X., Xia, X., Lo, D., Li, S., 2021. Automating developer chat mining. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 854–866.
- Papineni, K., Roukos, S., Ward, T., Zhu, W.-J., 2002. Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. pp. 311–318.
- Parra, E., Ellis, A., Haiduc, S., 2020. GitterCom: A dataset of open source developer communications in gitter. In: Kim, S., Gousios, G., Nadi, S., Hejderup, J. (Eds.), MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29–30 June, 2020. ACM, pp. 563–567. <http://dx.doi.org/10.1145/3379597.3387494>.
- Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P., Bakhtin, A., Wu, Y., Miller, A., 2019. Language models as knowledge bases? In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 2463–2473.

- Phong, M.V., Nguyen, T.T., Pham, H.V., Nguyen, T.T., 2015. Mining user opinions in mobile app reviews: A keyword-based approach (t). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, pp. 749–759.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al., Language models are unsupervised multitask learners.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J., et al., 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21 (140), 1–67.
- Ravaut, M., Joty, S., Chen, N., 2022. SummaReranker: A multi-task mixture-of-experts re-ranking framework for abstractive summarization. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 4504–4524.
- Sabour, S., Zheng, C., Huang, M., 2022. Cem: Commonsense-aware empathetic response generation. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 11229–11237.
- Santhanam, S., Karduni, A., Shaikh, S., 2020. Studying the effects of cognitive biases in evaluation of conversational agents. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. pp. 1–13.
- Santhanam, S., Shaikh, S., 2019. Towards best experiment design for evaluating dialogue system output. In: Proceedings of the 12th International Conference on Natural Language Generation. pp. 88–94.
- Shen, Q., Ma, H., Chen, J., Tian, Y., Cheung, S.-C., Chen, X., 2021. A comprehensive study of deep learning compiler bugs. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 968–980.
- Shi, L., Chen, X., Yang, Y., Jiang, H., Jiang, Z., Niu, N., Wang, Q., 2021a. A first look at developers' live chat on gitter. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 391–403.
- Shi, L., Jiang, Z., Yang, Y., Chen, X., Zhang, Y., Mu, F., Jiang, H., Wang, Q., 2021b. ISPY: Automatic issue-solution pair extraction from community live chats. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, pp. 142–154.
- Shi, L., Xing, M., Li, M., Wang, Y., Li, S., Wang, Q., 2020. Detection of hidden feature requests from massive chat messages via deep siamese network. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. pp. 641–653.
- Silva, C.C., Galster, M., Gilson, F., 2022. A qualitative analysis of themes in instant messaging communication of software developers. *J. Syst. Softw.* 192, 111397.
- Song, Y., Chaparro, O., 2020. Bee: a tool for structuring and analyzing bug reports. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 1551–1555.
- Subash, K.M., Kumar, L.P., Vadlamani, S.L., Chatterjee, P., Baysal, O., 2022. DISCO: A dataset of Discord chat conversations for software engineering research. In: Proceedings of the 19th International Conference on Mining Software Repositories. pp. 227–231.
- Suhara, Y., Wang, X., Angelidis, S., Tan, W.-C., 2020. OpinionDigest: A simple framework for opinion summarization. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 5789–5798.
- Sun, X., Chen, S., Feng, Z., Ge, W., Huang, K., 2018. A service annotation quality improvement approach based on efficient human intervention. In: 2018 IEEE International Conference on Web Services, ICWS, IEEE, pp. 107–114.
- Tian, Y., Zhang, Y., Stol, K.-J., Jiang, L., Liu, H., 2022. What makes a good commit message? *arXiv preprint arXiv:2202.02974*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. *Adv. Neural Inf. Process. Syst.* 30.
- Wang, C., Nong, Z., Gao, C., Li, Z., Zeng, J., Xing, Z., Liu, Y., 2022a. Enriching query semantics for code search with reinforcement learning. *Neural Netw.* 145, 22–32.
- Wang, Y., Wang, W., Joty, S., Hoi, S.C., 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 8696–8708.
- Wang, C., Yang, Y., Gao, C., Peng, Y., Zhang, H., Lyu, M.R., 2022b. No more fine-tuning? An experimental evaluation of prompt tuning in code intelligence. <http://dx.doi.org/10.48550/arXiv.2207.11680>, CoRR abs/2207.11680. URL: <https://doi.org/10.48550/arXiv.2207.11680>, arXiv:2207.11680.
- Wiseman, S., Rush, A.M., 2016. Sequence-to-sequence learning as beam-search optimization. In: EMNLP.
- Xie, X., Su, Y., Chen, S., Chen, L., Xuan, J., Xu, B., 2021. MULA: A just-in-time multi-labeling system for issue reports. *IEEE Trans. Reliab.* 71, (1), 250–263.
- Yang, G., Liu, K., Chen, X., Zhou, Y., Yu, C., Lin, H., 2022. CCGIR: Information retrieval-based code comment generation method for smart contracts. *Knowl.-Based Syst.* 237, 107858.
- Yang, Y., Xia, X., Lo, D., Grundy, J., 2020. A survey on deep learning for software engineering. *ACM Comput. Surv.*
- Yu, T., Joty, S., 2020. Online conversation disentanglement with pointer networks. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP, pp. 6321–6330.
- Yu, W., Zhu, C., Li, Z., Hu, Z., Wang, Q., Ji, H., Jiang, M., 2022. A survey of knowledge-enhanced text generation. *ACM Comput. Surv.*
- Zhang, N., Liu, C., Xia, X., Treude, C., Zou, Y., Lo, D., Zheng, Z., 2022. ShellFusion: Answer generation for shell programming tasks via knowledge fusion.

Guodong Fan is currently pursuing a Ph.D. degree in Computer Science and Technology with the College of Intelligence and Computing, Tianjin University. He is working on Cognitive Services. His main research interests are Representation Learning and Service Computing.

Shizhan Chen Ph.D. Professor with the College of Intelligence and Computing, Tianjin University. He received the bachelor's degree in engineering and the Ph.D. degree in computer applications from Tianjin University, Tianjin, China, in 1998 and 2010, respectively. His research interests include service computing and service ecosystem mining and analysis.

Hongyue Wu Ph.D., associate professor with the College of Intelligence and Computing, Tianjin University. His research interests include service computing, mobile edge computing, and cloud computing.

Cuiyun Gao Ph.D., associate professor at the Harbin Institute of Technology. His current research interests include software repository mining and natural language understanding.

Jianmao Xiao assistant professor at Jiangxi Normal University. His main research interests include service computing and intelligent software engineering.

Xue Xiao Ph.D., professor with the College of Intelligence and Computing, Tianjin University. His research interests include service computing, computational experiment, and swarm intelligence.

Zhiyong Feng Ph.D., professor with the College of Intelligence and Computing, Tianjin University. He received the Ph.D. degree from Tianjin University, Tianjin, China, in 1996. He has authored more than 200 articles, one book and 39 patents. His research interests include service computing, knowledge engineering, and software engineering.