Towards the Analysis and Completion of Syntactic Structure Ellipsis for Inline Comments

Xiaowei Zhang, Weiqin Zou, Lin Chen, Yanhui Li, *Member, IEEE*, and Yuming Zhou

Abstract—The ellipsis of the syntactic structure is a common phenomenon in ordinary textual documents. Existing studies have found that despite syntactic ellipsis could help avoid repetition of normative documents, it could also, for example, lead to ambiguity and hamper the understandability of document contents. As a fundamental component of software, code comments are generally written by developers in a non-structured way just like normative documents. This naturally inspires us to explore whether syntactic ellipsis is also a common phenomenon in code comments and what potential negative effects would such ellipsis have on software tasks such as code/comments comprehension activities. Such explorations, in our opinion, are expected to facilitate the research on code comments and comments-related software tasks. To this end, we conduct the first large-scale study to explore the syntactic structure ellipsis problem of code comments, with a focus on Java inline comments. Specifically, we construct a data set of 1,000 Java projects with 1,307,457 inline comments and associated codes. Based on this data set, we first study the prevalence of syntactic structure ellipsis in inline comments. We find that syntactic structure ellipsis is quite common in inline comments where 83.6% comments have structure ellipsis (such as subject/predicate omissions). Then, we investigate the effects of syntactic structure ellipsis on code/comment understanding activities. As a result, we find that there indeed exists a negative relationship between them, with a medium effect size. Based on these findings, we further propose neural network based approaches to complete the ellipsis parts for the inline comments. With our approach, we could achieve: 1) a medium improvement in assisting code/comment understanding activities, and 2) a substantial improvement of 11.3% in comment-assisted code abbreviation extension task.

Index Terms—Inline comments, syntactic structure, ellipsis analysis, ellipsis completion

1 Introduction

The ellipsis of syntactic structure (such as noun/verb phase ellipsis) is a common phenomenon in linguistics and natural language processing textual documents [1]. Although such ellipsis could help avoid repetition and highlight important information, it could also lead to textual ambiguity and greatly hamper the understandability of the document contents [2], [3], [4], [5], [6], [7]. In the natural language processing area, the documents (or some parts of them) are written in a more colloquial, unstructured, and casual way, hence with more syntactic structure ellipsis being observed. A large amount of syntactic structure ellipsis not only brings great challenges to syntactic analysis tasks [8], [9], [10], but also brings potential threats to the tasks that rely on text comprehension (such as information retrieval tasks and translation tasks) [9]. To alleviate the potential problems brought by syntactic structure ellipsis, researchers have done much work around ellipsis analysis (mainly on noun/verb phrase ellipsis) and recovery [8], [9], [11], [12], [13], [14], [15].

As a fundamental component of software documentation [16], code comments are always essential for software comprehension and maintenance [17], [18], [19]. Unfortunately, comments are generally written by developers in a non-structured way just like the ordinary documents in nature language processing area. This indicates it is very likely that code comments may also have similar syntactic structure ellipsis problems as that of ordinal textual documents. However, despite previous studies have performed some analyses on syntax of comments, almost none of them, precisely targets the syntactic structure ellipsis problem and its potential effects on code/comment related software tasks. This naturally inspires us to explore: 1) whether syntactic structure ellipsis is also a common phenomenon in code comments and what types of syntactic structure ellipsis might be found in code comments; 2) what potential impacts would this ellipsis has on code/comment-related tasks such as code/comments comprehension activities; and (3) how to avoid relevant potential negative impacts. Such explorations, in our opinion, are expected to facilitate the research on code/comments related software tasks such as code understanding/maintenance [17], [20], [21], [22], [23], [24], [25], code abbreviation extension [26], [27], [28], [29], [30], bug detection [31], [32], [33], and comments transfer [34].

To this end, in this paper we take the first step to conduct a large-scale study to explore the syntactic structure ellipsis problems of Java inline comments. Unlike Javadoc comments

- *Xiaowei Zhang, Lin Chen, Yanhui Li, and Yuming Zhou are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: xiaoweizhang@smail.nju.edu.cn, {lchen, yanhuili, zhouyuming}@nju.edu.cn.*
- *Weiqin Zou is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China. E-mail: weiqin@nuaa.edu.cn.*

Manuscript received 4 December 2021; revised 30 September 2022; accepted 17 October 2022. Date of publication 21 October 2022; date of current version 18 April 2023.
This work was supported in part by the National Natural Science Foundation of China under Grants 61872177, 62172202, 62172205, and 62002161, and in part by the Cooperation Fund of Huawei-Nanjing University Next Generation Programming Innovation Lab under Grant YBN2019105178SW35.
(Corresponding authors: Weiqin Zou and Lin Chen.)
Recommended for acceptance by Z. Jin.
This article has supplementary downloadable material available at https://doi.org/10.1109/TSE.2022.3216279, provided by the authors.
Digital Object Identifier no. 10.1109/TSE.2022.3216279

particularly written to describe the method on the whole and generally follow a standard writing format, inline comments usually focus on describing certain statements within a method (such as describe variables, explain statements, or debug) and are written in a more casual way. In other words, developers could write inline comments in any free style as they like [35]. Some may even not or just simply write very short or fragile comments due to release pressure [36]. This would generate many inline comments with syntactic structure ellipsis. How would such ellipsis affect code/comment comprehension tasks directly inspired us to do our investigations in our study.

Our mainly work is as follows.

First, we collect the 1,000 most popular Java projects on GitHub. Then, we link inline comments with their associated code, so that it is possible to analyze how the syntactic structure ellipsis of inline comments would affect the understandability of inline comments and the corresponding code. To this end, we propose heuristic rules to obtain such a data set of 1,307,457 pairs of <Inline Comment, Code>, together with manual check for guaranteeing the linking quality. This is the first large-scale data set of inline comments and their corresponding code. Based on this data set, we analyze the syntactic ellipsis in inline comments and mine the types of syntactic structures presented in inline comments, which mainly include subject structure, predicate structure, and object structure ellipsis in our classification. Our statistical results show that syntactic structure ellipsis is quite common in inline comments, and 83.6% of inline comments do have syntactic structure ellipsis. Then, we analyze the impact of syntactic ellipsis on comment-assist code understanding activities. Our results show that for comment-assist code understanding activities, inline comments without syntactic ellipsis are superior (with a medium effect size) to inline comments with syntactic ellipsis from the perspective of developers.

Based on the above-mentioned findings, we propose to automatically complete the ellipsis of syntactic structure of inline comments based on neural networks. For the completion of syntactic structure ellipsis task, our method could achieve a precision of 65.3%. To better understand the potential of our approach, we further conduct two kinds of evaluations: how helpful such completions are in facilitating code/comment understanding task, and whether the completion of structure ellipsis will improve the code abbreviation extension task. More specifically, for code/comment understanding task, we conduct a user questionnaire experiment by randomly sampling code-comment instances and ask participants to rate which code-comment (with or without syntactic ellipsis completion) is more understandable. For code abbreviation expansion task, we investigate whether our approach would enhance the performance of the common technique in expanding code abbreviations. Our experimental results show that after completion: 1) there is a medium improvement for the work of assisting code/comment understanding activities; and 2) there is a 11.3% improvement for the work of comment-assisted code abbreviation extension.

Our major contributions are as follows:

1) We are the first to conduct a large-scale empirical study on the syntactic structure ellipsis problems of inline comments.
2) We analyze the prevalence, types, and potential impacts on code-comment related software tasks of syntactic structure ellipsis of more than one million (1,307,457) inline comments.
3) We examine to what extent we can complete the syntactic structure ellipsis of inline comments, and evaluate the applicability of our completion from the understandability improvement of inline comments and codes, and the usefulness in comment-based code abbreviation expansion task.
4) We build and public the first large data set of inline comments and their corresponding code. The data set[1] can be used for replication and further research.

The rest of this paper is organized as follows. Section 2 introduces concept of syntactic structure ellipsis and gives several examples. Section 3 describes the experiment setup of our work. Section 4 presents the details of the empirical analysis results. Section 5 shows the results of the completion model and the corresponding evaluation work. Section 6 discusses the implications and possible threats of this work. Section 7 illustrates the related work on code comments. Finally, Section 8 provides conclusions.

2 Background

In this section, we introduce the concept of syntactic structure ellipsis and give several examples that illustrate the absence or not of such ellipsis.

Ellipsis Definition. A complete English sentence generally has a certain structure as shown in the following expressions (1) and (2); and we call the absence of any part of the syntactic structure Ellipsis. Specifically, for a given English sentence *St*, the syntactic structure of *St* may consist of subject (*Sub*), predicate (*Pre*), object (*Obj*), and sentential complements (*Cl*), with Obj being divided into direct object (*D-Obj*) and indirect object (*I-Obj*). *Sub* is the person, place, or thing that is performing the action. *Pre* expresses action within the sentence. *D-Obj* receives the action, while *I-Obj* indicates to whom or for whom the action is being done. *Cl* either renames or describes the subject or object.

$$St \rightarrow \begin{cases} Sub\ \&\ Pre\ \&\ Obj \\ Sub\ \&\ Pre\ \&\ Obj\ \&\ Cl \end{cases} \tag{1}$$

$$Obj \rightarrow \begin{cases} D\text{-}Obj \\ I\text{-}Obj. \end{cases} \tag{2}$$

In this paper, we mainly consider three types of ellipsis that may appear in code comments, i.e., the ellipsis of subject, predicate, and object structure. The reason for choosing these three types of ellipsis is that subject, predicate, and object structure are the basic and core components of a sentence [1], [37]. Their ellipsis may directly affect the understandability of sentence and pose potential threats to text-related tasks [8], [10], [12], [14], [38]. Through our observation, we find that syntactic structure ellipsis is not rare in code comments, and such ellipsis to a certain extent hampers the readability and understandability of those comments as well as related code. The following examples illustrate the possible ellipsis of *Sub*, *Pre*, and *Obj* within a comment. These examples are all from

1 https://github.com/Sherww/CC-SSE

```
// must have a payload
if (authHeaderBase64String.isEmpty()) {
  throw new HttpAuthenticationException("Authorization
  header received " + "from the client does not contain
  any data.");
}
return authHeaderBase64String;
```

(a) With subject structure ellipsis

```
// Authorization header must have a payload
if (authHeaderBase64String.isEmpty()) {
  throw new HttpAuthenticationException("Authorization
  header received " + "from the client does not contain
  any data.");
}
return authHeaderBase64String;
```

(b) Without subject structure ellipsis

Fig. 1. A piece of code and its inline comments from hbase[2] project.

```
// Everyone reservations on QueueC
verifySubmitReservationSuccess(QUEUE_B_USER,queueC);
verifySubmitReservationSuccess(QUEUE_B_ADMIN,queueC);
verifySubmitReservationSuccess(QUEUE_A_USER,queueC);
verifySubmitReservationSuccess(QUEUE_A_ADMIN,queueC);
verifySubmitReservationSuccess(COMMON_USER,queueC);
```

(a) With predicate structure ellipsis

```
// Everyone can submit reservations on QueueC
verifySubmitReservationSuccess(QUEUE_B_USER,queueC);
verifySubmitReservationSuccess(QUEUE_B_ADMIN,queueC);
verifySubmitReservationSuccess(QUEUE_A_USER,queueC);
verifySubmitReservationSuccess(QUEUE_A_ADMIN,queueC);
verifySubmitReservationSuccess(COMMON_USER,queueC);
```

(b) Without predicate structure ellipsis

Fig. 2. A piece of code and its inline comments from hibernate-orm[3] project.

real-world software projects, which also help us understand the possible negative effects of different ellipsis in software development practice.

Fig. 1 is an example of subject ellipsis from the hbase project. The inline comment in Fig. 1a on the first line lacks the subject. By only reading the comment, developers may encounter problems in finding the main object of the code. As a result, they may have to read the code itself to find the answer. If we can provide the comment (shown in Fig. 1b) which contains the corresponding subject structure (i.e., "Authorization header"), developers may quickly grasp the intent of the code.

Fig. 2 is an example of predicate ellipsis from the hibernate-orm project. The inline comment in Fig. 2a lacks the predicate, i.e., the action words (usually composed of verbs). Since a predicate is a conjunction of subject and object, predicate ellipsis would make it more difficult to understand the comment and hence more time is required to understand the code. If we can complete the predicate part (i.e., "can submit") as shown in Fig. 2b, developers may easily understand the code without even reading it.

```
// the log level would actually allow to be logged
if (logWarnings) {
  try {
    walkWarnings(statement.getWarnings(),handler);}
  catch (  SQLException sqlException) {
    LOG.debug("could not log warnings",sqlException);}
}
```

(a) With object structure ellipsis

```
// the log level would actually allow a warning to be
logged
if (logWarnings) {
  try {
    walkWarnings(statement.getWarnings(),handler);}
  catch (  SQLException sqlException) {
    LOG.debug("could not log warnings",sqlException);}
}
```

(b) Without object structure ellipsis

Fig. 3. A piece of code and its inline comments from hadoop[4] project.

Fig. 3 is an example of object ellipsis from the Hadoop project. The inline comment in Fig. 3a lacks the object structure while the one in Fig. 3b completes the object ("a warning"). It is not hard to figure out that the completed comment in Fig. 3b has a better readability and better describes the associated code.

We conclude from the examples that inline comments with syntactic structure ellipsis may have potential negative effect on code/comment-related tasks such as code/comments comprehension activities. Consequently, it is important to investigate whether syntactic structure ellipsis is a common phenomenon in code comments, what potential impacts would this ellipsis have, and how to avoid it.

3 Experiment Setup

The goal of our study is to explore the ellipsis of syntactic structure in code comments, and try to build models to improve comments and facilitate code/comment-related tasks (such as code/comment understanding and code abbreviation extension). Thus, our study mainly includes two parts, the empirical analysis of syntactic structure ellipsis and the automated completion of such ellipsis. Fig. 4 shows the framework of our work.

In the empirical analysis part, we mainly attempt to explore the following questions:

- *RQ1:* What is the prevalence of syntactic structure ellipsis in inline comments?
- *RQ2:* What potential impacts do syntactic structure ellipsis have on code/comment-related software tasks?

RQ1 investigates the prevalence of syntactic structure ellipsis, i.e., whether (and the frequency of) syntactic structure ellipsis exists in inline comments. RQ2 investigates the influence of syntactic structure ellipsis, i.e., may uncover the potential impacts of syntactic structure ellipsis in software tasks such as code understanding activities.

In the ellipsis completion & evaluation part, if we could conclude from RQ1 that ellipsis is common in inline

2 https://github.com/apache/hbase
3 https://github.com/hibernate/hibernate-orm
4 https://github.com/apache/hadoop

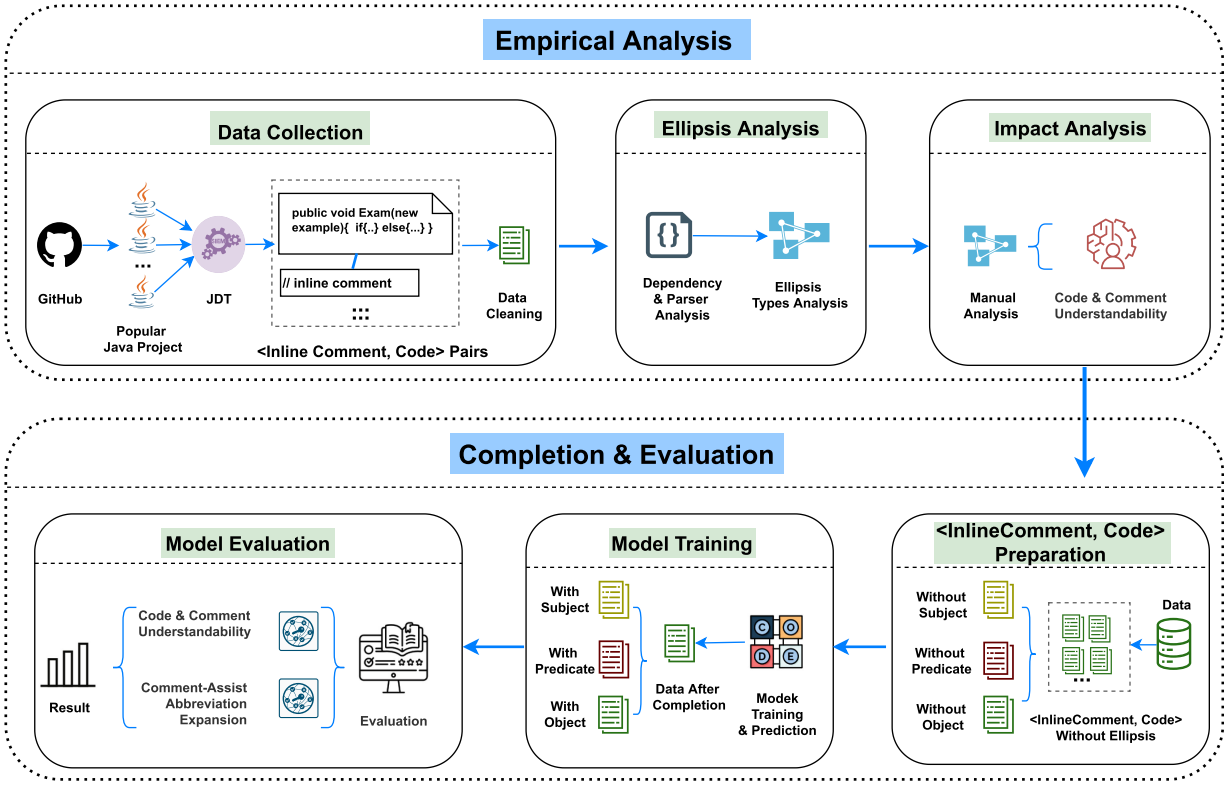

Fig. 4. The framework of syntactic structure ellipsis analysis and completion on inline comments.

comments, and RQ2 that the ellipsis of inline comments could cause negative impacts on code/comment-related software tasks, we further attempt to explore the following questions:

- *RQ3:* To what extent can we automatically complete the ellipsis of syntactic structure?
- *RQ4:* How helpful such completions are in code/comment understanding activities? How helpful such completions are in the comment-assisted abbreviation extension task?

RQ3 investigates the performance (e.g., precision and recall) of the proposed model to predict the ellipsis structures. RQ4 investigates the applicability of inline comments after syntactic structure ellipsis completion.

3.1 Empirical Analysis of Syntactic Structure Ellipsis

To answer RQ1, we first construct a data set for experimental analysis, which is composed of inline comments and corresponding code. Then, with this data set, we analysis the prevalence, types, and patterns of the syntactic structure ellipsis of inline comments. To answer RQ2, we carry out a manual inspection.

3.1.1 Data Set Construction

There is currently no data set of inline comments and their corresponding code for empirical analysis of structure ellipsis yet. Hence, we construct such a data set for later analysis. We select the most popular 1,000 Java projects on GitHub as the experimental projects. For these projects, we first use the Java development tools (JDT)[5] and heuristic association rules proposed in [39] to obtain the inline comments and code pairs. Then, we apply data cleaning to these code-comment pairs to get the final data set for analysis. The cleaning includes two steps. At the first step, we adopt an existing comment classification strategy [40] to obtain comments that describe and interpret code. At the second step, we further clean data based on the length distribution of inline comments. The data set construction details are as follows.

Selection of Experimental Projects. In this study, we consider Java projects meeting the following criteria from GitHub as the potential experimental projects.

1) Projects with great popularity. Popular projects are generally actively developed and are likely to contain more inline comments. Following [19], [41], [42], we use the *Stars* number to measure the popularity of a project on GitHub. We rank Java projects based on *Star* numbers, and select the top-ranked 1,000 projects with most stars as experimental projects.
2) English commented projects. We only consider projects with comments written in English. We first check if the comment can be encoded by ASCII. Then we calculate the percentage of comments that are all ASCII encoded in a project. If the percentage exceeds 90%, it will be considered as an English-commented project. Otherwise it will be considered as a non-English project and hence is removed.
3) Non-toy typical software development projects. These projects are mainly software development projects rather than for example documentation or experimental/test projects. We take a two-step method to filter out toy projects. First, we use heuristic patterns to identify potential toy projects, by checking whether

5. https://www.eclipse.org/jdt/

TABLE 1
Basic Statistics of 1,000 Selected Java Projects

Attributes	Total Number	Mean Number
Java files	550,107	550
Methods	5,702,126	5,702
Methods (Javadoc)	1,437,221	1,437
Methods (inline)	599,233	599
Methods (both)	132,896	133
Stars	4,114,901	4,115

[1]*Methods : total methods of all projects.*
[2]*Methods (Javadoc) :methods that contain Javadoc.*
[3]*Methods (inline) : methods that contain inline comments.*
[4]*Methods (both) : methods that both contain Javadoc and inline comments.*

their readme files contain keywords such as "toy", "test", "experiment", " learn", and " exercises". Then, for each obtained project, we manually check its readme file and code base, and determine whether it is a toy project or not. The first three authors are involved in the checking process.

Among the projects meeting above the criteria, we choose 1,000 projects (i.e., with most stars) as our experimental projects. The statistics for the data set are shown in Table 1. The average star number is 4,115 and the average commits number is 3,243, which indicates that projects are being actively developed [43]. As can be seen, 10.5% $(599,233/5,702,126)$ of methods contain inline comments. In addition, there are only 2.3% $(132,896/5,702,126)$ methods that include both Javadoc and inline comments. That is, in 8.2% of the methods, developers can only use inline comments to conduct code understanding activities, thus it is meaningful to explore the syntactic structure ellipsis of inline comments in our opinion.

<InlineComment, Code> Pairs Retrieval. After obtaining experimental projects, our next step is to retrieve inline comments as well as their associated commented code, i.e., <Inline Comment, Code> pairs for analysis. Specifically, for each *". java "* file, we use the JDT tool to extract inline comments, including line comments "//..." and block comments "/ *...*/". Then, we take two steps to extract the corresponding code blocks for these inline comments.

At the first step, we merge consecutive multi-line comments into a single comment as they actually belong to the same relatively-large comment. We will take the merge actions under two situations: (1) the inline comment is composed of consecutive line comments and there is no empty line among them; (2) the inline comment is not composed of consecutive line comments, but the corresponding code consist of multiple lines of single line code. For example, in Fig. 5, lines 36-37 would be merged into one comment according to situation (1), since no empty line exists between lines 36-37; and the obtained merged comment would be further merged with lines 38-39 to form a final comment according to situation (2), since line 38 is empty and the corresponding code consists of multiple lines of single line code (40-41).

Our second step is to determine the code associated with the inline comments. Based on the work [39] that roughly identify the related code for both inline comments and JavaDoc, we propose the following heuristic rules that can help us more accurately determine the scope of code corresponding to inline comments:

```
34 public RGBLuminanceSource(int width, int height, int[]
   pixels) {
35     ...
36     // In order to measure pure decoding speed, we
   convert the entire image to a greyscale array
37     // up front, which is the same as the Y channel of
   the YUVLuminanceSource in the real app.
38     //
39     // Total number of pixels suffices, can ignore shape
40     int size = width * height;
41     luminances = new byte[size];
42     for (int offset = 0; offset < size; offset++) {
43         ...
44     }
45   }
```

Fig. 5. A piece of code and its inline comments from *RGBLuminanceSource.java* of Zxing[6] project.

1) If a comment is on the same line as the code (e.g., a simple variable declaration code statement), then the code is selected as the corresponding code for the comment.
2) If a comment is written before a block of code, usually marked with a left parenthesis (e.g., the *if {...}* statement block), then this block of code is considered as the code corresponding to the comment.
3) If comments are on different lines, then all code statements (i.e., single line statements or code blocks before reaching a blank line or a next comment) at the same level with the comments are chosen as the associated code.
4) If a comment matches none of the above three heuristic rules (e.g., a comment is written on the last line of a code block.), then this inline comment would be ignored.

By applying the heuristic rules above, we get 1,307,457 pairs of <InlineComment, Code> from 1,000 projects.

To evaluate the accuracy of our heuristic rules, we conduct a manual inspection of the data pairs of <InlineComment, Code>. Specifically, following the sampling strategy in [40], [44], [45], we first randomly select 385 methods containing inline comments (646 inline comments in total) from 599,233 methods, with a confidence level of 95% and a sampling error range of $\pm 5\%$.[7] Then, for each inline comment, one Ph.D., one Ph.D. candidate, and one graduate student (non-author programmer) manually mark the corresponding code independently. They are of 2-3-year experience in developing Java projects, and have no problem in reading English books and even communicating with English native speakers. After that, they check the results and solve inconsistency by discussing until a consensus was achieved. The manual marked <InlineComment, Code> pairs are then taken as the ground truth to check the accuracy of our proposed heuristic rules. We find that, among 646 inline comments, our rules could retrieve the associated code for 611 inline comments. Hence, the accuracy of our heuristic rules is about 95% $((611)/646)$, which to some extent verifies the effectiveness of these rules.

6 https://github.com/zxing/zxing/
7 https://www.calculator.net/

TABLE 2
Six Categories and Sixteen Subcategories of Comments Proposed in [40]

Category	Subcategory	Description
Purpose	Summary Expand Rationale	Describe the functionality of linked source code
Notice	Deprecation Usage Exception	Describe the functionalities that should be used with care
Under Development	TODO Incomplete Commented code	Describe the topics related to ongoing and future development
Discarded	Automated generated Noise	Meaning is hard to understand due to their poor content
Style & IDE	Directive Formatter	Used to logically separate the code or provide special services
Metadata	License Ownership Pointer	Define meta information about the code

<InlineComment, Code> Pairs Filtering. Since our work is on the basis of comments that describe and interpret the code, we further remove non-English inline comments and noisy inline comments such as those that describe the debugging procedures, or just express the current mood of developers. To this end, we refer to the comments classification work of Pascarella and Bacchelli [40]. As shown in Table 2, in their work, comments are categorized into six categories and sixteen subcategories.

In this study, we keep six subcategories of two categories, i.e., *Purpose* and *Notice*, which describe and interpret the code and are shown in rows 2-7 in Table 2. We remove the rest four categories (as shown in rows 8-19 in Table 2) due to the following reasons: (1) the category of *Under development* has a weak correlation with the current code, such as an empty body of a comment; (2) the category of *Discarded* is difficult to understand or meaningless, such as comments with poor content; and (3) the categories of *Style & IDE* and *Metadata* are not helpful in assisting code understanding, such as comments composed of repeated symbols. In addition, we also remove non-English comments (this step was carried out before classification in the work of Pascarella and Bacchelli [40]).

In this study, we mainly use the following heuristics to remove comments belonging to the above-mentioned categories.

- *Under Development*. (1) *Todo*. Comments that contain keywords such as *"Todo"*, *"Fixme"* are removed; (2) *Incomplete* (aka. Empty). Comments without content (e.g., // .) are removed; and (3) *Commented code*. Comments that contain source code are removed.
- *Discarded*. Comments that contain keywords such as *"Nothing"*, and *"Ignore"* are removed.
- *Style & IDE*. Comments that contain keywords such as *"NON-NLS"*, and *"ASCII"* are removed.
- *Metadata*. Comments that contain keywords such as *"https://"*, and *"http://"* are removed.
- *Non-English*. Comments that are written in non-English are removed by checking if the comment can only be encoded using ASCII characters. Comments that cannot be encoded are removed because they contain characters from other alphabet.

After filtering, the inline comments that describe and interpret the code can be obtained. We further conduct a statistical analysis of the length (i.e., number of words) distribution of those inline comments. We find that the 1st quartile, median, and 3rd quartile values of the comment lengths are 2, 7, and 10, respectively. In our study, we remove inline comments which are too short (length <= 2) for typical-inline-comment analysis; and obtain a data set which includes 722,348 <InlineComment, Code> pairs.

Table 3 presents the basic statistics over the final data set. In Table 3, the length statistics for the comments and corresponding code (including minimum, 1st quartile, median, 3rd quartile, and maximum values) are calculated. We further divide these inline comments into six types according to Java documentation,[8] namely LineComment, SameLineComment, UnionLineComment, SameUnionLineComment, BlockComment, and SameBlockComment; and calculate these length statistics for each type of inline comments as well as their corresponding code. From the table, we can find that:

- Most inline comments are relatively short. The median value of the length of comments is 7, and 75% of comments have a length $< = 12$ words.
- Developers tend write inline comments directly above the corresponding code. This is indicated by the result that *LineComment* accounts for the largest amount (55,355/722,348 = 76.6%) among six different inline comment types.

3.1.2 Ellipsis Analysis

After data collection, we first analyze the prevalence of ellipsis to motivate our research, and then explore the impacts of ellipsis on code understandability.

Prevalence Analysis of Syntactic Structure Ellipsis. To analyze the syntactic structural components of sentences in inline comments, we use the Stanford Parser tool.[9] Fig. 6 shows an example of individual components for an inline comment sentence, including noun phrase (NP), verb phrase (VP), and so on. Based on this, we analyze three types of syntactic structure ellipsis in inline comments, namely subject structure ellipsis, predicate structure ellipsis, and object structure ellipsis. Then, we explore how prevalent the three types of syntactic structure ellipsis are in different code structures.

Since comments are closely related to the code they describe or interpret, to have a better understanding of the structural ellipsis of inline comments, we also take code structures into account during type analysis to explore their differences in ellipsis. Following [34], we consider four kinds of code structures, including individual code statement and three types of code blocks (namely, If-Else Branch, For-While Loop, and Try-Throw Exception).

Impact of Syntactic Structure Ellipsis Code Understandability. After analyzing the prevalence of different types of

8. https://docs.oracle.com/javase/tutorial/java/annotations
9. https://nlp.stanford.edu/software/lex-parser.shtml

TABLE 3
The Overall Length Distribution of Different Types of Inline Comments

	Comment Types	Min	1st Quartile	Median	3rd Quartile	Max	Numbers
Length of words	**All Comments**	**3**	**5**	**7**	**12**	**402**	**722,348**
	Corresponding Code	1	18	42	91	31,721	722,348
Length of words	**LineComment**	3	5	7	11	402	**550,355**
	Corresponding Code	1	20	45	93	31,721	550,355
	SameLineComment	3	5	7	11	402	55,215
	Corresponding Code	1	18	42	90	31,721	55,215
	UnionLineComment	3	5	7	12	402	87,517
	Corresponding Code	1	18	42	91	31,721	87,517
	SameUnionLineComment	3	5	7	12	402	1,373
	Corresponding Code	1	18	42	91	31,721	1,373
	BlockComment	3	5	7	12	402	23,032
	Corresponding Code	1	18	43	91	31,721	23,032
	SameBlockComment	3	5	7	12	402	4,856
	Corresponding Code	1	18	42	91	31,721	4,856
	Total	-	-	-	-	-	722,348

[1]*Corresponding Code : code that corresponding to inline comment.*
[2]*LineComment : single line comment that is directly written above the code.*
[3]*SameLineComment : LineComment that is written with the code in same line.*
[4]*UnionLineComment : comment that consists of multiple lines of comments.*
[5]*SameUnionLineComment : UnionLineComment that is written in same line.*
[6]*BlockComment : comment that is written with the form of "/ *...*/".*
[7]*SameBlockComment : BlockComment that is written in same line.*

syntactic structure ellipsis, we explore whether such ellipsis would affect code understanding activities. We perform a manual check to evaluate the impact of syntactic ellipsis of inline comments on code understanding. As a supplement, we also explore the impact of syntactic ellipsis on comment readability based on several readability metrics[10] in the appendix part, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2022.3216279.

We perform impact analysis with the help of two statistical approaches, namely Mann-Whitney U test [46] and Cliff's delta [47]. Mann-Whitney U test [46] is used to determine whether there is a statistically significant difference. A significance threshold is set at 0.05 (p-value < 0.05 indicates a statistically significant difference). Following [44], [48], [49], we use Cliff's delta [47] to measure the effect size δ of the difference. $|\delta| < 0.147$ indicates a negligible difference, $0.147 \leq |\delta| < 0.330$ indicates a small difference, $0.330 \leq |\delta| < 0.474$ indicates a medium difference, and $0.474 \leq |\delta|$ indicates a large difference.

Impact Analysis by Manual Inspection. Given that the role of comments studied in this paper is to describe/interpret code, we want to know how syntactic structure ellipsis of inline comments affects code understanding activities. To this end, we conduct a manual inspection to explore the differences in understanding activities when inline comments are combined with code. Specifically, we manually review the sampled <InlineComment, Code> pairs of the above-mentioned code structures with/without syntactic structure ellipsis. We only sample from those pairs with code having 42-91 words (median and 3rd quartile values, respectively) and comment having 7-12 words (median and 3rd quartile values) based on our observations of comment-code data. The reasons why we adopt such a sampling strategy are as follows: (1) too short code snippets may be easy to understand even without comment, while too long code snippets may be too difficult to understand; (2) too short comments may be relatively easy to understand and may not be precise enough to describe the code, while too long comments may be more likely to have no ellipsis of syntactic structure. The number of samples for each code structure is proportional to their total numbers in all inline comments.

Followed the sampling strategy in [40], [44], [45], we randomly select 384 <InlineComment, Code> pairs with syntactic structure ellipsis and 384 pairs without syntactic structure ellipsis, with a confidence level of 95% and a sampling error range of ±5%, respectively. The 768 (384+384) pairs are randomly ordered (to avoid personal bias) for manual inspection. One Ph.D., one Ph.D. candidate, and one graduate student (non-author programmer) participate

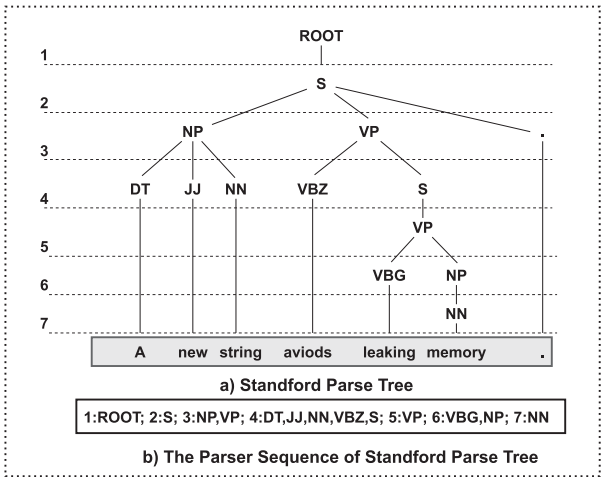

Fig. 6. The parser tree of an inline comment analyzed by using the Stanford parser.

10. https://en.wikipedia.org/wiki/Readability

in checking these pairs, with none knowing in advance whether the comment contains a syntactic structure ellipsis or not. These three participants are of 2-3-year experience in developing Java projects, and they have no problem in reading English books and even communicating with English native speakers. This largely helps guarantee the quality of our user evaluation.

We use a cross-validation method, and assign each <InlineComment, Code> pair to these three people. When divergence arises, they resolve it through open discussion. If the discussion fails to reach an agreement, the pair will be discarded, and a new pair will be re-selected to be analyzed so that the number of samples remains the same. In the review process, the five-point Likert scale [50] is used. 1, 2, 3, 4, and 5 represent the degree to which it helps understand the code, i.e., completely unable (1), basically unable (2), partially helpful (3), helpful (4), and very helpful (5).

Once 768 pairs of data have been scored. We re-separate the data into two groups with and without syntactic structure ellipsis. Then, we perform a statistical analysis to calculate the minimum, 1st quartile, median, 3rd quartile, and maximum values of scores for these two groups, respectively. After that, we use Mann-Whitney U test [46] and Cliff's delta [47] to check the difference between the two groups in terms of the above-mentioned statistical values. By analyzing the Likert options and statistical results could help us gain a deeper understanding of how the structure ellipsis of inline comments affects code understanding activities.

3.2 Completion of Ellipsis

In order to minimize the negative effect caused by syntactic structure ellipsis of inline comments on code/comment-related software tasks, we develop a technique to help automatically complete those inline comments with syntactic structure ellipsis.

Specifically, to answer RQ3, we first create a data set for model building. Then, we build three neural network model on the data set to predict the syntactic structure ellipsis in the word level, namely LSTM, Transformer, and CodeBERT. To answer RQ4, we not only conduct a user study, but also conduct an experiment to explore the impact of inline comments that before and after ellipsis completion on code abbreviation extension task.

3.2.1 Data Set Construction

In this study, we take the following steps to obtain such a data set. First, we obtain all inline comments with no syntactic structures ellipsis, the subject/predicate/object of them is used as the ground truth for model training and prediction; and remove comments whose subjects/objects are personal/demonstrative pronouns such as "We", and "It", as personal/demonstrative pronouns have low relevance to code and rarely provide information to describe the code.

After that, we create three data sets to build models that aim to predict the exact missing subject/predicate/object structure in inline comments. Finally, we randomly divided the data set into training set, validation set, and testing set according to 8:1:1. The subject/predicate/object structure in the train set and the valid set is used as the ground truth for model training. The subject/predicate/object structure in the test set is masked to test the completion model. In addition, following [51], the duplicate data are removed.

3.2.2 Model Training & Prediction

We train three models, namely LSTM, Transformer, and CodeBERT: each of them contains three parts, one for *Subject Ellipsis* prediction, one for *Predicate Ellipsis* predication, and another for *Object Ellipsis* prediction. The input of models is the inline comments with syntactic structure ellipsis and its corresponding code. The output of models is the inline comments after syntactic structure completion at the word level. During model training and prediction, all parameters use the default values by Transformers[11] and Keras[12] (The following introduction of each model presents the concrete parameter configurations).

LSTM. We use a five-layer sequential model combined with LSTM (implemented by Keras). As a special recursive neural network (RNN), LSTM [52] could well solve the gradient vanish and gradient explosion problems, and could better handle long-distance dependence. The five layers of our model are the Embedding layer (to vectormize the input code and comment), the Dropout layer (to prevent the problem of overfitting and improve the generalization ability of the model), the Softmax layer (to define the activation function), and two LSTM layers (to retrieve higher-level semantic features). The detailed parameters of LSTM are: number of embedding layers = 10, hidden size = 128, batch size = 32, dropout = 0.2, optimizer = Adam optimizer.

Transformer. A Transformer model [53] is based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Unlike LSTM, transformers process the entire input all at once which allows for more parallelization and reduces training times. We use the following parameter settings during model training: number of layers = 6, number of heads = 8, hidden size = 512, batch size = 64, beam size = 3, initial learning rate = 0.0001, dropout = 0.1, max length of input = 128, max length of ellipsis = 16, optimizer = Adam optimizer.

CodeBERT. A CodeBERT model [54] is a bimodal pre-trained model for programming language (such as Java and Python) and natural language, which is with transformer-based neural architecture. The detailed parameter settings for training a CodeBERT model are: batch size = 64, beam size = 5, learning rate = 5e-5, max length of input = 128, max length of ellipsis = 16, optimizer = Adam optimizer.

The model training and prediction were conducted on a machine with Nvidia GTX 1080 GPU, Intel(R) Core(TM) i7-6700 CPU, and 16 GB RAM. The operating system is Ubuntu.

3.2.3 Evaluation on Models

After we obtain the ellipsis completion models, we use three traditional metrics to evaluate the performance of the model, namely Precision, Recall, and F1-score. We regard ellipsis completion as a multiple classification task and use

11. https://github.com/huggingface/transformers
12. https://keras.io/guides/

sklearn[13] to calculate these metrics (with macro-average). Specifically, for a pair of <InlineComment, Code>, the masked subjects/predicates/objects from comment without ellipsis in the test set are truly class labels (ground truth); while the subjects/predicates/objects generated by our models are predicted class labels. We first calculate the $precision_i$ and $recall_i$ for each class following (3) and (4). Then, we calculate the average of the sum of $Precision_{ma}$ and $Recall_{ma}$ following (5) and (6). The final formula for macro-F1 is following (7). TP represents that the class labels are correctly predicted, FP/FN means a negative/positive class instance is predicted as a positive/negative one.

$$precision_i = \frac{TP_i}{TP_i + FP_i} \tag{3}$$

$$recall_i = \frac{TP_i}{TP_i + FN_i} \tag{4}$$

$$Precision_{ma} = \frac{\sum_{i=1}^{n} precision_i}{n} \tag{5}$$

$$Recall_{ma} = \frac{\sum_{i=1}^{n} recall_i}{n} \tag{6}$$

$$F1_{ma} = \frac{2 \times Precision_{ma} \times Recall_{ma}}{Precision_{ma} + Recall_{ma}}. \tag{7}$$

3.2.4 Evaluation on Facilitating Code Understandability

In this section, we aim to explore whether ellipsis completion would help facilitate code understanding activities. Considering that the goal of comments is mainly to help developers understand code, hence, we design a user study to check whether ellipsis completion of inline comments would facilitate code understanding tasks. The details are as follows.

Considering that it is impossible for us to manually check all inline comments (due to its large number) after ellipsis completion, we decide to analyze a sample set of them. By following the sampling strategy in [40], [44], [45], we randomly select 384 pairs of <InlineComment, Code> with syntactic structure ellipsis. Such a sampling could let us achieve a confidence level of 95%, with sampling error within the range of $\pm$ 5%.

Then, we conduct a user questionnaire experiment (based on 384 pairs of data). Our potential experiment participants are program developers. We send our questionnaire invitations to students (in the field of software engineering) by email, we also asked our friends in the industry to help broadcast our questionnaire to their friends and colleagues who may be interested to participate in our study. In the end, we have 32 responses from students (7) and industry developers (25), all of whom have no problem in reading English books and even communicating with English native speakers. Our questionnaire includes both demographic questions and ellipsis-related questions. The demographic questions are mainly designed to understand the background and experience of participants. Specifically, we create 5 demographic questions that ask about participants' educational qualifications, experience with software engineering and work, roles (e.g., development and testing), and kinds of projects they work on (e.g., open-source and closed-source projects). The ellipsis-related questions are designed to validate the usefulness of comments before and after ellipsis completion in code understanding activities.

We assign each participant 12 <InlineComment, Code> pairs whose inline comment contains subject/predicate/object structure ellipsis, along with 12 completed pairs with predicted missing structures. They do not know in advance whether the assigned inline comments have ellipsis and whether the completion is correct or not. By using a five-point Likert scale [50], they only need to rate the pairs on how useful the inline comment is for understanding the code, where 1, 2, 3, 4, and 5 indicate that the inline comment completely unable, basically unable, partially helpful, helpful, and very helpful to understand the code. After we obtain participants' ratings, we perform a statistical analysis to calculate the minimum, 1st quartile, median, 3rd quartile, and maximum values of scores for comments with and without ellipsis, respectively. The significant differences are calculated with the help of the Mann-Whitney U test [46] and Cliff's delta [47].

3.2.5 Evaluation on Facilitating Code Abbreviation Extension

Abbreviations (e.g., "str" for "string") are common in source code. Several studies have indicated that abbreviations may have a negative impact on software engineering tasks [26], [27], [30], [55], such as program comprehension and maintenance activities [26], [56]. The potential of comments in assisting code abbreviation extension [29], [55], [57], and the tendency of developers rarely write comments in detail [35], [36], inspired us to explore whether the comments after ellipsis completion could facilitate the code abbreviation extension task in this section.

Specifically, we choose the state-of-the-art code abbreviation extension approach *Linsen* [29] that also use comments as dictionary as our evaluation basis. We replicate Linsen on comments before and after ellipsis completion. Linsen uses dictionary-based and rule-based strategies to extend abbreviations. Comparing the results could help us better understand the usefulness of our completion approach in improving code abbreviation extension, and could also help us have a glimpse into application prospect of our approach in automated software engineering tasks.

4 Experimental Results of Empirical Analysis

4.1 RQ1: Prevalence of Syntactic Structure Ellipsis

In total, we have 722,348 pairs of <InlineComment, Code> to be analyzed. Among them, there are 603,912 comments that have ellipsis. Table 4 shows the three ellipsis types as well as their possible grammar patterns. In the table, Columns 1-5 represents ellipsis type, the possible grammar patterns for each type, the meaning of different symbols used in each ellipsis type (columns 3-5), respectively. Then we analyze the prevalence of ellipsis in different code structures (e.g., If-Else Branch). Table 5 shows our results. From the table, we have the following observations.

13. https://scikit-learn.org/stable/modules/classes.html

TABLE 4
The Types of Syntactic Structure Ellipsis and the Meaning of Different Symbols in Each Ellipsis Type

Ellipsis Type	Grammer Patterns	Abbreviation Used	Full Name	Description
Ellipsis of Subject	$\langle C, In\rangle \rightarrow$ OnlyContains VP $\langle C, In\rangle \rightarrow$ OnlyContains V&VP $\langle C, In\rangle \rightarrow$ OnlyContains (V&VP) & FW $\langle C, In\rangle \rightarrow$ OnlyContains (V&VP) & FP	In C NP	Inline Comment Code Noun Phrase	Comments inside a method Comments corresponding code Noun-centered phrase
Ellipsis of Predicate	$\langle C, In\rangle \rightarrow$ OnlyContains NP $\langle C, In\rangle \rightarrow$ OnlyContains FW $\langle C, In\rangle \rightarrow$ OnlyContains FP $\langle C, In\rangle \rightarrow$ OnlyContains NP & FP $\langle C, In\rangle \rightarrow$ OnlyContains NP & FW	VP V FW	Verb Phrase Verb Functional Words	Verb-centered phrase Composed of some kind of action Composed of conjunction/ adjective/adverb
Ellipsis of Object	$\langle C, In\rangle \rightarrow$ OnlyContains NP &V $\langle C, In\rangle \rightarrow$ OnlyContains (NP &V) & FW $\langle C, In\rangle \rightarrow$ OnlyContains (NP &V) & FP	FP	Functional Phrase	Composed of (preposition/ adjective/adverb/ positional) phrase

- *Only 16.4% of inline comments contain all necessary syntactic structures* (i.e., including subject, predicate, and object). This suggests that developers may not write inline comments in detail during development and maintenance process.
- *The ellipsis of subject structure is more prevalent than the other types of ellipsis, amounting to 42.5%*. This indicates that developers mostly use predicate and object structures to write inline comments. In addition, comments with predicate ellipsis take up only 8.3% (the smallest ratio among the three types). This is in line with our common sense, as predicate structure connects the subject and the object structure and people may not tend to ignore it in daily language communications.
- *All different code structures show that only a small part of inline comments do not have syntactic structure ellipsis, which ranges from 14.3% to 21.4%*. Nevertheless, inline comments of different code structures still show some differences in syntactic structure ellipsis. Inline comments of If-Else Branch structure show the least ellipsis, with 21.4% of them containing all necessary syntactic structures; while For-While Loop structure and Code Statement show most subject ellipsis (with 49.3% and 45.8% respectively compared to the average of 42.5%).

TABLE 5
The Prevalence of Different Ellipsis in Comments for Different Code Structures

Code Structure	Ellipsis Type	Number	Rate
All Comments	None Ellipsis	118,436	16.4%
	Ellipsis of Subject	307,160	42.5%
	Ellipsis of Object	237,187	32.8%
	Ellipsis of Predicate	59,565	8.3%
	All	722,348	100%
If-Else Branch	None Ellipsis	34,859	21.4%
	Ellipsis of Subject	55,888	34.3%
	Ellipsis of Object	63,738	39.1%
	Ellipsis of Predicate	8,548	5.2%
	All	163,033	100%
For-While Loop	None Ellipsis	6,164	14.3%
	Ellipsis of Subject	21,180	49.3%
	Ellipsis of Object	12,183	28.3%
	Ellipsis of Predicate	3,473	8.1%
	All	43,000	100%
Try-Throw Exception	None Ellipsis	10,795	17.9%
	Ellipsis of Subject	21,237	35.1%
	Ellipsis of Object	22,793	37.7%
	Ellipsis of Predicate	5,619	9.3%
	All	60,444	100%
Single Statements	None Ellipsis	66,618	14.6%
	Ellipsis of Subject	208,855	45.8%
	Ellipsis of Object	138,473	30.4%
	Ellipsis of Predicate	41,925	9.2%
	All	455,871	100%

Finding 1. On the whole, most (83.6%) inline comments do have syntactic structure ellipsis. Subject ellipsis is the most common (42.5%) in inline comments. A similar phenomenon could also be observed in different code structures.

4.2 RQ2: Potential Impacts of Structure Ellipsis on Code Understandability

According to the method in Section 3.1.2, Fig. 7 present the results of manual inspection. We can observe that:

- For code with comments having syntactic structure ellipsis, the 1st quartile and 3rd quartile of the Likert Scores are at 3 and 4, respectively, with a median value of 3. This suggests that *comments with ellipsis can be partially helpful in code understanding*.
- For code with comments having no syntactic structure ellipsis, the 1st quartile and 3rd quartile of the Likert Scores are at 4 and 5, respectively, with a median value of 4. This indicates that *comments without structure ellipsis are more helpful in code understanding compared to that with ellipsis*.
- For the statistical results, the p-value of the Mann-Whitney U test between comments with and without syntactic structure ellipsis is $2.6e^{-23}$ (<0.05), indicating a significant difference from the statistical perspective. The Cliff's delta effective size is 0.39, indicating that *comments without ellipsis are more*

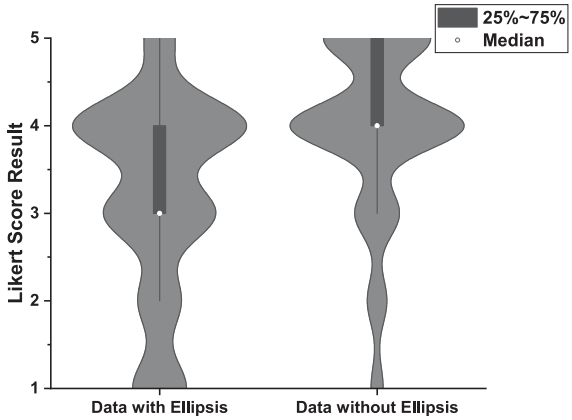

Fig. 7. The distribution of Likert scores of manual inspection on the helpfulness of comments in code understanding activities.

conducive to code understanding activities than comments with ellipsis to a medium degree.

- For the data with and without ellipsis, 16.1% of data (124/768) are scored as completely unable (score 1) and basically unable (score 2) to help code understanding activities. We further analyze this part of data and find that 4.8% (37/768) of them have inconsistency between code and comments. This indicates that *the inconsistency between code and comment would result in lower scores regardless of whether this part of data contains ellipsis or not*.

Finding 2. Inline comments without syntactic structure ellipsis are more helpful in assisting code understanding activities than those comments with syntactic structure ellipsis, with a medium effect size.

5 Experimental Results of Ellipsis Completion and Evaluation

5.1 RQ3: Completion of Structure Ellipsis of Inline Comments

We totally obtain 65,254 pairs of <InlineComment, Code>, with whose inline comments having no syntactic structures ellipsis (working as the ground truth in model building and testing) according to the data set construction approach mentioned in Section 3.2.1. Based on the data set, we build the three model introduced in Section 3.2.2. Table 6 shows the performance of the completion models. We could find that:

- Among three models, for exact word completion (i.e., precisely predicting the words that are masked from the ground truth), CodeBERT achieves the best results in complementing subject, predicate, and object ellipsis, with 62.5%, 61.0%, and 65.3% in precision, respectively.
- LSTM and Transformer perform better in subject completion while CodeBERT performs more effectively in object completion in terms of precision results. By analyzing our data set, we find that subjects tend to contain more common words, such as "the code", "this user", and "the task", while objects tend to contain more words related to code snippets. The generalizability of LSTM and Transformer used in various applications and the specificity of CodeBERT in code related tasks may be a potential reason for the performance difference of the three models in comment completion tasks.
- The precision of predicate completion is the lowest among three models, which indicates that the predicate structure is the most difficult to be predicated among the three structures.
- The recalls of three models are relatively lower compared to the precision. The not-so-high values of Recall are within our expectation, as predicting the original masked subject/predicate/object words is a strong constraint (even a simple lexical gap problem that two words expressing the same meaning would negatively affect the performance values).

TABLE 6
The Performance of Our Ellipsis Completion Model Based on Traditional Metrics

Model	Metrics	Subject	Predicate	Object
LSTM	Precision	45.5%	44.8%	41.5%
	Recall	28.9%	26.5%	28.9%
	F1-score	35.4%	33.3%	34.1%
Transformer	Precision	47.1%	40.3%	43.8%
	Recall	32.0%	35.2%	33.7%
	F1-score	38.1%	37.6%	38.1%
CodeBERT	Precision	**62.5%**	**61.0%**	**65.3%**
	Recall	40.7%	45.3%	48.4%
	F1-score	49.3%	52.0%	55.6%

Finding 3. Our neural network based ellipsis completion model could achieve a precision of 62.5%, 61.0%, and 65.3% in completing the exact missing subject/predicate/object words.

5.2 RQ4: Evaluation on Facilitating Code Understandability

Based on the method in Section 3.2.4, the results could be obtained as follows.

5.2.1 Code Understandability Based on User Study

As mentioned in Section 3.2.4, we randomly select 384 pieces of comment-code data (with 245 pieces being correctly completed and 139 pieces incorrectly completed). Then we design a questionnaire and distribute it to 32 participants.

By collecting answers to demographic questions, we find that: (1) Among the 32 participants, 21.9% of them are graduate students and 78.1% are industry developers; (2) 65.6% of participants' main role is development; (3) 84.4% of participants have advanced degrees (e.g., Master, Ph.D.); (4) The participants have different expertise in software development: 21.9% of the participants have >7 years of software development experience, and 59.4% have >4 years of experience; (5) The ratio of developers who mostly spent

TABLE 7
The Likert Rating Results on the Helpfulness of Comments in Code Understandability by Developers

	Correctly or Not	Completion	1st Quartile	Median	3rd Quartile	Mean	Difference	P-value	Effect Size	Number
All Comments	All	Before	2	3	4	2.98	0.81	$3.1e^{-19}$	0.36(Medium)*	384
		After	3	4	5	3.79				
	Correctly	Before	2	3	4	2.96	**1.0**	$1.4e^{-18}$	**0.45(Medium)***	245
		After	3	4	5	3.96				
	Incorrectly	Before	2	3	4	3.0	0.48	0.001	0.22(Small)*	139
		After	2	4	5	3.48				
Subject	All	Before	2	3	4	3.17	0.61	$5.7e^{-6}$	0.32(Small)*	129
		After	3	4	5	3.78				
	Correctly	Before	2	3	4	3.13	**0.82**	$1.6e^{-6}$	**0.42(Medium)***	83
		After	3	4	5	3.95				
	Incorrectly	Before	3	3	4	3.24	0.24	0.26	–	46
		After	2	4	5	3.48				
Predicate	All	Before	2	3	4	2.93	0.90	$7.0e^{-7}$	0.36(Medium)*	120
		After	3	4	5	3.83				
	Correctly	Before	1	3	4	2.86	**1.07**	$6.4e^{-6}$	**0.42(Medium)***	73
		After	3	4	5	3.93				
	Incorrectly	Before	2	3	4	3.04	0.62	0.02	0.26(Small)*	47
		After	3	4	5	3.66				
Object	All	Before	2	3	4	2.84	0.92	$4.e^{-9}$	0.40(Medium)*	135
		After	3	4	5	3.76				
	Correctly	Before	2	3	4	2.89	**1.11**	$6.7e^{-9}$	**0.49(Large)***	89
		After	3	5	5	4.0				
	Incorrectly	Before	2	3	4	2.74	0.56	0.04	0.24(Small)*	46
		After	2	3	5	3.30				

[1] *Correctly or Incorrectly : whether the comment is completed correctly.*
[2] *Completion means two cases : inline comments with ellipsis before completion and after completion.*
[3] **indicates significant with p-value <0.05*

time on open source projects and closed source projects or both of them are 9.4%, 53.1%, and 37.5%, respectively.

For ellipsis-related questions, Table 7 and Fig. 8 shows the results of the difference in code understandability between comments with correct and incorrect completion. By analyzing the results, we have the following observations:

- The mean value of Likert Scores gets improved for comments with correct completion (increased by 1.0 with a medium difference effect size based on Cliff's delta). There is also an improvement for comments with incorrect completion (increased by 0.48 with a small difference effect size). To get deeper insights into this observation, we provide possible explanations in the next paragraph. From a statistical perspective, *comments with correct ellipsis completion can help developers better in code understanding activities*.
- The mean value of the correctly complemented predicate and object gets larger improvement (increased by 1.07 and 1.11, which is higher than 1.0 of all data), and the mean value of the correctly complemented subject gets an improvement of 0.82. This indicates that *comments with correct predicate and object structure ellipsis completion could help developers better in code understanding*.
- The proportion of 5 in the Likert Scores increases significantly (46.9% of total data) for comments with correct completion, according to the increasing trend in Fig. 8. For comments with incorrect completion, the proportion of 5 also increased, but not as significant as those comments with correct completion. This also indicates that *comments after correctly completing the ellipsis are more helpful in code understanding activities*.

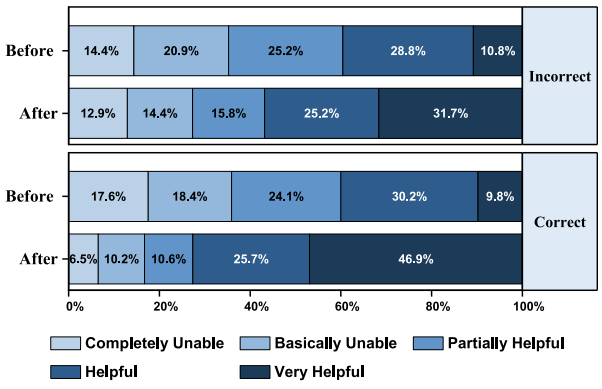

Fig. 8. The Likert rating results of the user study on the helpfulness of comments in code understandability.

To understand why there is also an improvement for comments with incorrect completion, we manually review all data that have been incorrectly completed to find possible reasons. Our findings are as follows:

- Participants give relatively high scores to the comments with incorrect completion when it is a synonym or a full extension of the ground truth (e.g., the model predicated word "Variable" is a full extension of the ground truth "Val"), or when it was a relevant word selected from the code snippet (but didn't hit ground truth).

TABLE 8
The Performance of Code Abbreviation Extension Based on Inline Comments Before and After Ellipsis Completion

Inline Comments	Components	Numbers	Before or After Completion	The Number of Abbreviation	The Number of Extension	Precision	Recall	Rate of Increase
All Data	Subject Structure	128	Before	118	77	65.3%	65.3%	
			After	118	83	70.3%	70.3%	7.7%
	Predicate Structure	126	Before	115	72	62.6%	62.6%	
			After	115	72	62.6%	62.6%	–
	Object Structure	130	Before	145	95	65.5%	65.5%	
			After	145	105	72.4%	72.4%	10.5%
	All	384	Before	378	244	64.6%	64.6%	
			After	378	260	68.8%	68.8%	6.5%
Correctly Completed	Subject Structure	77	Before	66	41	62.1%	62.1%	
			After	66	47	71.2%	71.2%	14.7%
	Predicate Structure	75	Before	62	38	61.3%	61.3%	
			After	62	38	61.3%	61.3%	–
	Object Structure	91	Before	96	64	66.7%	66.7%	
			After	96	74	77.1%	77.1%	15.6%
	All	243	Before	224	143	63.8%	63.8%	
			After	224	159	71.0 %	71.0%	11.3%

- Developers may naturally think that comment sentences with complete syntactic structure and conform to linguistic syntactic norms are easier to understand, regardless of whether the comments are completed correctly or not.

Finding 4. The completion of comments with ellipsis (especially correct completion) could improve the code understandability in a medium effect size.

5.3 RQ4: Evaluation on Facilitating Code Abbreviation Extension Work

We select the code abbreviation extension algorithm Linsen [29] for evaluation because it is accurate and well known, representing the state-of-the-art. After applying Linsen, we obtain 1,358 code abbreviations. We manually review all abbreviations to filter out those for which we could not find the corresponding expansions; and finally obtain a total of 378 abbreviations for expansion (as the ground truth). Table 8 shows the expansion results of Linsen towards 378 abbreviations based on inline comments before and after syntactic structure completion. From the table, we have the following observations. Note that, since Linsen only considered the precision performance metric, in this study, we also only consider the precision when studying how comment ellipsis completion could improve the performance of Linsen.

- The precision of abbreviation extension after subject completion increases from 65.3% to 70.3%, with an increase of 7.7% $((70.3 - 65.3)/65.3)$; and the precision value after object structure completion increases from 65.5% to 72.4%, with a higher increase of 10.5%. This indicates that *object structures contain more words related to code abbreviation*.
- The precision of abbreviation extension does not get improved (almost keep the same actually) after predicate completion compared to that of subject/object completion. One reason for such a difference may be that the missing subject/object in a comment is usually a noun, gerund, etc., and usually refers to variables/identifiers in the code while the predicate usually refers to the action word in the code, and generally is not abbreviated by developers; variables/identifiers are more likely to be shortened as an abbreviation and hence *subject/object completion is more helpful for code abbreviation extension*.

The above analysis is on all comments without distinguishing comments with correct or incorrect ellipsis completion. We further apply Linsen on comments with correct ellipsis completions to better understand how comment ellipsis completion could facilitate the code abbreviation task under an ideal situation. The results are shown in rows 11-18 (Correctly Completed) in Table 8. From Table 8, we have the following observations:

- The precision is greatly improved for correctly completed comments. After completing the subject structure, the precision is improved from 62.1% to 71.2%, with an increase of 14.7%. After completing the object structure, the precision is improved from 66.7% to 77.1%, with an increase of 15.6%. This also indicates that *object structures contain more words related to code abbreviation*.
- The precision of all 224 code abbreviations is improved by 11.3% on average (from 63.8% to 71.0%). There is an 4.8% $(11.3\% - 6.5\%)$ increase over all data, indicating that *a noticeable improvement could be achieved in the extension of code abbreviation after correctly ellipsis completions*.

Finding 5. On the whole, the completion of comment ellipsis could improve the precision of code abbreviation extension by 6.5%. And inline comments with correct completions could largely improve the precision by 11.3%.

6 Discussion

6.1 Implications for Researchers, Practitioners, and Tool Providers

Implications for Researchers. First, the exploration of different types of syntactic structures ellipsis would provide foresight

in the quality of software documentation. This is indicated by the result that correct completion of the ellipsis in comments could improve code understandability with a medium effect size (*Finding 4*). In the future, it would be interesting to explore other types of syntactic structure, such as complements (complements are also a component of sentence structure, which plays an important role in specific situations). The ellipsis of Javadoc may be worthy to explore in the future. The result of our preliminary exploration indicates that Javadoc has 28.5% pairs of data without structure ellipsis and the distribution and completion of types of ellipsis could be further explored. In addition, we build three models to complete the ellipsis in inline comments and achieved a precision of 65.3% (*Finding 3*), more advanced methods are expected to further improve the performance.

Second, Inline comments show great potential in the field of code abbreviation extension. Precision of the code abbreviation extension task could be improved by 11.3% in our research if the inline comments are correctly completed (*Finding 5*). It would be valuable to explore whether inline comments with ellipsis being completed would facilitate other code/comment related tasks in the future.

Third, the inconsistency of inline comments and code are also worth exploring, with 4.8% of inline comments in our initial review being nearly unable to help code understanding activity (results in Section 4.2), which indicates that the developer may neglect to make changes to the comments when conducting the code change activity, resulting in outdated comments.

Implications for Practitioners. As a important component of software document, comments are always essential for software comprehension and maintenance. Therefore, first, in programming activities, developers may need to be more patient when writing inline comments. This is indicates by the result that 83.6% of inline comments contain ellipsis and subject and object ellipsis is the most common (*Finding 1*).

Second, in code understanding activities, developers may need to spend more effort on the inline comments that with ellipsis. This is indicates by the result that comments without ellipsis are more helpful in assisting code understanding activities with a medium effect size (*Finding 2*). In addition, in 8.2% of the method (from 1,000 Java projects), developers can only use inline comments to conduct code understanding activity, which indicates that inline comments also play an important role.

Implications for Tool Developers. This work can motivate tool developers to develop some IDE plugins that can provide timely feedback once a developer finishes writing a comment, e.g., by suggesting possible missing content. Such feedback can work as an assistant for developers to improve their code/comment understandability and facilitate code/comment related tasks such as identifier abbreviation extensions (in case they use extension tools). This is indicated by our results that the completion of inline comments with ellipsis can improve the corresponding code's understandability in a medium size (*Finding 4*) and can improve the precision of code abbreviation extension (*Finding 5*).

6.2 Validity Discussion

Threats to Construct Validity. In the dataset construction (Section 3.1.1), we use some heuristic rules to associate comments with code like [39]. Despite that we obtained an accuracy of 95% in our case (sampled data with a confidence level of 95% and a sampling error range of $\pm$5%). We cannot guarantee that these heuristic rules could always obtain such an or even higher accuracy in any case.

Moreover, in the evaluation on facilitating code abbreviation extension part (Section 5.3), we need to replicate the comment-assist abbreviation extension technique Linsen. Since the code of Linsen is not open-sourced, we cannot guarantee that our implementation is 100% correct. To alleviate this threat, we have done several code reviews about the implementation.

Threats to Internal Validity. In the analysis of the potential impact of ellipsis on code understandability (Section 4.2), we directly use the original comments sampled from our datasets. These inline comments may not evolve with code, which leads to outdated inline comments that negatively impact code understanding. Completing the structure ellipsis for these outdated inline comments may not be so useful in practice. A more practical way may be to use several state-of-the-art techniques in identifying and correcting such inconsistencies first (such as [58], [59], [60], [61]), and then applying our ellipsis completion approach.

Threats to External Validity. In the evaluation on facilitating code understandability part (Section 5.2.1), we conduct a user study to evaluate whether comment ellipsis completion could improve code understandability. There are totally of 32 developers participating in our user study, which is not an extremely large number. Therefore, we cannot guarantee that our findings obtained by the user study could be generalized to all software developers. However, the participants of this study have different educational qualifications, experience levels, and contribute to various projects (i.e., open-source and closed-source projects). Such diversity in backgrounds could make us believe that our survey results still provided valuable insights into the usefulness of our ellipsis completion task.

Another threat is that all our experiments are conducted on only open source Java projects. We cannot guarantee that our conclusions are applicable to industrial or projects in other programming languages. Considering that Java is widely used by developers in developing software projects [40], [62], [63] and all experimental projects are popular and mature projects, we believe that our findings could still provide insights into the syntactic structure ellipsis problem of inline comments.

Threats to Conclusion Validity. In this study, we perform three types of manual reviews separately in (1) checking the validity of our heuristic rules in associating code with comments (Section 3.1.1), (2) evaluating the impact of ellipsis on code understandability (Section 4.2), and (3) constructing code abbreviation dataset for evaluation experiments (Section 5.3). All participants evolved in manual reviews are not the owners (developers) of the code, we cannot guarantee that all judgments made by the participants are correct. In order to reduce these biases, we cross-checked the data by one Ph.D., one Ph.D. candidate, and one graduate student (non-author programmer) and solved disagreements through discussions.

In the evaluations on facilitating code understandability (Section 4.2) and code abbreviation extension (Section 5.2),

we conduct some statistical tests to check whether our results have statistically significance. These tests require that our datasets meet certain distribution assumptions (e.g., normal distribution or not). Despite we have verified that our datasets meet the requirements before applying the tests, we cannot claim more than what these tests tell us, not to mention generalize them to other scenarios.

7 Related Work

Relevant research work could be roughly divided into two categories, namely comment quality analysis, and comment classification.

7.1 Comment Quality Analysis

Existing studies on quality analysis of code comments mainly focus on analyzing the density of comments, the correlation between comments and code, and the evolution of comments and code [23], [61], [64], [65], [66], [67]. Arafat et al. [68] used the metric of comment density and found that commenting source code was an ongoing and integrated practice, independent of the chosen programming language, the age of project, team size and project size. McBurney et al. [64] found that comments written by subsequent developers had a higher semantic similarity than those by the original developers. Padioleau et al. [35] studied the programming comments from the latest versions of Linux, FreeBSD, and OpenSolaris. They found that approximately 52.6% of comments could improve software reliability or programmer productivity. Shinyama et al. [62] presented a framework for analyzing comments in detail. They discussed the extent, target and category of comments in both Java and Python projects, and validated there is a universal "grammar" for code comments.

The relevance and consistency of comments and their corresponding code were also taken into consideration by researchers. Rabbi et al. [58] used a recurrent network to detect inconsistency between comment and code. They [59] also proposed an ensemble approach to detect code comment inconsistency by using topic modeling. Stulova et al. [60] proposed a technique, named upDoc, to automatically detect code comment inconsistency during code evolution. Khamis et al. [61] designed JavaDocMiner to automatic evaluate comment quality by analyzing the consistency relationship between comments and code.

Many researchers focused on how code and comments changed during software evolution. Wen [69] found that making changes to different types of code had different probabilities in triggering an update of comments. Ratol [36] took comments corresponding to code entities in Java code as fragile comments and proposed a rule-based method Fraco to detect such comments.

There have been a number of studies on detection or management of technical debt comments [70], [71] or self-admitted technical debt [72]. Nie et al. [73] presented the first framework to specify trigger-action todo comments to benefit the code comprehension and maintenance.

The abovementioned studies considered comment quality mainly from comment density and the evolution of comment with related code. It would be valuable for future studies to consider comment quality from the syntactic structure view for example by taking syntactic ellipsis as a comment quality measurement metric. This may be actionable as our study have revealed that syntactic ellipsis of comments would negatively affect code/comment related tasks such code/comment comprehension.

7.2 Comment Classification

Classifying comments can help us better understand the performance of comments under different categories. At present, there is no unified standard for the classification of comments. Haouari et al. [74] conducted an empirical study by analyzing existing comments in different open source Java projects from both quantitative and qualitative views. They defined thirteen categories from four dimensions and found that comments are intended to explain the code that follows them in most cases. They aimed to study the developers' habit of writing comments by proposing this taxonomy of comments.

Padioleau et al. [35] mainly studied comments from the perspectives of What, Who, Where, and When. Martin et al. [75] studied the classification of API documents. Steidl et al. [76] defined a preliminary taxonomy of comments comprising seven high-level categories. Pascarella and Bacchelli [40] studied the comments of six open source Java projects, classified the comments into a hierarchy of six categories and sixteen subcategories, and used machine learning methods to automatically classify code comments. Zhai et al. [34] treated comments as related attributes of code entities, and classified them at a fine granularity from different code perspectives. Chen et al. [17] classified the comments into types such as What, Why, and How-to-use.

Several studies analyzed types of self-admitted technical debt described in comments [71], [77], [78], [79], [80]. Ying et al. [81] attempted to explore the types of todo comments or task comments and presented a categorization of Eclipse task comments on the AWB codebase.

The abovementioned studies mainly classify comments based on their ability in describing code. It is worth research efforts to classify comments based on their syntactic structure characteristics. In addition, existing studies mainly target at JavaDoc comments, with paying little attention to inline comments. Given that inline comments play an important role in code understanding and its substantial format difference with JavaDoc (Inline comments are written in a more casual way by developers than JavaDoc comments), it would be valuable to develop new classification methods for inline comments.

8 Conclusion

In this article, we conducted the first large-scale empirical study of inline comments from the syntactic structure ellipsis perspective. We collected a large data set of inline comments with their corresponding code in Java projects. Based on the large data set, we analyzed the syntactic structure of comments, and identified the types of syntactic structure ellipsis of inline comments. Then, we uncovered the prevalence of syntactic structure ellipsis in inline comments and analyzed the impact of ellipsis on the code understanding activities. Our research indicated that when considering the context of the code, comments with ellipsis would

negatively affect (with a medium effect size) the code/comment understanding activities. Last, we tried to complete the syntactic structure ellipsis of inline comments by building neural network model and demonstrated the effectiveness of our completion approach in facilitating code understanding and code abbreviation expansion tasks. In terms of code understanding activities, the results of our user study showed that comments that correctly completed the syntactic structure could assist code understanding activities to a medium degree. As for code abbreviation expansion task, we found that the completion of inline comments with syntactic structure ellipsis could improve the precision of general code abbreviation expansion technique by 11.3%. Our explorations are expected to facilitate the research on code comments and comments-related software tasks.

REFERENCES

[1] N. Chomsky, *Syntactic Structures*, Berlin, Germany: Walter de Gruyter, 2002.

[2] P. Khullar, K. Majmundar, and M. Shrivastava, "NoEl: An annotated corpus for noun ellipsis in English," in *Proc. 12th Lang. Resour. Eval. Conf.*, 2020, pp. 34–43.

[3] G. Güneş and A. Lipták, *The Derivational Timing of Ellipsis*, vol. 79, Oxford, U.K.: Oxford Univ. Press, 2022.

[4] M. A. Saleem, "A study of ellipsis and elision in English," *J. Tikrit Univ. Humanities*, vol. 28, no. 2, pp. 60–81, 2021.

[5] G. Zhao, "The motivation of ellipsis," *Theory Pract. Lang. Stud.*, vol. 5, no. 6, 2015, Art. no. 1275.

[6] M. C. Kim, J. Park, and W. Jung, "Sentence completeness analysis for improving team communications of safety-critical system operators," *J. Loss Prevention Process Industries*, vol. 21, no. 3, pp. 255–259, 2008.

[7] N. Kim, L. Brehm, and M. Yoshida, "The online processing of noun phrase ellipsis and mechanisms of antecedent retrieval," *Lang. Cogn. Neurosci.*, vol. 34, no. 2, pp. 190–213, 2019.

[8] V. P. B. Hansen and A. Søgaard, "What do you mean 'why?': Resolving sluices in conversations," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 7887–7894.

[9] X. Ren, X. Sun, J. Wen, B. Wei, W. Zhan, and Z. Zhang, "Building an ellipsis-aware chinese dependency treebank for web text," in *Proc. 11th Int. Conf. Lang. Resour. Eval.*, 2018, pp. 1749–1754.

[10] S. Petrov and R. McDonald, "Overview of the 2012 shared task on parsing the web," *First Workshop Syntactic Anal. Non-Canonical Lang.*, vol. 59, pp. 1–8, 2012.

[11] P. Khullar, A. Antony, and M. Shrivastava, "Using syntax to resolve NPE in English," in *Proc. Int. Conf. Recent Adv. Natural Lang. Process.*, 2019, pp. 534–540.

[12] W.-N. Zhang, Y. Zhang, Y. Liu, D. Di, and T. Liu, "A neural network approach to verb phrase ellipsis resolution," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 7468–7475.

[13] E. Lapshinova-Koltunski, C. Hardmeier, and P. Krielke, "ParCorFull: A parallel corpus annotated with full coreference," in *Proc. 11th Int. Conf. Lang. Resour. Eval.*, 2018, pp. 423–428.

[14] S. M. Bouzid and C. B. O. Zribi, "Efficient learning approach for pronominal anaphora and ellipsis identification and resolution in arabic texts," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 29, no. 10, pp. 3335–3348, Oct. 2021.

[15] K. Kenyon-Dean, J. C. K. Cheung, and D. Precup, "Verb phrase ellipsis resolution using discriminative and margin-infused algorithms," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2016, pp. 1734–1743.

[16] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proc. 23rd Annu. Int. Conf. Des. Commun.: Documenting Designing Pervasive Inf.*, 2005, pp. 68–75.

[17] Q. Chen, X. Xia, H. Hu, D. Lo, and S. Li, "Why my code summarization model does not work: Code comment improvement with category prediction," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 2, pp. 1–29, 2021.

[18] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, "Summarizing source code with transferred API knowledge," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 2269–2275.

[19] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation with hybrid lexical and syntactical information," *Empir. Softw. Eng.*, vol. 25, no. 3, pp. 2179–2217, 2020.

[20] P. Rani, S. Panichella, M. Leuenberger, A. Di Sorbo, and O. Nierstrasz, "How to identify class comment types? A multi-language approach for class comment classification," *J. Syst. Softw.*, vol. 181, 2021, Art. no. 111047.

[21] S. B. Merriam and E. J. Tisdell, *Qualitative Research: A Guide to Design and Implementation*, Hoboken, NJ, USA: Wiley, 2015.

[22] Y. Huang, N. Jia, Q. Zhou, X. Chen, Y. Xiong, and X. Luo, "Guiding developers to make informative commenting decisions in source code," in *Proc. 40th Int. Conf. Softw. Eng.: Companion Proc.*, 2018, pp. 260–261.

[23] B. Yang, Z. Liping, and Z. Fengrong, "A survey on research of code comment," in *Proc. 3rd Int. Conf. Manage. Eng., Softw. Eng. Service Sci.*, 2019, pp. 45–51.

[24] J. L. Freitas, D. da Cruz, and P. R. Henriques, "A comment analysis approach for program comprehension," in *Proc. IEEE 35th Annu. IEEE Softw. Eng. Workshop*, 2012, pp. 11–20.

[25] Y. Shinyama, Y. Arahori, and K. Gondow, "Analyzing code comments to boost program comprehension," in *Proc. IEEE 25th Asia-Pacific Softw. Eng. Conf.*, 2018, pp. 325–334.

[26] Y. Jiang, H. Liu, J. Jin, and L. Zhang, "Automated expansion of abbreviations based on semantic relation and transfer expansion," *IEEE Trans. Softw. Eng.*, vol. 48, no. 2, pp. 519–537, Feb. 2022.

[27] Y. Jiang, H. Liu, J. Q. Zhu, and L. Zhang, "Automatic and accurate expansion of abbreviations in parameters," *IEEE Trans. Softw. Eng.*, vol. 46, no. 7, pp. 732–747, Jul. 2020.

[28] H. Liu, Q. Liu, C.-A. Staicu, M. Pradel, and Y. Luo, "Nomen est omen: Exploring and exploiting similarities between argument and parameter names," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 1063–1073.

[29] A. Corazza, S. Di Martino, and V. Maggio, "LINSEN: An efficient approach to split identifiers and expand abbreviations," in *Proc. IEEE 28th Int. Conf. Softw. Maintenance*, 2012, pp. 233–242.

[30] Y. Jiang, H. Liu, and L. Zhang, "Semantic relation based expansion of abbreviations," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 131–141.

[31] L. Tan, Y. Zhou, and Y. Padioleau, "aComment: Mining annotations from comments and code to detect interrupt related concurrency bugs," in *Proc. IEEE 33rd Int. Conf. Softw. Eng.*, 2011, pp. 11–20.

[32] S. H. Tan, D. Marinov, L. Tan, and G. T. Leavens, "@tComment: Testing javadoc comments to detect comment-code inconsistencies," in *Proc. IEEE 5th Int. Conf. Softw. Testing, Verification Validation*, 2012, pp. 260–269.

[33] C. Rubio-González and B. Liblit, "Expect the unexpected: Error code mismatches between documentation and the real world," in *Proc. 9th ACM SIGPLAN-SIGSOFT Workshop Prog. Anal. Softw. Tools Eng.*, 2010, pp. 73–80.

[34] J. Zhai et al., "CPC: Automatically classifying and propagating natural language comments via program analysis," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng.*, 2020, pp. 1359–1371.

[35] Y. Padioleau, L. Tan, and Y. Zhou, "Listening to programmers–taxonomies and characteristics of comments in operating system code," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, 2009, pp. 331–341.

[36] I. K. Ratol and M. P. Robillard, "Detecting fragile comments," in *Proc. IEEE/ACM 32nd Int. Conf. Autom. Softw. Eng.*, 2017, pp. 112–122.

[37] J. Bowers, "The syntax of predication," *Linguistic Inquiry*, vol. 24, no. 4, pp. 591–656, 1993.

[38] X. Ren, X. Sun, J. Wen, B. Wei, W. Zhan, and Z. Zhang, "Building an ellipsis-aware chinese dependency treebank for web text," 2018, *arXiv:1801.06613*.

[39] Z. Liu, H. Chen, X. Chen, X. Luo, and F. Zhou, "Automatic detection of outdated comments during code changes," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf.*, 2018, pp. 154–163.

[40] L. Pascarella and A. Bacchelli, "Classifying code comments in java open-source software systems," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories*, 2017, pp. 227–237.

[41] T. D. Nguyen, A. T. Nguyen, and T. N. Nguyen, "Mapping API elements for code migration with vector representations," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. Companion*, 2016, pp. 756–758.

[42] P. Leitner and C.-P. Bezemer, "An exploratory study of the state of practice of performance testing in java-based open source projects," in *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng.*, 2017, pp. 373–384.

[43] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 92–101.
[44] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, "Retrieval-based neural source code summarization," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng.*, 2020, pp. 1385–1397.
[45] Y. Jiang, H. Liu, Y. Zhang, N. Niu, Y. Zhao, and L. Zhang, "Which abbreviations should be expanded?," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 578–589.
[46] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 1947. [Online]. Available: https://doi.org/10.1214/aoms/1177730491
[47] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.
[48] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu, "How practitioners perceive automated bug report management techniques," *IEEE Trans. Softw. Eng.*, vol. 46, no. 8, pp. 836–862, Aug. 2020.
[49] F. Liu, G. Li, B. Wei, X. Xia, Z. Fu, and Z. Jin, "A unified multi-task learning model for ast-level and token-level code completion," *Empir. Softw. Eng.*, vol. 27, no. 4, pp. 1–38, 2022.
[50] R. A. Likert, "A technique for measurement of attitudes," *Arch. Psychol. New York*, vol. 22, no. 140, pp. 1–55, 1932.
[51] M. Allamanis, "The adverse effects of code duplication in machine learning models of code," in *Proc. ACM SIGPLAN Int. Symp. New Ideas, New Paradigms, Reflections Program. Softw.*, 2019, pp. 143–153.
[52] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[53] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
[54] Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," 2020, *arXiv:2002.08155*.
[55] D. Lawrie, H. Feild, and D. Binkley, "Extracting meaning from abbreviated identifiers," in *Proc. IEEE 7th Int. Work. Conf. Source Code Anal. Manipulation*, 2007, pp. 213–222.
[56] D. Binkley and D. Lawrie, "The impact of vocabulary normalization," *J. Softw.: Evol. Process*, vol. 27, no. 4, pp. 255–273, 2015.
[57] N. Madani, L. Guerrouj, M. Di Penta, Y.-G. Gueheneuc, and G. Antoniol, "Recognizing words from source code identifiers using speech recognition techniques," in *Proc. IEEE 14th Eur. Conf. Softw. Maintenance Reengineering*, 2010, pp. 68–77.
[58] F. Rabbi and M. S. Siddik, "Detecting code comment inconsistency using siamese recurrent network," in *Proc. 28th Int. Conf. Prog. Comprehension*, 2020, pp. 371–375.
[59] F. Rabbi, M. N. Haque, M. E. Kadir, M. S. Siddik, and A. Kabir, "An ensemble approach to detect code comment inconsistencies using topic modeling," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, 2020, pp. 392–395.
[60] N. Stulova, A. Blasi, A. Gorla, and O. Nierstrasz, "Towards detecting inconsistent comments in Java source code automatically," in *Proc. IEEE 20th Int. Work. Conf. Source Code Anal. Manipulation*, 2020, pp. 65–69.
[61] N. Khamis, R. Witte, and J. Rilling, "Automatic quality assessment of source code comments: The javadocminer," in *Proc. Int. Conf. Appl. Natural Lang. Inf. Syst.*, 2010, pp. 68–79.
[62] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: A large-scale field study with professionals," *IEEE Trans. Softw. Eng.*, vol. 44, no. 10, pp. 951–976, Oct. 2018.
[63] S. Cass, "The 2018 top programming languages," *IEEE Spectr.*, vol. 31, 2018, Art. no. 1.
[64] P. W. McBurney and C. McMillan, "An empirical study of the textual similarity between source code and source code summaries," *Empir. Softw. Eng.*, vol. 21, no. 1, pp. 17–42, 2016.
[65] T. M. Khoshgoftaar, Y. Xiao, and K. Gao, "Software quality assessment using a multi-strategy classifier," *Inf. Sci.*, vol. 259, pp. 555–570, 2014.
[66] Y. Liu, X. Sun, and Y. Duan, "Analyzing program readability based on WordNet," in *Proc. 19th Int. Conf. Eval. Assessment Softw. Eng.*, 2015, Art. no. 27.
[67] X. Sun, Q. Geng, D. Lo, Y. Duan, X. Liu, and B. Li, "Code comment quality analysis and improvement recommendation: An automated approach," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 06, pp. 981–1000, 2016.
[68] O. Arafat and D. Riehle, "The commenting practice of open source," in *Proc. 24th ACM SIGPLAN Conf. Companion Object Oriented Program. Syst. Lang. Appl.*, 2009, pp. 857–864.
[69] F. Wen, C. Nagy, G. Bavota, and M. Lanza, "A large-scale empirical study on code-comment inconsistencies," in *Proc. IEEE/ACM 27th Int. Conf. Prog. Comprehension*, 2019, pp. 53–64.
[70] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2014, pp. 91–100.
[71] E. Maldonado, E. Shihab, and N. Tsantalis, "Using natural language processing to automatically detect self-admitted technical debt," *IEEE Trans. Softw. Eng.*, vol. 43, no. 11, pp. 1044–1062, Nov. 2017.
[72] Z. Guo et al., "How far have we progressed in identifying self-admitted technical debts? A comprehensive empirical study," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 4, pp. 1–56, 2021.
[73] P. Nie, R. Rai, J. J. Li, S. Khurshid, R. J. Mooney, and M. Gligoric, "A framework for writing trigger-action todo comments in executable format," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 385–396.
[74] D. Haouari, H. A. Sahraoui, and P. Langlais, "How good is your comment? A study of comments in Java programs," in *Proc. IEEE 5th Int. Symp. Empir. Softw. Eng. Meas.*, 2011, pp. 137–146.
[75] W. Maalej and M. P. Martin, "Patterns of knowledge in API reference documentation," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1264–1282, Sep. 2013.
[76] D. Steidl, B. Hummel, and E. Jürgens, "Quality analysis of source code comments," in *Proc. IEEE 21st Int. Conf. Prog. Comprehension*, 2013, pp. 83–92. [Online]. Available: https://doi.org/10.1109/ICPC.2013.6613836
[77] R. M. Santos, M. C. R. Junior, and M. G. D. Mendonça Neto, "Self-admitted technical debt classification using LSTM neural network," in *Proc. 17th Int. Conf. Inf. Technol.–New Gener.*, 2020, pp. 679–685.
[78] M. Farias, M. Neto, M. Kalinowski, and R. Spínola, "Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary," *Inf. Softw. Technol.*, vol. 121, 2020, Art. no. 106270.
[79] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "Is using deep learning frameworks free?: Characterizing technical debt in deep learning frameworks," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng.: Softw. Eng. Soc.*, 2020, pp. 1–10.
[80] G. Bavota and B. Russo, "A large-scale empirical study on self-admitted technical debt," in *Proc. IEEE/ACM 13th Work. Conf. Mining Softw. Repositories*, 2016, pp. 315–326.
[81] A. T. Ying, J. L. Wright, and S. Abrams, "Source code that talks: An exploration of eclipse task comments and their implication to repository mining," *ACM SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, 2005.
[82] G. Spache, "A new readability formula for primary-grade reading materials," *Elementary Sch. J.*, vol. 53, no. 7, pp. 410–413, 1953.
[83] H. Antunes and C. T. Lopes, "Analyzing the adequacy of readability indicators to a non-english language," in *Proc. Int. Conf. Cross-Lang. Eval. Forum Eur. Lang.*, 2019, pp. 149–155.
[84] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2008, pp. 308–318.

Xiaowei Zhang is currently working toward the PhD degree with the Department of Computer Science and Technology, Nanjing University. Her research interest is intelligent software analysis.

Weiqin Zou received the bachelor's degree in software engineering and the master's degree in computer science from the Dalian University of Technology, and the PhD degree in software engineering from Nanjing University. She is currently an associate professor in College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. Her research interests include empirical study and mining software repositories.

Lin Chen received the PhD degree in computer science from Southeast University, in 2009. He is currently an associate professor with the Department of Computer Science and Technology, Nanjing University. His research interests include software analysis and software maintenance.

Yanhui Li (Member, IEEE) received the BS, MS, and PhD degrees in computer science from Southeast University. He is currently an assistant professor with the Department of Computer Science and Technology, Nanjing University. His main research interests include AI testing and debugging, empirical software engineering, software analysis, knowledge engineering, and formal methods. He is a member of IEEE.

Yuming Zhou received the PhD degree in computer science from Southeast University, in 2003. He is currently a professor with the State Key Laboratory for Novel Software Technology and the Department of Computer Science and Technology, Nanjing University. His main research interests are software testing, defect prediction/detection, and program analysis.