Empirical Software Engineering
This is a pre-copyedit version of this article.

Characterizing and Classifying Developer Forum Posts with their Intentions

Xingfang Wu · Eric Laufer · Heng Li · Foutse Khomh · Santhosh Srinivasan · Jayden Luo

Received: date / Accepted: date

Abstract With the rapid growth of the developer community, the amount of posts on online technical forums has been growing rapidly, which poses difficulties for users to filter useful posts and find important information. Tags provide a concise feature dimension for users to locate their interested posts and for search engines to index the most relevant posts according to the queries. Most tags are only focused on the technical perspective (e.g., program language, platform, tool). In most cases, forum posts in online developer communities reveal the author's intentions to solve a problem, ask for advice, share information, etc. The modeling of the intentions of posts can provide an extra dimension to the current tag taxonomy. By referencing previous studies and learning from industrial perspectives, we create a refined taxonomy for the intentions of technical forum posts. Through manual labeling and analysis on a sampled post dataset extracted from online forums, we understand the relevance between the constitution of posts (code, error messages) and their intentions. Furthermore, inspired by our manual study, we design a pre-trained transformer-based model to automatically predict post intentions. The best variant of our intention prediction framework, which achieves a Micro F1-score of 0.589, Top 1-3 accuracy of 62.6% to 87.8%, and an average AUC of 0.787, outperforms the state-of-the-art baseline approach. Our characterization and

Xingfang Wu, Heng Li, Foutse Khomh
Department of Computer Engineering and Software Engineering
Polytechnique Montréal
Montréal, Québec, Canada
E-mail: {xingfang.wu, heng.li, foutse.khomh}@polymtl.ca

Eric Laufer, Jayden Luo
Peritus.ai Canada Inc. Montréal, Québec, Canada
E-mail: {eric, jayden}@peritus.ai

Santhosh Srinivasan
Peritus.ai, Inc. Palo Alto, California, United States
E-mail: sms@peritus.ai

automated classification of forum posts regarding their intentions may help forum maintainers or third-party tool developers improve the organization and retrieval of posts on technical forums.

Keywords Developer Forum · Online Community · Intention · Tag Recommendation.

1 Introduction

Online technical communities such as Stack Overflow have been growing rapidly in recent years. By Apr 2023, more than 24 million posts and 35 million answers exist on Stack Overflow only[1], let alone a large number of posts on the innumerable private channels and dedicated forums that are not universal and general for all developers. The rapid growth of online developer communities demands more efficient and advanced approaches to managing content and providing users with more precise query results and accurate recommendations.

Tags often serve as a starting point for developers to investigate the topics discussed in forums (Barua *et al.*, 2014). Tags function as crucial meta information, aiding in the categorization of content (Maity *et al.*, 2019). This empowers users to efficiently filter out irrelevant posts, enabling them to swiftly refine their search. On the other hand, tags enable the recommendation of posts to specific user groups by aligning with their user portrait, which is constructed from their activity history (including browsing history, answered questions, etc.) associated with those specific tags, ensuring a more personalized content recommendation (Greco *et al.*, 2018; Guo *et al.*, 2008; Huang *et al.*, 2017). Efficient recommendations with tags have the potential to enhance the visibility of a question, increasing the likelihood of a swift response from domain experts (Yazdaninia *et al.*, 2021).

Most online technical communities provide users with pre-defined tags when posting. However, most of these pre-defined tags prioritize technical aspects, including programming languages, platforms, and tools (Beyer *et al.*, 2020; Chen *et al.*, 2019). For example, tags related to programming languages, such as JavaScript, Python, Java, C#, and PHP predominate among the most popular tags on Stack Overflow. [2] In some cases, users are allowed to input their customized tags. However, the quality of customized tags depends largely on users' level of expertise. Customized tags may suffer from improper granularity and redundancy (Maity *et al.*, 2019; StackOverflow, 2022).

These tags can sometimes be too wide-ranging or specific, leading to poor post distinction (StackOverflow, 2022). Moreover, the presence of similar or duplicated tags complicates the process of recommending posts practically, making it more challenging to provide accurate and useful post recommendations (Maity *et al.*, 2019). Therefore, tag recommendation approaches are

[1] https://stackexchange.com/sites?view=list

[2] https://stackoverflow.com/tags

researched and developed for online technical communities (Al-Kofahi *et al.*, 2010; Hong *et al.*, 2017; Li *et al.*, 2020; Liu *et al.*, 2018; Wang *et al.*, 2015; Zhou *et al.*, 2019). Most of these approaches are based on textual features of posts and adopt natural language processing to recommend potential post tags. These tag-recommendation approaches, which only focus on technical topics of posts, have achieved good performances both on public datasets and real application scenarios (He *et al.*, 2022).

However, a recent study (Beyer *et al.*, 2017) suggests that it may not be sufficient to only consider technical issues and topics when analyzing the questions proposed by developers. It stands a better chance of getting more insights into the posts if we investigate reasons why questions are asked (Allamanis and Sutton, 2013). These reasons can provide an extra dimension for developers to find solutions in online communities and support better recommendations of auxiliary tools for developers (Beyer *et al.*, 2020). In this paper, we refer to these reasons as *intentions*. To exemplify the distinctions between technique-oriented tag taxonomies and an intention-based taxonomy for technical posts, we present a concrete example. Suppose a novice programmer posts a question seeking a solution to address an error in their Python code related to data stored in an array data structure. Additionally, the programmer expresses a desire to find relevant tutorials to enhance their comprehension. The existing tag taxonomy would likely tag the question with 'python' and 'array' tags, emphasizing the technical aspects involved. On the other hand, the post embodies dual purposes: first, it serves as a request for assistance in resolving an *error* within the Python language; second, the programmer seeks *learning* resources about the related knowledge, particularly regarding data structures. Consequently, within an intention taxonomy, this post may be labelled with tags like 'error' and 'learning'. This distinction underscores the diverse nature of the questioners' intentions, extending beyond the limited technical categorization offered by the current tag taxonomy.

Previous works have proposed different taxonomies of online technical posts with different focuses and approaches (Allamanis and Sutton, 2013; Beyer and Pinzger, 2014; Beyer *et al.*, 2020; Treude *et al.*, 2011). However, most of the previous studies suffer from several drawbacks; in most studies, only posts from a single technical community are considered, some of which only focus on a single specialized domain (Beyer and Pinzger, 2014; Beyer *et al.*, 2020). As far as we know, most works are empirical studies carried out by researchers in academia. Therefore, a gap may exist between existing works and actual industrial practices. In our study, our primary goal is to narrow the gap by integrating industry insights into the construction of an intention detection approach for technical forum posts. To achieve this, we've outlined four key objectives. Firstly, we aim to **understand the distribution and placement of post content**, providing crucial insights for constructing our post-analysis workflow. Secondly, our focus is to **identify latent intentions** of technical posts and **their correlations with content distribution**, aiding us in crafting an intention taxonomy and detection approach that fits practical scenarios. Our third objective revolves around **crafting an intention taxonomy** that incorporates suggestions and needs from the industry. Finally, our fourth objective entails **constructing an intention detection approach** based on our refined taxonomy. To achieve these objectives, we base our study on a dataset collected from an in-use recommendation system and continually integrate feedback from our industrial collaborator.

We first conducted a qualitative study to understand the common posting practices in online technical communities. In the qualitative study, we examined a dataset which contains posts from different technical communities of different platforms. With the dataset, we manually look into the general composition of forum posts and the usage of different facilities (e.g., code block, image and etc.) supported by the platforms.

To obtain a taxonomy that can better serve the industrial application scenarios and use cases, we reviewed previous works and their suggested taxonomies. We evaluated the significance (e.g., the number of relevant posts found, usefulness regarding to industrial use cases) of existing intention categories. Based on this evaluation, significant intention categories were identified, which were reused and adapted accordingly. Our work is performed on a dataset of forum posts provided by our industrial partner that covers multiple developer communities (e.g., Stack Overflow, Discourse forums, etc.).

Furthermore, we manually annotate the intentions of posts following a rigorous process according to the resulting taxonomy of technical forum post intentions. Based on the findings and insights from the qualitative study, we propose an intention prediction framework for technical online posts. In the framework, we employ transformer-based pre-trained language models to generate embeddings for both title and description of posts. In addition to the textual descriptions of the posts, we also consider structural features such as the category of content contained in the code blocks. To improve the performance, we further fine-tuned the pre-trained model with our annotated dataset. To examine the effectiveness and get a better understanding of the intention prediction framework, we proposed the following research questions (RQs):

RQ1: Which pre-trained model (PTM) works best in our framework?

RQ2: Can our framework benefit from fine-tuning the PTMs? Compared with the baseline models, how effective is our intention detection framework?

RQ3: Can the content category of code blocks really help the detection of post intentions?

The major contributions of this work are as follows:

1. Our results from the qualitative study provide insights into the composition of technical posts and the correlation between the composition and the intention of posts. These findings can provide guidance for future approaches on technical forum post analysis.
2. We devise a post intention taxonomy by incorporating suggestions and needs from the industry. Based on the taxonomy, we construct and pub-

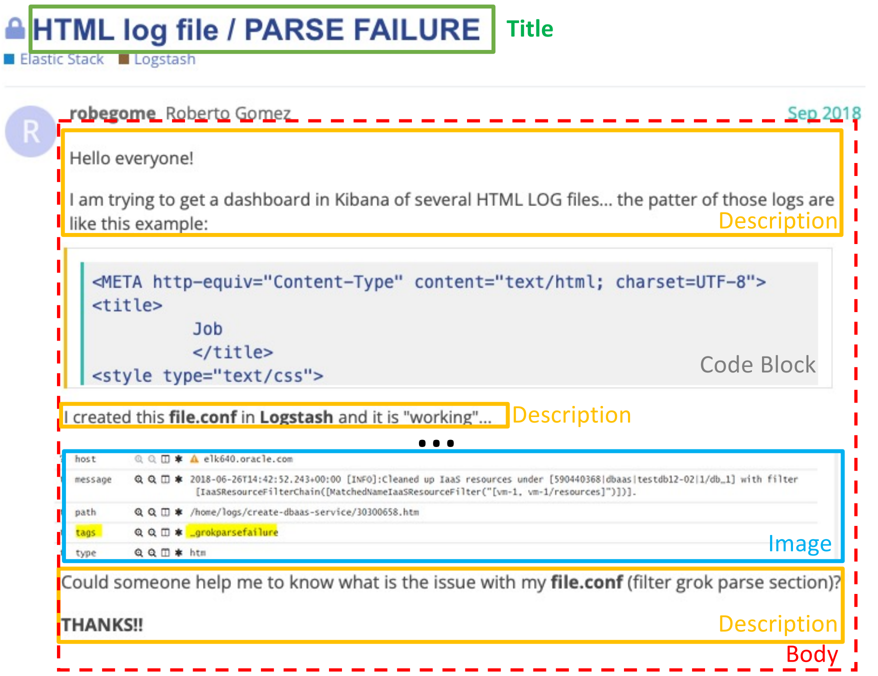
lish a technical post dataset with intention annotations, which future researchers and practitioners can utilize to build and evaluate their intention detection approaches.
3. We propose an intention detection approach, leveraging and fine-tuning pre-trained language models, which outperformed the baselines. To ensure reproducibility, we have made the code publicly accessible.
4. Our evaluation of six PTMs (including both general-purpose and domain-specific ones) in the task of intention detection provides guidance for choosing PTMs in processing technical forum post data. Together with the findings from the qualitative study, we provide some recommendations for practitioners when developing and using technical forums.
5. We provide insights for technological implementations and endeavours in industry-academia collaborations, drawing from lessons learned during the construction of our intention detection approach and the co-construction process with our industrial partner.

The paper is structured as follows: Section 2 introduces the qualitative study on online technical posts and our proposed taxonomy of post intentions. Section 3 specifies our approach to predicting post intention with textual and structural information of posts, and the details of our proposed framework and training process are detailed. In section 4, we evaluate our proposed framework by three research questions (RQs) and analyze the results. Section 5 presents insights from collaborative industry endeavours, highlights key findings from our experiments, and provides suggestions for forum users regarding posting behaviors. Section 6 identifies the threats to the validity of our study. Section 7 surveys related works. At last, we conclude and summarize our work in section 8.

2 Qualitative Study on Technical Posts and Their Intention

2.1 Posts in online technical communities

To provide some background for our study, we first briefly introduce the common post structure in online technical communities. While our study is not constrained to any particular online community or technical domain, technical posts typically adhere to a comparable structure and composition with minor deviations resulting from variations in platforms. Fig. 1 shows an example of posts on Discourse Forum. Generally, a post contains a short *title* which presents the main purpose or topic of the post. There are usually some detailed *descriptions* written in natural language in the *body* part of the post. *Code blocks*, *images*, or even *files* may be appended to a post as supplementary materials to provide a clearer picture for readers to understand the situation. However, not all of them are supported by the different platforms of online communities. These supplementary materials may contain various formats, varying by users with different usage habits. For example, code blocks are supposed to contain code snippets, while in reality, some users may not follow



HTML log file / PARSE FAILURE

Elastic Stack Logstash

robegome Roberto Gomez Sep 2018

Hello everyone!

I am trying to get a dashboard in Kibana of several HTML LOG files... the patter of those logs are like this example:

```
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>
          Job
          </title>
<style type="text/css">
```

I created this **file.conf** in **Logstash** and it is "working"...

• • •

```
host      ⚠ elk640.oracle.com
message   2018-06-26T14:42:52.243+00:00 [INFO]:Cleaned up IaaS resources under [590440368|dbaas|testdb12-02|1/db_1] with filter
          [IaaSResourceFilterChain([MatchedNameIaaSResourceFilter("[vm-1, vm-1/resources]")])].
path      /home/logs/create-dbaas-service/30300658.htm
tags      _grokparsefailure
type      htm
```

Could someone help me to know what is the issue with my **file.conf** (filter grok parse section)?

THANKS!!

Fig. 1 An example post from elastic.co, a Discourse-based online community.

the norms and put some descriptions in natural language in code blocks. Last but not least, most online platforms support *tags*, which helps organize the contents in the communities.

2.2 Overview of the Manual Study

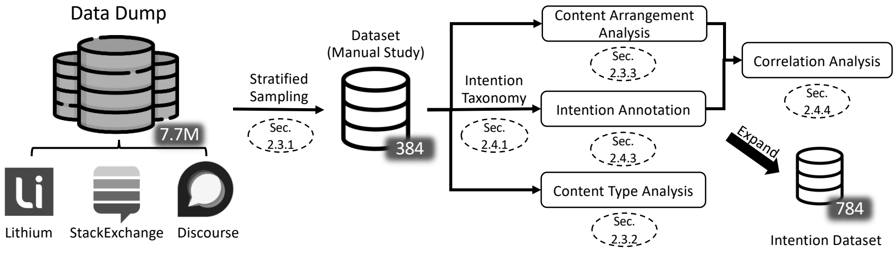


Fig. 2 An overview of our manual study process.

Figure 2 shows an overview of our manual study, with each step aligned to a corresponding section for easy navigation. Generally, the entire manual study belongs to the *Sample Study* (Stol and Fitzgerald, 2018), encompassing the examination of content types and intentions within our sampled dataset. Initiated with a data dump from our industrial partner, our manual study begins by sampling this dataset to create a representative subset. Within this subset, we first conducted a qualitative manual study on the content and its arrangement of technical posts, and we further studied the intentions of technical forum posts from an industrial point of view. For the content and arrangement study, we first developed a content type taxonomy through an *open coding* approach (Khandkar, 2009) and annotated the sampled posts accordingly. Utilizing the outcomes from the content type analysis, we delved further into understanding how these content types were positioned within posts, employing an additional manual annotation process. In our manual study of post intention, we also employed the *open coding* method. However, we scrutinized the sample dataset and consulted existing technical post taxonomies and industrial feedback to construct our intention taxonomy. Based on this taxonomy, we annotated the sample dataset, involving a systematic analysis and categorization of posts into intention categories. Finally, we explored correlations between the occurrence of content types and intentions, aiming to derive insights for designing intention detection approaches. We then expanded our annotated dataset to encompass 784 samples, essential for crafting our intention detection approach. In this section, we delve into each step of this process, detailing our methodology and findings.

2.3 A Manual Study of Content Types and Their Arrangement in Technical Forum Posts

In this section, we focus on an examination of the various types of content present in forum posts, such as code snippets, error messages, and images, and how these elements are arranged. Our focus on how these elements are organized aims to provide a clearer understanding of how information is typically structured in these forum posts. Insights from this analysis have the potential to guide the development of automatic intention detection approaches for technical posts, potentially enhancing their ability to interpret and categorize posts more effectively and discern questioners' intentions in online technical communities.

2.3.1 Dataset and Data Sampling

We constructed our studied dataset by sampling from a data dump provided by our industrial partner. The data dump was constructed during the time frame from June 25, 2020, to April 5, 2022. The dump contains primary posts (initial topic-setting posts) from different sources (i.e., online communities), mainly from three different platforms: Stack Exchange[3], Lithium[4] forums and

[3] A question-and-answer website that covers a wide range of topics and domains. The data dump only contains contents from selected technical subforums.

[4] A forum software developed by Lithium Technologies.

Table 1 Distribution of Sampled Posts by Data Source Updated.

Data source	Total	Sampled
Stack Exchange	4,058,490	198 (52%)
Lithium	2,694,643	137 (36%)
Discourse	948,580	49 (12%)

Table 2 Distribution of Content Types in the Post Dataset Updated.

Content	Percentage
Code	26.8%
Error message	15.9%
Image	10.4%
Config	8.9%
Command line	6.5%
Others	10.4%

Discourse[5] forums. The acquisition process of this data dump was facilitated by our partner's own web crawlers. Due to NDA, we could not share more details about the data crawling approaches. From around 7.7 million primary posts in the data dump, we adopted a stratified random sampling strategy to sample 384 posts in order to obtain precise estimates of the characteristics of posts from different platforms. The sampled amount is based on a 95% confidence level and a 5% margin of error (Boslaugh, 2012). It's worth noting that in our sample size calculation, we considered only the question posts from the data source. We did not include answer posts in our calculations.

However, during the manual annotation process, we found that 20 posts were no longer accessible online. As the dump does not contain some media (e.g., image), we could not conduct the content annotation for these posts. Thus, we randomly sampled an additional 20 posts that were available online during this research. Finally, we constructed a dataset with 384 posts, of which 198 were from Stack Exchange, 137 from Lithium and 49 from Discourse forums, as shown in Table 1. As the annotation for content types and their arrangement did not involve significant discrepancies, and any disagreements arose from mistakes made by raters, which were subsequently corrected by a third rater, we did not measure the inter-rater agreements for these two annotation process.

2.3.2 Manual Analysis of Content Types

Introduction The primary goal of this analysis is to gain insights into the utilization of various content types, such as code snippets and images, within technical forums.

Methodology The manual study of content types includes both annotation and result interpretation. Three authors (for example, A1, A2, and A3) participate in the annotation for content types of technical posts, involving the following three phases:

1. A1 and A2 went over the instances in the sample dataset together. Through the initial inspection which involved an *open coding* approach (Khandkar,

[5] An open-source forum software.

2009), the two authors summarized a list of content types that posts contain. And then, the two authors annotated 100 random samples collaboratively to reach a consensus on annotation standards.
2. A1 and A2 independently annotated sampled posts with the consensus reached by the discussion in Phase 1.
3. After finishing the individual works, A3 checked and summarized the results.

Results We count the occurrence of different types of content authors use to describe their issues. The content types mainly include natural language, code (both inline and multi-line code snippets), error message (stack trace, log, error output), image, command line and others. These types of content can appear in different parts of a post. For example, logs can appear both in the description and code blocks. It is worth noting that we count the occurrence of content no matter where they are in the posts, and we only count once if one type of content appears several times in a post.

All the posts in our dataset contain natural language to describe their issues. For other content types, we list the percentages of posts in Table 2. Code (both inline or multi-line) is the most common (26.8%) supporting content authors employ to provide extra information. 15.9% of posts contain Error messages (stack trace, log, error output) in their body. As authors often truncate long stack traces and log sequences to their posts, we cannot distinguish among these content types in detail in most cases. Therefore, we merge these as *Error message* here. Around 10% of posts contain images. Configs are usually given in markup formats (e.g., XML, JSON, etc.) in 8.9% of the posts. Only 6.5% of posts contain command lines. There are also some contents that do not belong to the previous categories, or there is not enough clue for us to recognize their types. We assign them in the *Others* class.

Finding 1: Code snippets are the most common supplementary content for programming-related posts. Besides code, program outputs (e.g., stack trace, log, etc.), configs, and command lines are utilized to provide additional information regarding the issues of posts.

2.3.3 Manual Analysis of Content Arrangement

Introduction Previously in Section 2.3, we examined the prevalent content types utilized by authors to articulate their programming issues. This analysis focuses on how the questioners arrange contents of different types in their posts (e.g., code is put in the code block or code is shown in a screenshot).

Methodology Based on the annotations obtained in the previous analysis process, we then delve deeper into understanding how these content types are positioned within posts. Since the annotation process for content type arrangements doesn't require complex judgments, it is relatively straightforward.

To maintain accuracy, both A1 and A2 independently annotated all samples. Following this, A3 reviewed the results to rectify any mistakes.

Results From our annotation of the code snippets, we found that most posts arrange their code snippets well: 90.6% of the posts which contain code snippets utilize the code block as the container. Among the misuses, most are the cases in which users did not use inline code elements to mark their short code snippets (e.g., variable name, function name). 33.3% of the posts that contain inline codes did not mark them correctly. Besides, only one post in our dataset utilizes a screenshot to present its code snippet. 6.0% of the posts contain a code snippet that is not in a code block. We also found that in some cases, authors of developer forum posts may use the code block as a blockquote, in which they tend to put words from other sources, outputs of the program and other texts in natural language.

Among the stack traces, which usually extend over multi-lines, 55.0% are arranged in code blocks, 25.0% are shown by screenshots, and only 20.0% are mixed with descriptions in natural language. However, error messages or fragments of outputs, which are shorter in length, are more often (65.7%) mixed with the description. Around half of the configs and command lines are arranged in code blocks (51.6% and 52.0%, respectively).

Finding 2: Authors of programming-related posts treat code blocks as a container and use them differently. Code blocks may contain various types of content other than code snippets. Authors tend to arrange their stack trace, configs, command lines and other programming-related textual information in code blocks.

2.4 Intentions of Technical Forum Posts

Previous works propose different taxonomies for technical posts (Allamanis and Sutton, 2013; Beyer and Pinzger, 2014; Beyer *et al.*, 2020; Rosen and Shihab, 2016; Treude *et al.*, 2011). These works have analyzed technical post categories and motivations from different angles, considering the particular technical fields and their corresponding contexts. Some of the categories resulted from these studies express or are closely related to the intended purposes of technical posts. By incorporating the use cases and suggestions from our industrial partner, we review and adapt existing categories proposed in previous works to our proposed taxonomy that focuses on the intention aspects of technical posts. In our taxonomy, we have seven intention categories, as follows. We assign a keyword to each of these intention categories. We will use these keywords to mention these intention categories hereinafter to enhance conciseness and clarity.

Table 3 Intention Categories of Online Technical Forum Posts

Intention Keywords	**Definition**	**Snippet Examples**	**Related Prior Definitions**
Discrepancy	Seeking explanations for software behavior discrepancies not explicitly related to errors	Any ideas what I am doing wrong here? [1] I don't understand why I cannot ping internet clients. [2] why won't my css or js apply in Firefox? [3]	Treude *et al.* (2011); Allamanis and Sutton (2013); Beyer and Pinzger (2014); Beyer *et al.* (2020)
Explicit Error	Seeking solutions for errors or exceptions	But I get error Unexpected null value. I can't handle it, have someone had similar problem? [4] It gives me this exception. [5] Does anyone know what this error means? [6]	Treude *et al.* (2011); Beyer and Pinzger (2014); Beyer *et al.* (2020)
Review	Looking for improved solutions or guidance to make well-informed decisions	I've completed ... exercise. Any feedback would be greatly appreciated. [7] Should I concatenate all certificates ... for ... directive in NGINX. [8] Here is how my service account is configured: ... if I am using kubectl auth can-i incorrectly. [9]	Treude *et al.* (2011); Beyer and Pinzger (2014); Beyer *et al.* (2020);
Conceptual	Seeking information or explanations without concrete implementations	What are BigQuery audit logs supposed to produce? [10] Is Terraform the official Infrastructure as code solution for IBM Cloud? [11] What is the gRPC++ equivalent of the Go context.Background()? [12]	Beyer and Pinzger (2014); Beyer *et al.* (2020)
Learning	Seeking learning resources for libraries, tools, or programming languages	If I could get a detailed guide or a link to an existing one that would be amazing. [13] I'm reading Vulkan Tutorial ... the "Subpass dependencies" section confused me a lot. [14] I can't seem to find any current documentation that discusses this. [15]	Allamanis and Sutton (2013); Beyer *et al.* (2017); Beyer *et al.* (2020)
How-to	Requesting step-by-step instructions for specific tasks	How to read utf16 text file to string in golang? [16] Workflow to clean badly scanned sheet music. [17] What do I need to do to make sure each group has its own directory ... [18]	Treude *et al.* (2011); Allamanis and Sutton (2013); Beyer and Pinzger (2014)

[1] StackOverflow (ID: 68442411) [2] Aruba Networks Community (ID: 438751) [3] StackOverflow (ID: 31944197) [4] StackOverflow (ID: 67894563) [5] StackOverflow (ID: 65468209) [6] Cisco Community (ID: 202789) [7] Mozilla Discourse (ID: 96262) [8] Server Fault (ID: 775101) [9] Server Fault (ID: 1019782) [10] StackOverflow (ID: 34302214) [11] StackOverflow (ID: 56739646) [12] StackOverflow (ID: 61408251) [13] Server Fault (ID: 971124) [14] StackOverflow (ID: 68004511) [15] Rancher Forums (ID: 5085) [16] StackOverflow (ID: 15783830) [17] StackOverflow (ID: 68401240) [18] Cisco Community (ID: 1967272)

2.4.1 Intention Taxonomy

Intention 1 (Discrepancy) Seeking explanations for software behavior discrepancies that are not explicitly related to errors. The posts of this category contain questions about problems and unexpected behaviors of systems, services or code snippets which the questioner has no clue how to solve. The problems or unexpected behaviors are not necessarily associated with errors or exceptions and could instead be related to user errors. In previous works, this type of posts are categorized as *Do not work* (Allamanis and Sutton, 2013) or *What is the Problem. . . ?* (Beyer and Pinzger, 2014). In works (Beyer *et al.*, 2020; Treude *et al.*, 2011), their taxonomies also have this category.

Intention 2 (Explicit Error) Seeking solutions for explicit errors or exceptions. This category addresses problems related to exceptions or errors. Often, error messages, exceptions, and stack traces are attached to posts, and the questioners usually ask for help in finding the root cause of an exception and the solutions to fix an error. This category differs from the *Discrepancy* category by primarily focusing on troubleshooting and resolving specific errors or exceptions encountered in software. Unlike the *Discrepancy* category, which deals with a broader range of unexpected behaviors, this category is specifically tailored to address issues that are directly manifested by errors or exceptions. Questioners in this category seek assistance in pinpointing the root causes of these errors and soliciting effective solutions for rectifying them. The inclusion of error-specific information and the nature of the inquiries set this category apart as a specialized resource for those encountering error-related challenges in developing or using software. It is a common category shared by many previous works (Beyer and Pinzger, 2014; Beyer *et al.*, 2020; Treude *et al.*, 2011).

Intention 3 (Review) Looking for improved solutions or guidance to make well-informed decisions. Typically, the questioners who ask questions in this category already have solutions for their problems. Their intentions are to validate their proposed decisions or to search for a better solution for accomplishing a task. Usually, authors will post their code snippet decisions for readers to review. Related categories proposed by previous works are *Decision Help* (Treude *et al.*, 2011), *Better Solution* (Beyer and Pinzger, 2014), etc. This category also exists in Beyer *et al.* (2020).

Intention 4 (Conceptual) Seeking background information, explanations, or a better understanding of subjects or technology aspects without concrete implementations. This category of posts usually contains questions about abstract or non-implementation level concepts, such as design patterns, background information, or limitations about some libraries or devices. In some cases, the authors want to know whether it is feasible to do something with tools, libraries, or other supplements mentioned (i.e., limitations of tools, libraries, etc.). In previous work (Beyer and Pinzger, 2014), this category is mentioned as *Is it possible. . . ?*. This category also exists in Beyer *et al.* (2020).

Table 4 Examples of posts belonging to the *Other* class

Intention	Definition	Post Snippet Examples
Requesting software resources	Seeking access to resources for immediate use or application.	Are there any plugins/tools available to . . . ? [1] Where can I download gcc . . . ? [2]
Announcing	Informing or sharing news, updates, or events without seeking deeper understanding or background information.	Release v0.46.0 New Features [3] We released new iOS versions for ... [4] Read all about this latest release in this blog ... [5]
Discussing a topic	Open-ended queries or topics aimed at sparking conversation, sharing opinions, or seeking input from a community.	Are there any plans to increase this? [6] WPA2 Vulnerability Discussion [7] Anyone know if it is worth upgrading to 4.1.3b? [8]
Reporting a problem or a bug*	Identifying, describing, and potentially addressing issues or bugs within software.	The word License is misspelled. [9] Settings place edit screen has a misspelled hint. [10]
...	...	...

[1] StackOverflow (ID: 248589) [2] HPE Community (ID: 2787835) [3] Rancher Community (ID: 995)
[4] Cisco Community (ID: 2571411) [5] HashiCorp Discuss (ID: 23223) [6] Paloalto Networks (ID: 39013)
[7] Aruba Networks Community (ID: 310066) [8] Cisco Community (ID: 1261498) [9] Cisco Community (ID: 4154656)
[10] Roblox Developer Community (ID: 705907)
* This category differs from *Explicit Error* and *Descrepancy* in its primary focus on reporting issues or bugs for attention, rather than seeking immediate solutions for specific errors or unexpected behaviors.

Intention 5 (Learning) Seeking learning resources. This category usually features requests for documentation or tutorials on a specific library, tool, or programming language. Compared with *How-to* posts, posts in this category usually do not focus on a specific question and ask for solutions or instructions. Instead, they are seeking for support to learn on their own. This category is also proposed in Beyer *et al.* (2020), and is the combination of *Learning a Language/Technology* (Allamanis and Sutton, 2013) and *Tutorials/Documentation* (Beyer *et al.*, 2017).

Intention 6 (How-to) Requesting specific, step-by-step instructions for particular tasks. This post category mainly asks for concrete instructions for a specific application scenario or a particular task to fulfill. This category subsumes post type *API usage* or *Interaction of API classes* proposed in Beyer *et al.* (2020) and Beyer *et al.* (2017), respectively. Other works (Allamanis and Sutton, 2013; Beyer and Pinzger, 2014; Treude *et al.*, 2011) also have similar or equivalent types for this category of posts.

Intention 7 (Other) Other intentions. We noticed some technical forum posts that did not fit into common categories. However, creating specific categories for them may be unproductive and could undermine recommendation systems' effectiveness, based on the feedback from our industrial partner, as the number of these posts is not significant. We group extra categories under *Other* in our work. Table 4 presents a list of example intentions and example post titles that belong to the *Other* category.

2.4.2 Manual Study of Post Intention

To further investigate the intentions behind technical QA posts and understand the correlation between post structure and intention, we conducted a manual study of post intention. This manual study process involves both annotation and result interpretation. Three authors (for example, A1, A2, and A3, including an expert from our industrial partner) participate in annotating the intentions for technical posts. Importantly, each intention is not exclusive,

Table 5 Results of Intention Annotation

Intention	Number of post
Discrepancy	149
Explicit Error	150
Review	86
Conceptual	159
Learning	23
How-to	273
Other	86

meaning one post can contain more than one intention. For instance, authors may request a solution to an error (i.e., *Explicit Error*) while simultaneously seeking help to understand a related abstract concept (i.e., *Conceptual*). Therefore, in the manual annotation process, we assigned multiple labels for posts with more than one intention. Similar to the manual analysis process described in Section 2.3.2, we adopted an open coding approach. We began by examining a set of sampled posts to identify recurring intentions. For example, we noticed a recurring theme of posts seeking help to find tutorials or other learning materials, which we labeled as *Learning* and added to our initial intention categories. This iterative process enabled us to refine and categorize intentions progressively based on the content and context of the posts. The annotation typically involves the following three phases:

1. The authors summarized a list of intentions and collaboratively annotated 100 random samples to establish an annotation consensus.
2. A1 and A2 independently annotated sampled posts with the consensus reached by the discussion in Phase 1.
3. After finishing the individual works, A1 and A2 compared the results, inter-rater agreements were measured, and any disagreement regarding the annotation was discussed to reach agreements. If the two authors could not reach a consensus, A3 got involved in the discussion, and the three authors voted and made the final decisions.

Besides the sampled posts, we further annotated more posts in our data dump to acquire more training and test data for our proposed framework for automatically detecting post intentions (in Section 3). At last, we were able to extend our intention annotation dataset to the size of 784.

2.4.3 Inter-rater Agreement

We measured the inter-rater agreement between two coders using Krippendorff's Alpha score (Krippendorff, 2011) for the outcomes of Phase 2. Krippendorff's Alpha, a standard and flexible coefficient for measuring inter-coder agreement, takes the form of:

$$\alpha = 1 - \frac{D_o}{D_e} \tag{1}$$

where D_o represents the observed disagreement among coders, while D_e signifies the disagreement anticipated by chance. The scale, ranging from -1 to 1, signifies the level of agreement among raters, where -1 indicates perfect disagreement, 0 implies no agreement beyond chance, and 1 denotes perfect agreement. Since our annotation involves multiple intention classes, we decomposed a multi-category observation as multiple binary observations and conducted the analysis as if it were a binary scenario. Following Phases 1 and 2, the Krippendorff's Alpha coefficients for all intention categories range between 0.62 and 0.81, indicating moderate to good agreements. All three authors achieved agreement on every sample following Phase 3. Hence, our manual labeling for intentions can be considered trustworthy.

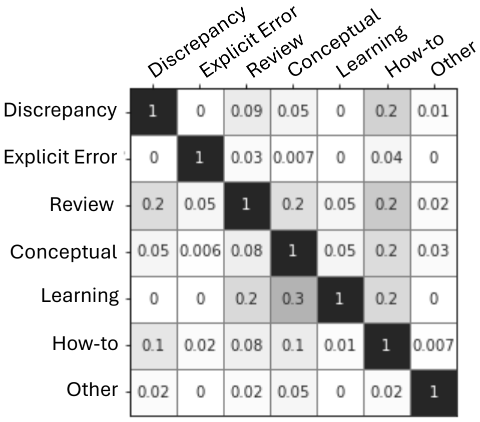
2.4.4 Results of Intention Annotation

The numbers of posts that belong to each intention class are shown in Table 5. We assigned 83% of the posts with one label, 16% with two labels, and only 1% with three labels. *How-to* is the most common intention, which accounts for 34.8% of the posts. *Review* and *Learning* are the two least frequent intentions, which only occur in 11.0% and 2.9% of the posts, respectively. Moreover, we further counted the co-occurrence of the types of intentions when posts contain more than one type of intention. The co-occurrence matrix is shown in Figure 3. As the number of posts is unevenly distributed in the seven types, we divide each row of the original co-occurrence matrix by the number of posts with the intention corresponding to that row. Thus, an element in row i, column j shows the percentage of posts with intention i that also have intention j. By summing up the elements in each row of the matrix, we can find that 69.6% of the *Learning* posts and 67.4% of the *Review* posts have other intentions. *Learning* posts are usually also *Conceptual*, *Review* or *How-to* posts. *Review* posts are likely to be *Discrepancy*, *Conceptual* or *How-to* posts. These co-occurrences are natural and reasonable. For example, when developers encounter an unexpected behavior of a program (*Discrepancy*), they may provide their code snippets or operations for readers to check (*Review*).

Finding 3: Posts may have more than one type of intention. *How-to* is the most common type of intention, while the number of posts is unevenly distributed in the seven types of intentions.

2.4.5 Correlations Between the Occurrence of Certain Content Types and Post Intentions

From Table 6, we can find differences in the distributions of supplements among posts with different intention types. 65.8% of Review posts and 50% of Explicit Error posts have posted codes regarding their issues. The ratio is significantly lower for posts with other types of intention. As the nature of



	Discrepancy	Explicit Error	Review	Conceptual	Learning	How-to	Other
Discrepancy	1	0	0.09	0.05	0	0.2	0.01
Explicit Error	0	1	0.03	0.007	0	0.04	0
Review	0.2	0.05	1	0.2	0.05	0.2	0.02
Conceptual	0.05	0.006	0.08	1	0.05	0.2	0.03
Learning	0	0	0.2	0.3	1	0.2	0
How-to	0.1	0.02	0.08	0.1	0.01	1	0.007
Other	0.02	0	0.02	0.05	0	0.02	1

Fig. 3 The co-occurrence matrix of intentions. Each row is divided by the number of posts of the corresponding intention.

Table 6 Distribution of content types by intention types.

Intention	Code	Error Message		Config	Command line
		Error text	Stack trace		
Discrepancy	43.0%	9.3%	1.2%	**15.1**%	**9.3%**
Explicit Error	50.0%	**46.9**%	**26.6**%	10.9%	7.8%
Review	**65.8**%	2.6%	0.0%	7.9%	5.2%
Conceptual	33.3%	1.2%	1.2%	3.7%	6.2%
Learning	11.1%	0.0%	0.0%	0.0%	0.0%
How-to	33.3%	2.6%	0.9%	6.8%	5.1%
Other	4.0%	3.8%	0.0%	0.0%	3.8%

Explicit Error intention, posts of this type are more likely to have error texts or stack traces as their supplements. 46.9% and 26.6% of this type of posts contain error texts and stack traces separately. Also, we found that authors of Discrepancy posts are more likely to post their configurations for readers to address their issues.

Finding 4: There exists a correlation between posts' intention types and their supplementary resources, which may serve as a feature for detecting the intentions of posts. The existence of different types of content may serve as a feature for detecting the intentions of posts.

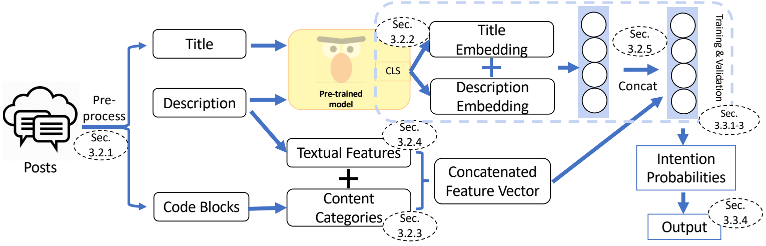


Fig. 4 An overview of our intention detection framework. The section numbers in the dashed circles correspond to the respective descriptions.

Summary

In addition to natural language descriptions, technical posts often contain different types of supplementary content (e.g., code, stack trace, etc.). Authors tend to arrange all these in code blocks. One technical post may have multiple intentions. We observed a correlation between the presence of specific content types and the intentions.

3 Automatically Detecting Post Intentions

Inspired by the findings from our manual study and previous works, we propose a framework to detect the intentions for technical QA forum posts automatically. In this section, we describe in detail the proposed framework, which is based on a transformer-based PTM and formulates the process of detecting post intentions as a multi-class multi-label classification problem.

3.1 Overview of the Framework

The overall structure of our proposed intention detection framework is illustrated in Figure 4. Generally, the framework contains three processing stages: data pre-processing, feature extraction, and classification. During the pre-processing stage, we remove unexpected tokens (e.g., HTML tag) from the raw forum data. Then, in the feature extraction stage, we use a PTM as an encoder to generate embeddings for the title and description of posts. The two embeddings are merged by a fully connected layer, the output of which is concatenated with a feature vector. The feature vector contains two parts of features: the content feature of code blocks and the textual features of the description of posts. The content feature is generated with a code block classifier, and textual features are generated with different metrics. Finally, the concatenated features are fed into a fully connected layer which outputs the classification results.

3.2 Data Pre-processing & Feature Extraction

3.2.1 Pre-processing

According to the design of the online communities, posts may contain different HTML tags or other platform-specific tokens for the front-end formatting and presentation of the content. Code blocks are usually embedded in the *Body* of the posts with specific tags (i.e., <pre><code>...</code></pre> in Stack Overflow posts). In the pre-processing stage, we extract the content of code blocks and remove platform-specific tokens in the *Body* of posts, which can be noise to the input of the PTM. Typical pre-processing methods such as eliminating stopwords, performing stemming, and lemmatization are frequently utilized in natural language processing but are not mandatory for contextual embedding techniques. Stop words and declensions can sometimes provide contextual information for the model to better present the semantic information of texts, removing them may lead to a loss of information. Transformer-based PTMs can effectively manage variations in word forms and map them to continuous vector representations. A previous study has demonstrated that applying stopword removal has no effect on performance in their task Qiao *et al.* (2019). Therefore, we exclude these preprocessing strategies in our workflow as we adopt PTMs in our model.

3.2.2 Generating embeddings with pre-trained models

Bidirectional Encoder Representations from Transformers (BERT) (Devlin *et al.*, 2018) is a transformer-based architecture that is capable of capturing long dependency in natural language, and various transformer-based PTMs have been achieving state-of-the-art results in different natural language tasks (Jin *et al.*, 2020). Besides, different PTMs which inherit the BERT architecture are developed and trained with program-related data to fulfill the tasks in software engineering and achieve promising results (e.g., CodeBERT (Feng *et al.*, 2020), BERTOverflow (Tabassum *et al.*, 2020)). In our proposed intention detection framework, we employ PTMs released in the Hugging Face (Wolf *et al.*, 2019) to generate contextual embeddings for natural language content in QA posts.
Maximum Input Length Due to the significant degradation in the performance of the BERT model in terms of the speed and accuracy of representing long documents, the authors of BERT set a limit to the input length of 512 sub-tokens (Devlin *et al.*, 2018). Sequences longer than the limit should be truncated. Choosing a proper maximum input length suitable for the data is essential for the framework's performance in terms of speed and accuracy. In our framework, we feed the title and description separately to the tokenizer, followed by a PTM. All online communities have their limits for the post title, thus titles are of limited length. However, the length may vary in the description of posts. We found that most posts have a description part of fewer than 200 tokens. The average, median and maximum lengths are 112, 83 and 1168, respectively, in our dataset. Therefore, we set the maximum input sequence

length as 256. For sequences longer than 256 tokens, we adopt the head-only truncation. The description refers to the preprocessed *Body* of posts, in which unexpected tokens and code blocks are removed. Therefore, we census the description length in our sampled dataset. The distributions of description lengths are shown in figure 5.

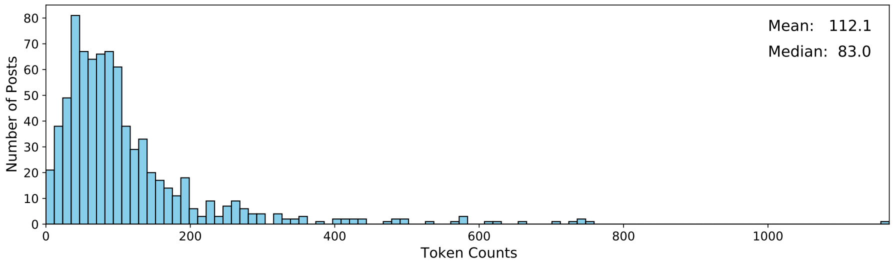


Fig. 5 The distributions of description lengths of posts in our dataset.

Pooling Strategy After feeding the tokenized title and description to the PTM, word embeddings are generated for tokens. To acquire embeddings to represent the title and the description, we need to aggregate the embeddings with a pooling strategy. The common ways for that include: (1) using the output of the first <*CLS*> token, (2) applying average or max pooling across each dimension of last hidden state embeddings (Reimers and Gurevych, 2019), (3) applying pooling on a concatenation of last few layers (Devlin *et al.*, 2018). However, there is no clear guideline on which pooling strategy should be used in all application scenarios and all PTMs (Devlin *et al.*, 2018; Reimers and Gurevych, 2019). We choose to use the output of the first <*CLS*> token as we will further fine-tune the models. At last, we concatenate the two embedding vectors for the post title and description and feed them into a fully-connected layer.

3.2.3 Content of Code Blocks

From our qualitative study, we find that there exists a close correlation between the intentions and certain types of content that posts may have. Code blocks are the most common container users may employ to drop supplementary resources that are in different formats or forms besides code snippets. As code block frequently appears in technical posts, utilizing the content types of code block as a feature may improve the performance of intention detection for technical posts.

However, as indicated by our qualitative study, code blocks can be used in different ways to present information. Moreover, previous works (Li *et al.*, 2020; Wang *et al.*, 2015) consider code snippets from posts in online communities to be of low quality. These factors undermine the effectiveness of directly leveraging the code block content to help represent QA posts. Therefore, we propose to use the content categories as a feature for intention detection. According to the findings from our qualitative study (in 2.3), we consider the content categories that frequently appear in the code blocks of QA posts.

To automatically detect the content categories (i.e., natural language, code, error message, config, command line, and others) of code blocks, we constructed a multinomial Naive Bayes classifier. Our approach utilizes regular expressions to tokenize texts from code blocks, distinguishing identifiers, operators, and brackets. We then employed TF-IDF (term frequency-inverse document frequency) to transform tokens into numerical arrays, representing each token frequency across the dataset. Based on the document-term matrix of textual data in code blocks, we trained the classifier.

We constructed a code block dataset for training and evaluating of the classifier by sampling and annotating code blocks from our data dump. The code block dataset has 10k samples in total, which are unevenly distributed in different classes (i.e., natural language, code, error message, config, and command line). We randomly splitted the dataset into an 80% training set, a 10% validation set, and a 10% test set. We adopted grid-search to tune the hyperparameter (i.e., Additive smoothing parameter) of the Multinomial Naive Bayes classifier using the evaluation set. We utilized SMOTE resampling (Chawla *et al.*, 2002) to address class imbalance during the training process. Utilizing the classifier's probability outputs across predefined content types, we assessed accuracy by considering classes with probabilities surpassing 0.5. A correct prediction was registered when one or more classes aligned with the ground truth. The classifier achieved an accuracy of 83.3% on the test set.

The probability outputs serve as a feature of posts for the model to detect intentions. Notably, we concatenate all the texts in all code blocks if a post contains more than one code block. As detecting content in code blocks is not the main focus of this work, we do not go into details here. The implementation of feature extraction and the classifier is included in our replication package.

3.2.4 Other features

Beyer *et al.* (2020) constructed QA posts intention classifiers, and their experiments showed that some textual features were beneficial for the recognition of certain intentions of posts. Thus, we incorporate the features (i.e., Word Count, Readability and Sentiment) identified in their study to improve our performance.

3.2.5 Feature Fusion

We concatenate the embeddings of the title and description and feed the feature vector with 768×2 dimensions to a fully connected layer. Other features are also concatenated and then merged with a fusion layer. Finally, an output layer (see 3.3) is followed and outputs the probabilities of intentions.

3.3 Model Training and Inference

3.3.1 Multi-label Loss Function

As we formulate the intention detection task as a multi-label multi-class classification task, we use a Sigmoid function as our output layer and adopt the Binary Cross Entropy loss (BCE loss) for each output node, which is between the target and the predicted probabilities. Therefore, the loss function for our model is the summation of the BCE losses of all output nodes over a batch of training data.

3.3.2 Training & Fine-Tuning

In our experiments, we adopt two different training settings according to the research questions we proposed. The first setting is to freeze the parameter of the PTM and train a classifier based on the embeddings and other features of the posts. The second setting allows updates to the parameters of the pooler layer of the PTM and assigns different learning rates for different components of the framework. The details can be found in the next section.

3.3.3 Cross-validation

We used five-fold cross-validation to counter the limited size of the dataset, aiming to enhance the reliability of our evaluation. We randomly divided our annotated dataset into five folds. For each iteration, we used one fold as the test set and the other four folds as the training set. We randomly separated 1/8 of the training set as our validation set, which was used to calculate the loss for guiding the early stopping. In total, we have 784 samples. For each iteration, we used 157 samples as the test set, 502 as training data, and 125 as the validation set. This approach enabled us to assess models using all available data. Even though the test set of each interaction is relatively small, the test sets of the five iterations combined cover all the 784 instances in the dataset. Our final evaluation result is the aggregated performance over the combined test sets of the five iterations, increasing the reliability of our evaluation result.

Each iteration had one fold of data as the test set and the other four folds as the training set, with a randomly selected 1/8 of the training set used for validation and to calculate the loss for early stopping. This allowed us to cover all 784 instances and improve the final evaluation's reliability.

3.3.4 Prediction Refinement

We map the output to categories with a threshold of 0.5 when evaluating our model. However, we made some adjustments to the predicted labels to make them more reasonable. First, we find that there exist cases when the output probabilities of all classes are under the threshold. In such cases, we force the model to output at least one label by assigning the class with the highest probability as the detection result. Second, we eliminate any other predicted labels when the probability of the *Others* class exceeds the threshold. This is because when the content covers multiple aspects of a programming topic or issue, the model may produce several output labels other than *Others*.

4 Evaluation

This section first introduces the dataset and metrics we use to evaluate our proposed intention detection framework. Organized along three research questions (RQs), we describe our experiments, aiming to have a better understanding of the characteristics of our proposed framework. Moreover, we analyze and summarize the results and findings.

4.1 Evaluation Metrics

As we formulate our intention detection task as a multi-class multi-label classification problem, we follow previous works on tag recommendation (He *et al.*, 2022; Li *et al.*, 2020; Zhou *et al.*, 2017) to use *Precision@k, Recall@k, F1-score@k* to evaluate the performance of our approach. However, as our baseline models do not predict the probability for each class, we can not apply these metrics to them, which hinders the direct comparison with the baseline models. Therefore, we further employ the *Micro F1 score* to get the overall performance over all classes, considering that posts of different categories take up different proportions of our dataset.

Precision@k, Recall@k, F1-score@k evaluate the tag recommendation approaches on their performance predicting top-k tags. Our qualitative study found that the posts usually have less than three intentions. Therefore, we set the value of k to 3. *Precision@k* is the average ratio of the correctly predicted tags among the top-k labels. *Recall@k* is the ratio of correctly predicted top-k tags among the ground truth tags. As the value may be capped to be small, a modification is made to the equation when the k is smaller than the number of ground truth tags. And, *F1-score@k* is the harmonic mean of *Precision@k* and *Recall@k*.

For each sample, the $Precision@k_i$ is defined by Equation 2 and we average the value for all samples and get Equation 3.

$$Precision@k_i = \frac{\left|Tag_i^{Pred} \cap Tag_i^{GT}\right|}{k} \tag{2}$$

$$Precision@k = \frac{\sum_{i=1}^{n} Precision@k_i}{n} \tag{3}$$

Recall@k is defined by Equation 4 and Equation 5.

$$Recall@k_i = \begin{cases} \frac{\left|Tag_i^{Pred} \cap Tag_i^{GT}\right|}{k} & , \left|Tag_i^{GT}\right| > k \\ \frac{\left|Tag_i^{Pred} \cap Tag_i^{GT}\right|}{\left|Tag_i^{GT}\right|} & , \left|Tag_i^{GT}\right| \leq k \end{cases} \tag{4}$$

$$Recall@k = \frac{\sum_{i=1}^{n} Recall@k_i}{n} \tag{5}$$

, which is defined by:

$$F1 - score@k_i = \frac{2 \times Precision@k_i \times Recall@k_i}{Precision@k_i \ + Recall@k_i} \tag{6}$$

$$F1 - score@k = \frac{\sum_{i=1}^{n} F1 - score@k_i}{n} \tag{7}$$

Micro Precision, Recall and F1-score are commonly used to access the performance of a multi-class classifier when there exists more than one class and need to aggregate in some way. As our dataset is unbalanced, and the number of posts of different intention categories varies, we do not employ macro averaging for the scores. Micro average aggregation uses the normal version of scores by calculating with total numbers of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) over all classes instead of for each class.

Area Under Curve (AUC) measures the degree of separability. It reflects the capability of a classifier to distinguish between classes. In our experiments, we adopt the one-vs-one configuration to compute the average AUC for all pairwise combinations of classes (Hand and Till, 2001).

Top-K accuracy takes the k predictions with the highest probability to calculate the accuracy. It measures how likely the true label appears in the top-k predictions.

4.2 Research Questions

We propose the following three research questions (RQs) to assess the performance and understand the characteristics of the proposed framework.

RQ1: Which pre-trained model (PTM) works best in our framework?

Motivation The transformer-based PTMs have been achieving promising results on various natural language (Wolf *et al.*, 2019) and computer vision tasks (Carion *et al.*, 2020; Dosovitskiy *et al.*, 2020). The BERT architecture (Devlin *et al.*, 2018) gains great popularity due to its ability to achieve outstanding performance on various natural language processing tasks when trained with massive data. Prior work has applied different variants of BERT in software engineering tasks, such as tag recommendations for Stack Overflow posts (He *et al.*, 2022).

However, the efficacy of using different PTMs varies according to the specific tasks and data according to previous works (Von der Mosel *et al.*, 2022; Yang *et al.*, 2022). And according to evaluations from previous works (Yang *et al.*, 2022), domain-specific PTMs do not necessarily have better performances on domain-specific tasks. We do not have general guidance on what specific PTMs should be used in our circumstances. Therefore, in RQ1, we aim to comparatively evaluate the performance of our intention detection framework with the different PTMs.

Approach In this RQ, We compare the performance of six variants of our intention detection framework with transformer-based PTMs.

Basically, the BERT architecture contains an encoder stack of transformer blocks. The original BERT is released in two sizes. We use the **BERT$_{\mathbf{base}}$** in the experiment. The BERT$_{base}$ has 12 layers of transformer block with a hidden unit size of 768 and 12 self-attention heads in the encoder stack. In total, it contains 110M parameters and is trained with a large corpus of English data. In the following, we will be using BERT to denote the BERT$_{base}$ model.

Most other transformer-based PTMs inherit BERT architecture while adopting different training settings (e.g., tasks, hyper-parameters, data, etc.) to train according to their specific application scenarios. **RoBERTa** (Liu *et al.*, 2019) modified some hyper-parameters and training tasks while maintaining the original BERT architecture. **ALBERT** (Lan *et al.*, 2019) further improve the original BERT by adopting parameter reduction techniques. **DistilBERT** (Sanh *et al.*, 2019) is a distilled version, which has 40% fewer parameters while maintaining over 95% of the BERT model. To process domain-specific texts in software engineering, which contain technical jargon that can not be properly processed by general language models, **BERTOverflow** (Tabassum *et al.*, 2020) is proposed with a named entity recognition technique. It is trained with sentences from Stack Overflow and can achieve better performance on domain-specific tasks. Further, there are also PTMs targeting software engineering tasks. Pre-trained with both natural language corpus and programming language data, **CodeBERT** (Feng *et al.*, 2020) is able to generate embeddings for both forms of input data. It has been achieving promising results on several software-related downstream tasks (Huang *et al.*, 2021; Mashhadi and Hemmati, 2021).

We compare the performances of six variants of our framework with the PTMs mentioned above. We leverage the PTMs released in the online community Hugging Face (Wolf *et al.*, 2019) in our experiments. We use the pooler output of the PTMs, which corresponds to the representation of the first token. Since DistilBert is not pre-trained with the next sentence prediction task, there is no pooler output layer. Instead, we use the output of a linear classification head. During the training process, we fixed all parameters of the PTMs, and only updated the parameters of layers on top of the PTMs.

Results and Analysis Table 7 shows the results of our experiments on different variants of our proposed intention detection framework with different PTMs. In terms of the Micro F1-score, the variants with BERT and RoBERTa models achieved the same performance (F1-score of 0.522) and outperformed other variants. It may be worth noting that, **as a multi-class multi-label problem with seven classes, it is generally significantly more difficult to make accurate classifications than binary classification scenarios (Sahare and Gupta, 2012).** The worst performance was achieved by the variant with BERTOverflow, a domain-specific PTM trained with Stack

Table 7 Comparison of variants of our framework with Micro F1-score. The highest scores and best variants are shown in **bold**.

Variant	Micro averaging		
	Precision	Recall	F1-score
BERT	0.571	**0.480**	**0.522**
RoBERTa	**0.597**	0.465	**0.522**
ALBERT	0.465	0.371	0.413
DistilBERT	0.576	0.454	0.508
BERTOverflow	0.402	0.295	0.340
CodeBERT	0.567	0.435	0.492

Table 8 Comparison of variants of our framework with different PTMs with *Precision@k*, *Recall@k* and *F1-score@k*. The highest scores and best variants are shown in **bold**.

Variant	Precision			Recall			F1-score		
	@1	@2	@3	@1	@2	@3	@1	@2	@3
BERT	0.575	0.448	0.363	0.575	0.673	0.802	0.575	0.523	0.485
RoBERTa	**0.588**	**0.486**	**0.385**	**0.588**	**0.740**	**0.864**	**0.588**	**0.571**	**0.518**
ALBERT	0.459	0.383	0.332	0.459	0.576	0.736	0.459	0.447	0.445
DistilBERT	0.570	0.465	0.375	0.570	0.706	0.836	0.570	0.545	0.503
BERTOverflow	0.397	0.354	0.320	0.397	0.546	0.724	0.397	0.418	0.432
CodeBERT	0.561	0.452	0.370	0.561	0.688	0.831	0.561	0.530	0.498

Overflow data, with a Micro F1-score of 0.340. The result seems counter-intuitive as its pre-training data is most relevant to our task and dataset. However, previous works (He *et al.*, 2022; Yang *et al.*, 2022) also found that this domain-specific PTM may perform worse than other general-purpose counterparts. The possible explanation may be that general-purpose PTMs were usually trained with larger data and have a better generalization ability. Compared with BERTOverflow, another domain-specific PTM (i.e., CodeBERT) achieved a moderate result, only inferior to the best ones by 5.7%. This is likely due to the similarity between our input data and the training data of CodeBERT, as our input texts are sometimes a mixture of natural language and truncated codes, while CodeBERT is trained with both natural and programming languages. We also observed a performance loss when PTMs with fewer parameters were used: the distilled version of BERT (i.e., DistilBERT) compared with BERT. The variant with ALBERT was outperformed by the best two by 26.4% in terms of Micro F1-score.

The results with F1-score@k scores further confirm the different performances. From Table 8, we find that the variant with RoBERTa consistently outperformed others in terms of all F1-score@k, which indicates the predictions of this variant are of higher quality. However, the difference is insignificant between the two best variants: the differences of F1-score@k are 0.013, 0.048 and 0.033 when k is 1 to 3, respectively. The AUC score and Top-k accuracy show a similar trend as the F1-score, so we do not include them in the table.

Answer to RQ1

Our intention detection framework achieves the best performance with the BERT variants **RoBERTa** and **BERT**. Generally, general-purpose PTMs work better than domain-specific counterparts in our intention detection framework, as they may be trained with larger data. PTMs with fewer parameters may suffer a performance loss.

RQ2: Can our framework benefit from fine-tuning the PTMs? Compared with the baseline models, how effective is our intention detection framework?

Motivation In this research question, we have two goals: The first objective is to examine if the performance of our approach can be further improved by fine-tuning the PTMs with the intention detection task. We chose two baselines for our study. The first one, proposed by Beyer *et al.* (2020), uses a set of random forest binary classifiers for QA post intention detection. The second baseline is a convolution neural network (CNN)-based approach from Huang *et al.* (2020) which is designed for extracting intentions from GitHub issue reports.

Approach In RQ1, we fixed the parameters of PTMs and only updated the layers on top of them in backpropagation to examine the effectiveness of various PTMs. To answer this research question, we further fine-tune the pooler layer in the two best-performing PTMs (e.g., BERT and RoBERTa). We then compare the fine-tuned models with our baseline approach. As the taxonomy for intentions from the baseline approaches differs from that of this work, we cannot directly compare the classification results. Therefore, we follow the implementations from the previous work and evaluate the approaches on our annotated dataset. We evaluate the baseline approaches with the same cross-validation process mentioned in Section 3.3.3.

Baseline 1: Random Forest Binary Classifiers are used in Beyer *et al.* (2020) to classify Stack Overflow posts into seven intention categories. In their work, a set of features extracted from the posts serves as input to the machine-learning-based classifiers. The feature combinations mainly include *N-gram* of the text or the part-of-speech tags (POS) of the text, word count, code snippets, and some other textual features (e.g., readability, sentiment). The authors conducted experiments on all feature combinations with a set of random forest binary classifiers and determined the best configurations for the task. This approach, to our knowledge, is the state-of-the-art work that proposed an automated approach for detecting QA post intentions. In our work, we follow the preprocessing and configurations from their work and train a set of random forest classifiers for our intention categories with our dataset as the baseline model. In our approach, individual random forest classifiers are dedicated to distinct intention categories, functioning as binary classifiers. These classifiers generate predictions specific to their assigned categories. To craft the multi-class multi-label classification output for each post, we merge the outputs from these classifiers. The final prediction for a post's intention categories is determined by aggregating the individual classifier's outputs.

Table 9 The performance of our approach after fine-tuning the PTMs compared with baselines. The values in parentheses indicate the absolute differences compared with results in RQ1.

Model	Micro averaging			Average AUC
	Precision	Recall	F1-score	
BERT	0.562 (0.052↑)	0.536 (0.056↑)	0.549 (0.027↑)	0.754
RoBERTa	**0.601** (0.004↑)	**0.577** (0.112↑)	**0.589** (0.067↑)	**0.787**
Baseline 1 (Random Forest)	0.597	0.462	0.521	0.745
Baseline 2 (CNN-based)	0.558	0.577	0.567	0.765

Baseline 2: A CNN-based approach is introduced by Huang *et al.* (2020) for the task of extracting intentions from issue reports on GitHub. This approach applies a CNN-based network and classifies sentences from issue reports into seven pre-defined intention categories. Batch normalization is integrated with the CNN layer to enhance training speed. In our work, we utilize the same CNN architecture while we substitute the cross entropy loss with the BCE loss to adapt to our task, which requires a multi-label output. We concatenate the title and description of posts and use the pre-trained GloVe word embeddings (Pennington *et al.*, 2014) to transform words into the corresponding vector representations as the input to the CNN model.

Results and Analysis To better evaluate the performance of our proposed approach, we performed 10-fold cross-validation and calculated the metrics over all the posts in our dataset. Table 9 shows the performances of the baseline models and two best-performing variants from RQ1 after fine-tuning. From the tables, we observe an overall improvement in performance: compared with the models without fine-tuning PTMs, the Micro F1-scores increase by 5.2%, and 12.8%, respectively. From Table 10, we can observe the variations of the Precision@1, recall@1, and F1-score@1, which follow the same trend as the previous metrics. For these two variants after fine-tuning, the Top 1-3 accuracy ranges from 58.7% to 84.8% and 62.6% to 87.8%, respectively.

Since the pooler layer in PTMs is often used by downstream tasks in the pre-training stage, its parameters could be highly pertinent to the particular tasks, which undermines the quality of the output embeddings. This may explain the improvement in performance observed in our experiments. As by fine-tuning this layer with our task, the quality of embedding may be improved for our downstream task. The performance improvement is reflected in the average AUC, with an increment of 3.0% and 6.8% for the two variants.

Table 10 The performance after fine-tuning.

Variant	@1		
	Precision	Recall	F1-score
BERT	0.587 (0.012↑)	0.587 (0.012↑)	0.587 (0.012↑)
RoBERTa	0.626 (0.038↑)	0.626 (0.038↑)	0.626 (0.038↑)

Answer to RQ2

By fine-tuning the pooler layer of PTMs with our annotated dataset, the performance of our intention detection framework is further improved, achieving a Micro F1-score of 0.589, Top 1-3 accuracy of 62.6% to 87.8%, and an average AUC of 0.787. Our proposed approach outperforms the baselines.

RQ3: Can the content category of code blocks really help the detection of post intentions?

Motivation From our qualitative study, we found that the content of code block has a close correlation with the intention of posts. Thus, we implement a code block content classifier and employ the predicted probabilities of content categories as a feature for intention detection. In this research question, we examine the effectiveness of this feature and further validate the findings from our qualitative study from an experimental point of view.

Approach We conduct an ablation study to investigate the importance of employing code block content as a feature. To answer this research question, we remove the code block classifier and modify our proposed framework to fit the dimensions of the input feature. We train and evaluate the ablated framework with 5-fold cross-validation to get an unbiased performance estimation.

Results and Analysis Table 11 shows the results of our ablation study. When compared with the framework with the code block content classifier, we observed a minor performance loss of the ablated ones. However, the loss is not significant: The F1-scores and average AUC for BERT decrease by 1.1% and 0.5%. The F1-scores and average AUC for RoBERTa decrease by 2.3% and 1.8%. The results confirm our assumption that the occurrence information of code block content can help the intention detection. However, the benefits of using this as a feature for intention detection may be undermined by the strong representation ability of efficient PTMs.

Answer to RQ3

The category of code block content can serve as a feature to help the detection of post intentions in our framework. However, the effectiveness is limited.

Table 11 The results of ablation study.

Ablated Model	Micro averaging			Average
	Precision	Recall	F1-score	AUC
BERT	0.554 (0.008↓)	0.533 (0.003↓)	0.543 (0.006↓)	0.753 (0.001↓)
RoBERTa	0.590 (0.011↓)	0.560 (0.017↓)	0.575 (0.014↓)	0.773 (0.014↓)

5 Lessons Learned

5.1 Insights from Collaborative Industry Endeavors

Our research has been driven and conducted in close collaboration with an industry partner that specializes in gathering, analyzing, and recommending information from online technical communities. The industry partner's use cases, feedback, and involvement in the co-construction approach have provided valuable insights and contributions for building the taxonomy, designing and improving the intention detection tool, and adopting the outcomes in the industry environment.

Driven by the industrial use cases Our collaboration with our industrial partner enables us to have access to certain industrial use cases that motivate our research. Primarily, our partner is keen on enhancing content recommendations for their platform's end users. This endeavour involves correlating potential posts with user profiles crafted from users' info and their activity histories, which may reflect their varying levels of expertise. However, the existing post tags available focusing on technical topics fail to capture the intentions of posts that may be related to the level of expertise of users. For example, novice programmers are more interested in finding learning resources—a prevalent focus commonly often found in posts categorized under the *Learning* intention. Conversely, this intention can serve as a proactive strategy to minimize the delivery of undesired content to specific user groups. Experienced users often perceive *Learning* posts as repetitive or less engaging due to their advanced knowledge. Leveraging our intention detection approach, we aim to refine the recommendation system to address the distinct needs of both novices seeking learning materials and experienced users with more advanced requirements. This strategy enables a more engaging and enriching user experience, ensuring that content recommendations cater to diverse proficiency levels across the platform.

Improving the generalizability The partnership has been helpful in identifying certain limitations in the existing intention taxonomies (e.g., Allamanis and Sutton (2013); Beyer and Pinzger (2014); Beyer *et al.* (2020); Treude *et al.* (2011)). Feedback from our partner indicates that certain categories may hold little significance due to a small number of related posts, as well as a lack of use cases and low generalizability. For instance, the category *API-related*, commonly found in previous works on specific domains such as Android (Beyer and Pinzger, 2014), may not be applicable to general domains that do not always involve API usage. Its relevance can also overlap with almost all other intentions, making it less generalizable. Therefore, it has been combined with a more general *How-to* intention.

Enhancing the practicability In order to better serve the needs of our industrial partner, we have designed our taxonomy to improve the usability in an industry environment. For example, our partner has observed that beginner programmers often post elementary programming questions, which experienced users may find repetitive. To address this, we have included a *Learning* category in our intention taxonomy to categorize these types of posts, despite the fact that such posts are under-represented. Furthermore, by combining intention categories with technical topics, we can direct questions to the appropriate domain experts more efficiently. For instance, identifying *How-to* posts and pairing them with technical tags can help the recommendation system suggest related questions to domain experts who are willing to answer questions. While our intention taxonomy covers many use cases, there may be posts that don't fit into any of our categories. These posts are classified under the *Other* category as our partner has not found any practical use or benefit for them in the recommendation system.

Improving the implementation As mentioned previously, employees from our industrial partner have been involved in our intention annotation process. The involvement of these domain experts and developers makes our annotation results to be more accurate, trustworthy, and applicable. Furthermore, their input is also incorporated into the design, construction, and evaluation of our intention classification models. By assimilating their suggestions and opinions, we ensure that our models better align with the practical requirements of the industry, making them more relevant and applicable.

Continuous improvement with industrial adoption The performance of our prototype model may be limited due to the shortage of well-annotated data, which requires significant manpower. However, our industry partner is integrating our taxonomy and classification approach into their platform to help their clients find relevant technical forum posts. We plan to enhance our model's performance by gathering more annotated data through end-user feedback. By doing so, we stand a good chance of improving the model's accuracy and effectiveness in a continuous and iterative manner.

5.2 Using pre-trained language models on forum post data

Pre-trained language models (PTMs) are effective in representing technical forum data. In our study, we explored the efficacy of PTMs in representing technical forum data, specifically targeting the task of intention classification for technical posts. Our analysis revealed that PTMs, leveraging their advanced language understanding capabilities, excel at effectively capturing and representing the nuanced information within technical forum discussions. By utilizing PTMs, our proposed model demonstrated competitive performance in detecting post intentions, even without fine-tuning, when compared to baseline methods employing traditional feature extraction or basic word embedding. This underscores the utility and proficiency of PTMs in handling textual data related to software. Therefore, we suggest that researchers and practitioners explore the utilization of PTMs across a spectrum of challenges within the realm of software engineering (e.g., uncover known issues in users' feedback for software maintenance).

Fine-tuning PTMs can be expensive. Our experimental results highlight the importance of fine-tuning the PTMs to achieve superior results for target downstream tasks. However, acquiring well-annotated data for fine-tuning PTMs for the targeted downstream task may be a resource-intensive endeavour. In our experiment, we had only 784 annotated posts available for fine-tuning and model evaluation. Consequently, to adapt the model to our tasks, we chose to exclusively update the pooler layer within the PTMs. When fine-tuning the PTMs containing an extensive number of parameters, data deficiency can potentially lead to overfitting during the fine-tuning process. In this situation, practitioners and researchers may consider selectively updating specific layers of the PTMs. This approach allows the model to update only a subset of the parameters to adapt to the downstream tasks, without jeopardizing the generalizability of the generated embeddings.

Domain-specific PTMs may not perform better. In our experiment, we compare variants of our proposed intention detection approach with both domain-specific and general-purpose PTMs. Contrary to the intuition that domain-specific PTMs, trained with software engineering (SE)-related data, would outperform their general-purpose counterparts in our task, our experimental results present a contrary outcome. Across various evaluation metrics, the general-purpose PTMs generally demonstrated superior performance over their domain-specific counterparts. This unexpected result prompts a reconsideration of the intuition that domain-specific PTMs inherently lead to better outcomes for tasks within a particular domain. The performance of PTMs may be jointly influenced by multiple factors, such as model complexity, pretraining corpus volume, etc. We encourage practitioners and researchers to evaluate both types of PTMs for their specific downstream tasks to attain optimal results.

5.3 Recommendations for technical forum developers and contributors

Using intention as a separate dimension to identify forum posts. In addition to employing technical tags that categorize posts based on subject matter or technical topics, exploring the intentions behind forum posts presents an exciting chance to improve user experience and content relevance in technical online communities. Incorporating an intention dimension in the content organization of technical forums offers an added context, revealing the motivations behind users' inquiries. Consequently, our recommendation is for forum developers to integrate a dedicated tagging or categorization system focused on discerning the intentions behind individual posts. This integration would empower forum users to contribute to and discover posts that align with particular intentions, cultivating a more purposeful interaction within these online communities.

Making good use of the code blocks. In our manual study, we observed numerous instances of code snippet misplacements and misuse of code blocks within technical forum posts (e.g., inline codes are often mixed with other descriptions). The mixture of code snippets and pieces of natural language can pose significant challenges for existing recommendation systems or technical forum data analysis methods (e.g., tag recommendation, intention mining), which are mainly based on extracting features from the texts, to generate accurate results. Hence, we encourage technical forum contributors to adhere to posting conventions, placing code snippets in code blocks, and marking inline code appropriately. This adherence will enable a more accurate presentation of their posts.

Setting clear objectives before posting. The intention taxonomy can serve as a thinking aid for individuals formulating their questions in technical forums. By leveraging this taxonomy, questioners can better structure their questions, leading to a clearer and more precise expression of their objectives. By remaining mindful of the question's intended purpose, questioners improve their ability to articulate issues effectively, benefiting both repliers and readers. A clearer delineation of objectives aids not only those providing answers but also the broader audience in comprehensively understanding the issue, enabling targeted and more helpful assistance. Therefore, we suggest that contributors to technical forums consider adopting this approach, as it has the potential to improve the efficiency and productivity of information exchange within the community.

6 Threats to Validity

Internal Validity. Manual study and annotation may be subject to the subjectivity and even bias of the authors. To reduce this bias, the two authors examine the data independently. In most cases, the agreement can be made. In case of disagreement, two authors discuss, and one other author is involved in helping reach a consensus. The involvement of other experts other than authors can further mitigate the threat.

External Validity. The datasets used (i.e., post intention and code block dataset) are restricted to limited numbers of technical forums. There are many forums or communities in the domain of software and hardware systems. However, the datasets were extracted from a working system from our industrial partner. The data has good coverage of mainstream technical developer communities. Future works can validate the generalizability of our findings and our approach with new forum posts from different sources.

Construct Validity. We used random sampling to split our dataset into folds for cross-validation, potentially causing overrepresentation or underrepresentation of certain data sources in the folds used for testing. These biased representations might influence the accuracy of our evaluation. However, we utilize all available data in our evaluation process, ensuring its robustness.

Regarding the utilization of PTMs, we employed the output from the first $<CLS>$ token to represent the post data. Various pooling strategies exist for PTMs, and choosing among them can significantly affect the performance of downstream tasks, thus influencing our assessment of PTMs. Nevertheless, the $<CLS>$ token encapsulates contextual information learned across the input sequence and commonly acts as an initial reference for downstream tasks. Our fine-tuning process, based on this token, further reduced this influence. Concerning the fine-tuning of PTMs, our initial attempt involved updating the parameters of the entire pre-trained models. However, due to the limited size of the dataset, we encountered challenges in achieving favorable training outcomes. Our subsequent approach focused solely on updating the pooler layer of the PTMs, which might not be the most optimal solution. Future works may explore additional fine-tuning strategies to enhance overall performance.

In our approach to handling code blocks within post data, we developed a classifier specifically tailored for predicting content categories. The efficacy of this classifier may influence the accuracy of intention detection, potentially impacting the construct validity of our approach. Moreover, relying solely on content categories as a feature while disregarding the actual content might lead to the loss of information, considering that the text within code blocks may hold essential insights into the post's intention. Employing advanced feature extraction and representation techniques, such as leveraging text embedding techniques to process textual data and generate representations for code block contents, holds promise for achieving more precise intention detection results. However, the adopted classifier has lower computational costs than many embedding techniques, resulting in fewer resources for practical implementation in production environments, ensuring the scalability and feasibility of our approach. Therefore, the feature of code block content is incorporated into the framework intended to be implemented on the industry partner platform.

Additionally, we implemented the baseline approaches and assessed their performance using our dataset. Given that our tasks and taxonomy differ from those in the original studies, we adapted the approaches accordingly. However, this re-implementation introduces the potential for errors and bias in our research. To mitigate this, we referenced the source codes of the original implementations during the baseline approach implementation, striving to maintain consistency with the original work. This effort aims to enhance the construct validity of our study.

Conclusion Validity. We labeled a limited number of posts, which may limit the reliability of our conclusion (the taxonomy of post intentions). To mitigate this, we sampled a statistically representative sample of 384 posts from our data dump to conduct our manual study. We further increased the number of posts for intention annotation to 784 posts, which is still limited. Using the small number of posts to train and evaluate our intention classification approach may threaten our conclusion about the performance of the model. To mitigate this issue, we use a 5-fold cross-validation for the model and calculate its performance by aggregating the results across the five test folds. Future work could involve expanding the annotated dataset and employing a larger testing set to evaluate the model.

Moreover, the limited numbers of samples in certain intention categories (i.e., *Review*, *Learning*) impedes our ability to accurately evaluate and compare the performance of our models and baseline approaches in classifying these intentions, potentially affecting the robustness and reliability of our study's findings and conclusions regarding these specific intention categories, which presents a threat to the conclusion validity of our study. Initially, our dataset contained 384 samples. In an effort to expand its size to 784, our focus primarily centered on annotating more data while maintaining the original distribution of intention categories. We did not acquire additional annotated data to train our approach due to the costs of an extensive manual annotation process. To enhance our models' performance and evaluation, future strategies might involve leveraging our approach to identify posts within these categories. Subsequently, a human verification process could be employed to augment the training set selectively, aiming to uphold the original data distribution while refining model accuracy over these less frequent intention categories.

7 Related Work

7.1 Mining Intentions from Developers' Discussions

The rapid growth of programming-related online communities has highlighted the need to better understand the characteristics and nature of online community posts. Therefore, more and more researchers have been focusing on mining and analyzing the content produced by software practitioners. Besides the technical aspects, intention can serve as an important factor in classifying and arranging technical posts in online communities. Many researchers have proposed different taxonomies for the post by manual analysis. Although some of the works did not explicitly mention the word *intention*, we can tell that some of their categories are a description of the posting purposes. Treude *et al.* (2011) were the first ones to manually classify Stack Overflow posts into ten categories from an intention aspect. Allamanis and Sutton (2013) used topic modeling to analyze questions from Stack Overflow and found the correlation between question types and programming concepts and identifiers. Instead of studying all question types, Beyer and Pinzger (2014) focused only on the android development questions and summarized 8 question types. They further employed a k-NN classifier to classify questions. Similarly, Rosen and Shihab (2016) conducted a study on mobile application development posts and classified them into *How, What, Why.* In Beyer *et al.* (2020), researchers proposed a taxonomy based on previous taxonomies and tried to construct classifiers to automatically classify Android-related QA posts from Stack Overflow. Besides the studies focusing on question-answering post data, researchers have conducted analyses to extract intentions from other software-related sources. For instance, Huang *et al.* (2020) introduced a taxonomy of intentions specific to issue reports in GitHub projects. They developed a Convolutional Neural Network (CNN)-based approach to automatically categorize sentences into predefined intention categories. Lu *et al.* (2022) focused on app reviews and proposed a deep-learning-based framework to classify them into four intention categories. **In this work, we studied the characteristics of community posts (including a classification of post intentions) that cover multiple developer communities and consider inputs from the industry. In addition, we proposed an automated intention detection framework that outperforms the state-of-the-art baseline.**

7.2 Tag recommendation for developer community posts

Researchers have developed various approaches to fulfill the task of tag recommendation in the software engineering domain. These approaches can automatically propose tags (mostly in technical aspects) for software artifacts, software objects, etc. (Al-Kofahi *et al.*, 2010; Wang *et al.*, 2018, 2015). Here, we only briefly introduce recent works on the tag recommendation for developer community posts or objects in software information sites. Hong *et al.* (2017) propose a tag recommendation method based on topic modeling. This method computes tag scores according to the document similarities and historical tag occurrence. Liu *et al.* (2018) proposed FastTagRec, which is a neural network-based method that can infer tags for new postings accurately and fast. Zhou *et al.* (2019) proposed four tag recommendation methods based on four contemporary deep learning approaches, among which TagCNN and TagR-CNN work better than traditional approaches. TagDC (Li *et al.*, 2020) further improved the performance by leveraging deep learning techniques and collaborative filtering techniques. **Our proposed intention detection framework can complement these tag recommendation approaches by providing a different perspective for locating relevant posts.**

8 Conclusions

The ever-growing online developer communities demand more efficient and rational ways of organizing content and making recommendations for users. Our work is just under this background. In this work, we first conducted a qualitative study on a sampled dataset from an industrial source to understand the common posting practices in technical communities. We proposed an intention taxonomy of technical posts by seeking feedback from our industrial partner and referring to previous studies. With this taxonomy, we manually annotated posts and analyzed the correlation between post intention type and post content. Based on the findings from the qualitative study, we proposed an intention detection framework that utilizes transformer-based pre-trained language models. We further examine the characteristics of the framework with three research questions, from which we validated the effectiveness of our approach compared with a baseline model and confirmed the relevance of code block content and post intention can be utilized and thus boost the intention detection task. Our taxonomy of post intentions and automated detection framework may be leveraged by technical forum maintainers or third-party tool developers to improve the organization and search of relevant posts on technical forums. To expand on our findings, future research could involve creating a more extensive post-intention dataset, or assessing the impact of utilizing post intents for enhancing post searches or recommendations.

Acknowledgements

We would like to gratefully acknowledge the Mitacs-Accelerate program and the Natural Sciences and Engineering Research Council of Canada (NSERC) for funding this project.

Conflicts of Interests

The authors have no competing interests to declare relevant to this article's content.

Data Availability Statements (DAS)

We have released our annotated dataset and code in the supplementary material package hosted on a GitHub repository[6].

References

Al-Kofahi, J. M., Tamrawi, A., Nguyen, T. T., Nguyen, H. A., and Nguyen, T. N. (2010). Fuzzy set approach for automatic tagging in evolving software. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10. IEEE.

Allamanis, M. and Sutton, C. (2013). Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *2013 10th Working conference on mining software repositories (MSR)*, pages 53–56. IEEE.

Barua, A., Thomas, S. W., and Hassan, A. E. (2014). What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, **19**(3), 619–654.

[6] Supplementary material package:
https://github.com/mooselab/suppmaterial-TechnicalPostIntention

Beyer, S. and Pinzger, M. (2014). A manual categorization of android app development issues on stack overflow. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 531–535. IEEE.

Beyer, S., Macho, C., Di Penta, M., and Pinzger, M. (2017). Analyzing the relationships between android api classes and their references on stack overflow. *Technical Report*.

Beyer, S., Macho, C., Di Penta, M., and Pinzger, M. (2020). What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories. *Empirical Software Engineering*, **25**(3), 2258–2301.

Boslaugh, S. (2012). *Statistics in a nutshell: A desktop quick reference*. " O'Reilly Media, Inc.".

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, **16**, 321–357.

Chen, H., Coogle, J., and Damevski, K. (2019). Modeling stack overflow tags and topics as a hierarchy of concepts. *Journal of Systems and Software*, **156**, 283–299.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., *et al.* (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

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*.

Greco, C., Haden, T., and Damevski, K. (2018). Stackintheflow: behavior-driven recommendation system for stack overflow posts. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 5–8.

Guo, J., Xu, S., Bao, S., and Yu, Y. (2008). Tapping on the potential of q&a community by recommending answer providers. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 921–930.

Hand, D. J. and Till, R. J. (2001). A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine learning*, **45**(2), 171–186.

He, J., Xu, B., Yang, Z., Han, D., Yang, C., and Lo, D. (2022). Ptm4tag: Sharpening tag recommendation of stack overflow posts with pre-trained models. *arXiv preprint arXiv:2203.10965*.

Hong, B., Kim, Y., and Lee, S. H. (2017). An efficient tag recommendation method using topic modeling approaches. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pages 56–61.

Huang, C., Yao, L., Wang, X., Benatallah, B., and Sheng, Q. Z. (2017). Expert as a service: Software expert recommendation via knowledge domain embeddings in stack overflow. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 317–324. IEEE.

Huang, J., Tang, D., Shou, L., Gong, M., Xu, K., Jiang, D., Zhou, M., and Duan, N. (2021). Cosqa: 20,000+ web queries for code search and question answering. *arXiv preprint arXiv:2105.13239*.

Huang, Q., Xia, X., Lo, D., and Murphy, G. C. (2020). Automating intention mining. *IEEE Transactions on Software Engineering*, **46**(10), 1098–1119.

Jin, D., Jin, Z., Zhou, J. T., and Szolovits, P. (2020). Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34-05, pages 8018–8025.

Khandkar, S. H. (2009). Open coding. *University of Calgary*, **23**(2009).

Krippendorff, K. (2011). Computing krippendorff's alpha-reliability. *Computing*, **1**, 25.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Li, C., Xu, L., Yan, M., and Lei, Y. (2020). Tagdc: A tag recommendation method for software information sites with a combination of deep learning and collaborative filtering. *Journal of Systems and Software*, **170**, 110783.

Liu, J., Zhou, P., Yang, Z., Liu, X., and Grundy, J. (2018). Fasttagrec: fast tag recommendation for software information sites. *Automated Software Engineering*, **25**(4), 675–701.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lu, J., Wu, Y., Pei, J., Qin, Z., Huang, S., and Deng, C. (2022). Miar: A context-aware approach for app review intention mining. *International Journal of Software Engineering and Knowledge Engineering*, **32**(11n12), 1689–1708.

Maity, S. K., Panigrahi, A., Ghosh, S., Banerjee, A., Goyal, P., and Mukherjee, A. (2019). Deeptagrec: A content-cum-user based tag recommendation framework for stack overflow. In *Advances in Information Retrieval: 41st European Conference on IR Research, ECIR 2019, Cologne, Germany, April 14–18, 2019, Proceedings, Part II 41*, pages 125–131. Springer.

Mashhadi, E. and Hemmati, H. (2021). Applying codebert for automated program repair of java simple bugs. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 505–509. IEEE.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Qiao, Y., Xiong, C., Liu, Z., and Liu, Z. (2019). Understanding the behaviors of bert in ranking. *arXiv preprint arXiv:1904.07531*.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Rosen, C. and Shihab, E. (2016). What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, **21**(3), 1192–1223.

Sahare, M. and Gupta, H. (2012). A review of multi-class classification for imbalanced data. *International Journal of Advanced Computer Research*, **2**(3), 160.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

StackOverflow (2022). Best practices for tag lifecycle management: Applying tags.

Stol, K.-J. and Fitzgerald, B. (2018). The abc of software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **27**(3), 1–51.

Tabassum, J., Maddela, M., Xu, W., and Ritter, A. (2020). Code and named entity recognition in stackoverflow. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Treude, C., Barzilay, O., and Storey, M.-A. (2011). How do programmers ask and answer questions on the web?(nier track). In *Proceedings of the 33rd international conference on software engineering*, pages 804–807.

Von der Mosel, J., Trautsch, A., and Herbold, S. (2022). On the validity of pre-trained transformers for natural language processing in the software engineering domain. *IEEE Transactions on Software Engineering*.

Wang, S., Lo, D., Vasilescu, B., and Serebrenik, A. (2018). Entagrec++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering*, **23**, 800–832.

Wang, X.-Y., Xia, X., and Lo, D. (2015). Tagcombine: Recommending tags to contents in software information sites. *Journal of Computer Science and Technology*, **30**(5), 1017–1035.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., *et al.* (2019). Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Yang, C., Xu, B., Khan, J. Y., Uddin, G., Han, D., Yang, Z., and Lo, D. (2022). Aspect-based api review classification: How far can pre-trained transformer model go. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE Computer Society*.

Yazdaninia, M., Lo, D., and Sami, A. (2021). Characterization and prediction of questions without accepted answers on stack overflow. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 59–70. IEEE.

Zhou, P., Liu, J., Yang, Z., and Zhou, G. (2017). Scalable tag recommendation for software information sites. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 272–282. IEEE.

Zhou, P., Liu, J., Liu, X., Yang, Z., and Grundy, J. (2019). Is deep learning better than traditional approaches in tag recommendation for software information sites? *Information and software technology*, **109**, 1–13.