Improving API Knowledge Comprehensibility: A Context-Dependent Entity Detection and Context Completion Approach using LLM

***Abstract*—Extracting API knowledge from Stack Overflow has become a crucial way to assist developers in using APIs. Existing research has primarily focused on extracting relevant API-related knowledge at the sentence level to enhance API documentation. However, this approach can lead to a loss of crucial context, especially when sentences contain context-dependent entities (i.e., understanding them requires referring to the surrounding context), which may hinder developers' understanding. To investigate this issue, we conducted an empirical study of 384 Stack Overflow posts and found that (1) approximately one-third of API functionality sentences contain context-dependent entities, and (2) these entities fall into two categories: Referential Context-Dependent Entities and Local Variable Context-Dependent Entities. In response, we developed a novel method, CEDCC, which combines an entity filtering strategy informed by insights from our empirical study, with a large language model (LLM) to construct coreference chains for detecting context-dependent entities. Additionally, it employs a step-by-step approach with the LLM to complete the necessary context for understanding these entities. To evaluate CEDCC, we constructed a dataset of 1,023 API knowledge sentences, including 574 context-dependent entities and their required contexts. The results show that CEDCC significantly outperforms baseline methods in both detecting context-dependent entities and completing context tasks, achieving an F1-score of 0.86 and a BERTScore of 0.397. Human evaluations further confirmed that CEDCC effectively improves the comprehensibility of API knowledge sentences.**

***Index Terms*—API Knowledge, Context-dependent, LLM**

I. Introduction

In modern software development, Application Programming Interfaces (APIs) are crucial for software reuse, but their documentation often lacks practical examples and detailed explanations [1], [2]. As a result, developers frequently turn to online communities like Stack Overflow (SO) for real-world advice and usage examples. Existing research has aimed to extract relevant API-related knowledge from SO to enhance API documentation [3], [4]. These knowledge, e.g., API functionality and constraints, are obtained by identifying posts mentioning specific APIs and considering their contents as related knowledge for those APIs [5], [6].

Researchers have employed different levels of granularity in their extraction methods, e.g., post-level [7] (extracting entire questions, answers, or comments), and sentence-level [8] (isolating individual sentences that specifically contain API-related information). Take the SO post in Fig. 1 as an example. In this post, the API *matplotlib.pyplot.subplots_adjust* is mentioned. When extracting knowledge at the post level, all sentences

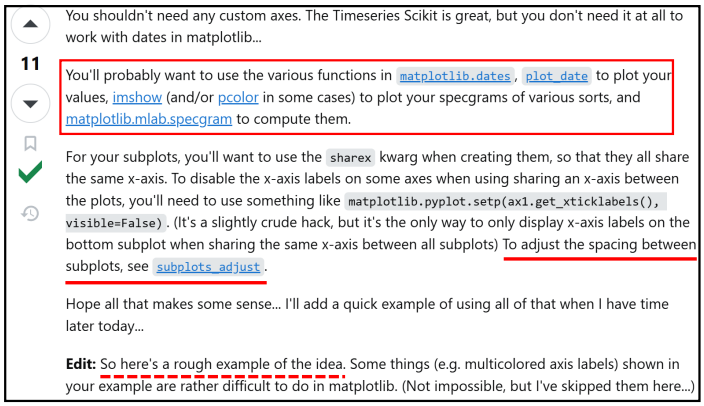

Fig. 1. An example from Stack Overflow illustrating API knowledge extraction at the post and sentence Levels.

within the post are considered as relevant knowledge to this API. This results in the extraction of not only sentences that describe the functionality of the API (highlighted with red underlines) but also API-unrelated content, such as "So here's a rough example of the idea" (highlighted with red dotted lines). These sentences introduce unnecessary information and hinder developers' ability to identify the key details. In contrast, sentence-level extraction targets only the sentence marked with red underlines, thereby reducing redundancy and emphasizing the most important information. Therefore, sentence-level extraction has become the preferred approach [9], [10].

Although sentence-level extraction has many advantages, it also leads to a loss of context. Take another sentence in Fig. 1 for example: "... *plot_date* to plot your values" (highlighted with a red box). This sentence describes the functionality of the *matplotlib.pyplot.plot_date* API by indicating its use—"plot your values." However, fully understanding what "your values" refers to requires additional context (i.e., other sentences that were not extracted during the sentence-level extraction). Therefore, we call "your values" **context-dependent entities**, and extracting this sentence alone as a functional description of *matplotlib.pyplot.plot_date* could undermine developers' understanding and reduce its practicality. In contrast, the sentence describing the functionality of *matplotlib.pyplot.subplots_adjust* marked with red underlines in Fig. 1 can be understood without additional context, indicating that it does not contain context-dependent entities. Thus,

detecting context-dependent entities within API knowledge sentences and providing the necessary context are crucial for improving the comprehensibility of the extracted API knowledge.

Motivated by this, we conducted an empirical study of 384 SO posts to explore the characteristics of context-dependent entities within API knowledge sentences. Our findings revealed that: (1) approximately 1/3 of API knowledge sentences contain context-dependent entities; (2) there are two types of context-dependent entities: Referential Context-Dependent Entities and Local Variable Context-Dependent Entities; and (3) the context required to assist in understanding these entities is often distributed both within the post containing the API sentence and across other posts. These results underscore the need to detect context-dependent entities and complete the necessary context, which has inspired the design of our correspondence detection and completion approach.

Based on these findings, we developed CEDCC (**C**ontext-dependent **E**ntity **D**etection and **C**ontext **C**ompletion), an approach designed to improve the comprehensibility of API knowledge. This method takes an API knowledge sentence as input and identifies both the context-dependent entities within it and the context necessary for understanding them. Specifically, in the detection stage, we adopt corresponding filtering strategies to filter candidate entities for each type of context-dependent entity identified in our empirical study, leveraging syntactic and code analysis. Next, we utilize a large language model (LLM) to construct coreference chains (i.e., a sequence of phrases in a text that refer to the same entity [11]) that include the candidate entities. A rule-based analysis is then performed on these chains to detect context-dependent entities. In the completion stage, we implement a step-by-step context completion process using the LLM. The LLM first extracts context related to the context-dependent entity from SO posts. Drawing on our empirical analysis of the distribution of required context, we limit the scope of SO posts to the SO thread where the entity appears, including the question, accepted answer, other answers, and their comments. Finally, the LLM uses the selected relevant context to generate the necessary context to aid in comprehending the entities.

We further constructed a specific dataset to evaluate the performance of CEDCC. This dataset consists of 1,023 API knowledge sentences from five popular Python libraries—matplotlib, pandas, numpy, scipy, and sklearn—and contains 574 context-dependent entities along with the corresponding contexts required for their understanding. In the context-dependent entity identification stage, CEDCC achieved a precision of 0.912, a recall of 0.814, and an F1 score of 0.860. In the context completion stage, CEDCC achieved ROUGE-1 of 0.375, ROUGE-L of 0.351, and BERTScore of 0.397. To the best of our knowledge, we are the first to address this research problem, there is no existing baseline available for direct comparison. Therefore, we chose to compare CEDCC with a large language model (LLM), which has shown excellent performance in natural language understanding tasks, and ALLenNLP, which is well-suited for similar tasks, such as coreference resolution. In both the detection and completion stages, CEDCC significantly outperformed these two baselines. We also conducted ablation experiments to investigate the contribution of various design decisions in CEDCC, and the results demonstrated that all these decisions positively contribute to CEDCC's performance. Additionally, we performed a human evaluation of CEDCC's performance, which showed that CEDCC effectively improves the comprehensibility and utility of API knowledge sentences.

In summary, the contributions of this paper are:

- We conducted an empirical study of 384 SO posts and identified two types of context-dependent entities in API knowledge sentences, which hinder developers' understanding of API knowledge within these sentences.
- We developed a novel method, CEDCC, that can detect context-dependent entities in API knowledge sentences and complete the necessary context. This method can be used independently or alongside existing API knowledge extraction techniques to enhance the comprehensibility of the extracted API knowledge.
- We constructed a specialized dataset consisting of 1,023 API knowledge sentences and 574 context-dependent entities, which can be used to evaluate both the detection of context-dependent entities and the completion of the context that assists in understanding these entities.

II. Empirical Study

To understand the challenges and potential solution space for detecting context-dependent entities in API knowledge sentences within Stack Overflow posts and completing their required context, we conducted an empirical study to explore the characteristics of such entities, focusing on answering the following research questions:

- RQ1: How prevalent are context-dependent entities in API functionality sentences?
- RQ2: What types of context-dependent entities are in API functional sentences?
- RQ3: What is the distribution of the context on which the entity depends?

A. Data Collection

First, we download the official data released by SO as of December 2023[1]. Then we selected the posts related to five popular Python libraries (i.e., NumPy, Pandas, Matplotlib, SciPy, and Scikit-learn) as study objects due to the widespread use of these libraries in the developer community and their rich APIs. There are a significant amount of discussions related to these libraries on SO, providing a rich dataset for studying context dependency in API sentences. Additionally, focusing on these libraries allowed us to conduct a thorough analysis within a manageable scope, ensuring the reliability of our findings.

To ensure the reliability of the extracted API sentences, developers generally extract them from the accepted answers

[1]https://archive.org/details/stackexchange

in SO [12], [13] (i.e., the answer that the questioner thinks can solve his/her problem). We followed their practices by selecting accepted answers as our knowledge source. That is, we collected all accepted answers from SO that were tagged with at least one of the five Python libraries mentioned above, and obtained 290,316 accepted answers.

Second, due to the large volume of answers, conducting a manual analysis of each answer would be impractical. Therefore, we employed a sampling method to ensure the feasibility of the analysis and the representativeness of the results. Specifically, we randomly sample 384 answers from these answers with 95% confidence level and 5% confidence interval [14]. We further preprocessed these answers from HTML format to clean text using BeautifulSoup[2]. Considering that we need to annotate context-dependent entities at sentence-level, we split these 384 answers into 1,981 sentences with the NLTK sentence parser [15].

B. RQ1

Approach. To answer RQ1, we employed two master's students proficient in Python to manually classify 1,981 sentences from the dataset, determining whether each sentence describes API functionality. In cases where their classifications diverged, one of the authors facilitated a discussion with them to reach a final decision. After extracting the API function sentences, they need to determine whether these sentences contain entities that require context outside the sentences to assist in understanding, and then annotate the entities.

Results. As a result, we identified 373 API functionality sentences, with a Cohen's Kappa coefficient [16] of 0.81. Among these, 112 sentences contained 141 context-dependent entities, with a Cohen's Kappa coefficient of 0.84. In other words, approximately one-third of the sentences describing API functionality require additional context beyond the sentence itself for full understanding. This finding underscores the necessity of detecting context-dependent entities and completing necessary context in API functionality sentences.

C. RQ2

Approach. To answer RQ2, these two masters and one author proceeded to classify the context-dependent entities identified in RQ1. The classification of context-dependent entities was conducted as an iterative process by following steps similar to the process of Liu *et al.* [17] and Snow *et al.* [18]. Specifically, we first randomly sampled 10 sentences from a set of 112 sentences identified in RQ1 as containing context-dependent entities for an exploratory annotation. Through independent annotations and subsequent discussions among annotators, we established an initial category termed "Referential Phrases", which refers to phrases containing demonstrative pronouns, such as "this data." Following this, we proceeded to annotate the remaining sentences in the dataset. if an entity was identified as context-dependent but did not match any of the existing defined types, we modified the definitions of current context-dependent entity types or created a new type based on discussions. If any changes to the context-dependent entity types occurred, all sentences were re-annotated accordingly. This iterative approach ensured comprehensive classification.

It is important to note that during the annotation process, certain entities were identified as domain-specific terms, such as "Mersenne-Twister PRNG" in the sentence "Random numbers are created by the Mersenne-Twister PRNG in numpy.random." Since our focus is on entities that require additional context beyond the API sentence for understanding, and understanding domain-specific terms depends on the annotators' expertise in the respective domains, we chose not to classify these entities as context-dependent, despite their potential to hinder users' comprehension of the API knowledge sentences.

To further validate the correctness and completeness of our coding of context-dependent entity types and to minimize bias, we enlisted two master's students proficient in Python (who had not participated in the previous annotation) to annotate a subset of the above API functionality sentences following our coding protocol. Specifically, we randomly extracted 20 API sentences from 373 API functionality sentences and the annotation was conducted independently by the two students. For each sentence, they annotated the context-dependent entities based on our definitions. If none of the existing type definitions were suitable, they labeled the sentence as "New Type." The Cohen's Kappa coefficient [16] for this annotation process was 1. The annotation results in this round were consistent with our previous annotations, and no new types were reported.

Results. Through the open coding process described above, we identified two types of context-dependent entities: Referential Context-Dependent Entities (**RCDE**) and Local Variable Context-Dependent Entities (**LVCDE**). Their definitions are as follows:

- RCDE: An entity is classified as RCDE if it includes referencing terms, such as demonstrative pronouns (e.g., "this") or spatial references (e.g., "above"), where the explanation of the referenced entity requires context beyond the API sentence itself.
- LVCDE: An entity is classified as LVCDE if it is defined within a code snippet or data snippet in SO, and its explanation requires context beyond the API sentence itself.

Table I presents the examples and the number of occurrences of the two identified types of context-dependent entities. In the examples, the context-dependent entities are highlighted in bold. Among these, RCDE is the most common type of context-dependent entity.

D. RQ3

Approach. To answer RQ3, we first need to collect the SO posts to which each context-related entity belongs. This includes the accepted answer containing the entity, the corresponding question, other answers to the same question, and their comments. The two masters who were responsible for

[2]https://pypi.org/project/beautifulsoup4/

TABLE I
THE DETAILS OF CONTEXT-DEPENDENT ENTITY TYPES

Type	Example	Count
RCDE	you can convert **it** to a dict using pandas.DataFrame.to_dict method	114
LVCDE	use numpy.setdiff1d to find all the rows of **df_a** that not in the inner join	27

TABLE II
DISTRIBUTION OF CONTEXTS ON WHICH ENTITIES DEPEND

Entity Type	Question	Accepted Answer	Other Answer
RCDE	69	44	1
LVCDE	7	20	0

annotation in RQ1 and RQ2 continued their involvement by reading the SO posts associated with each context-dependent entity to identify the specific context required for understanding these entities. For each context-dependent entity, they need to provide the location in the SO post of the context required to understand that entity. It should be noted that when the required context appears in a comment, its location is recorded as the corresponding question or answer to which the comment belongs. If the required context appears in multiple locations, the first occurrence sorted by time is used as the reference location. In cases where discrepancies arose between the two annotators, one of the authors coordinated a discussion to resolve the conflicts.

Results. Table II presents the distribution of the required contexts for each type of context-dependent entity. We observe that, regardless of the type of context-dependent entity, the context required for understanding may be distributed across various parts of the SO posts (e.g., questions, accepted answer and other answers) without being constrained by the type of the entity. This observation suggests that, when designing methods for completing context, it is crucial to consider all parts of the SO posts comprehensively.

III. OUR PROPOSED APPROACH

The overall framework of our approach is shown in Fig. 2. Generally, `CEDCC` consists of three stages (i.e., candidate entity selection, coreference chain construction, step-by-step context completion). The first two stages are to detect the context-dependent entities in the sentence, and the last stage is to complete the required context for understanding the detected context-dependent entities.

A. Stage 1: Candidate entity selection

Based on the characteristics of the two context-dependent entities discovered in Section II, we designed corresponding screening strategies to select candidate entities.

Extraction of candidate RCDE. For the extraction of candidate RCDE, we utilized spaCy, a library that performs well in natural language process tasks [19]. First, we extracted all noun phrases from the API sentences using spaCy's syntactic parsing features. This is because in API sentences, the context-dependent entities are generally nouns. Then, we filter the extracted noun phrases according to a predefined keyword list, and select a noun phrase as a candidate RCDE if it contains any of the keywords in the list. This keyword list was meticulously designed to capture phrases that refer to other entities. The details of the list is provided in Table III. The construction of this keyword list is based on the results of empirical study and linguistics knowledge [20].

TABLE III
CEDCC CANDIDATE ENTITY SELECTION KEYWORD LIST

Type	Keywords
Personal Pronouns	he, she, it, they, him, her, them
Demonstrative Pronouns	this, that, these, those
Possessive Pronouns	my, your, his, her, its, our, their, mine, yours, hers, ours, theirs
Interrogative Pronouns	what
Indefinite Pronouns	such
Spatial Reference	above, below

Extraction of candidate LVCDE. According to our empirical study results, LVCDE are local variables defined in code snippets or data snippets in SO posts, as shown in Fig. 3 (*df* is a local variable defined in the code, and *Sell* is displayed in the data). So we should select candidate LVCDE from these local variables. Considering that code and data snippets are rich sources of local variable definitions and assignments and they are often surrounded by $< Code >$ tags in SO, we first extract the snippets marked by the $< Code >$ tag in the SO posts where the given API sentence is located.

After extracting them, we further differentiated them and designed corresponding processing strategies for each type of snippet to extract local variables. Specifically, first we use Tree-sitter[3], a powerful parsing library capable of handling complex programming language grammars to parse each line of the snippet. If a line was successfully parsed, it was classified as a code snippet. For lines that the Tree-sitter could not parse, we conducted a heuristic analysis to detect typical code characteristics. We searched for programming constructs such as keywords (`for`, `while`, `return`), the assignment operator (=), and other syntax elements indicative of code. Lines meeting these criteria were also classified as code snippets. Finally, lines that did not meet the code criteria were classified as data snippets.

After distinguishing code snippets from data snippets, we further process them as follows:

- From the identified code snippets, we extracted code identifiers, which include variable names and function names. Given that programming language keywords and built-in functions do not require context to assist in understanding, we excluded these from our identifier list (In this paper, we excluded Python keywords, built-in functions, and the APIs of the five Python libraries (i.e.,

[3]https://github.com/tree-sitter/tree-sitter

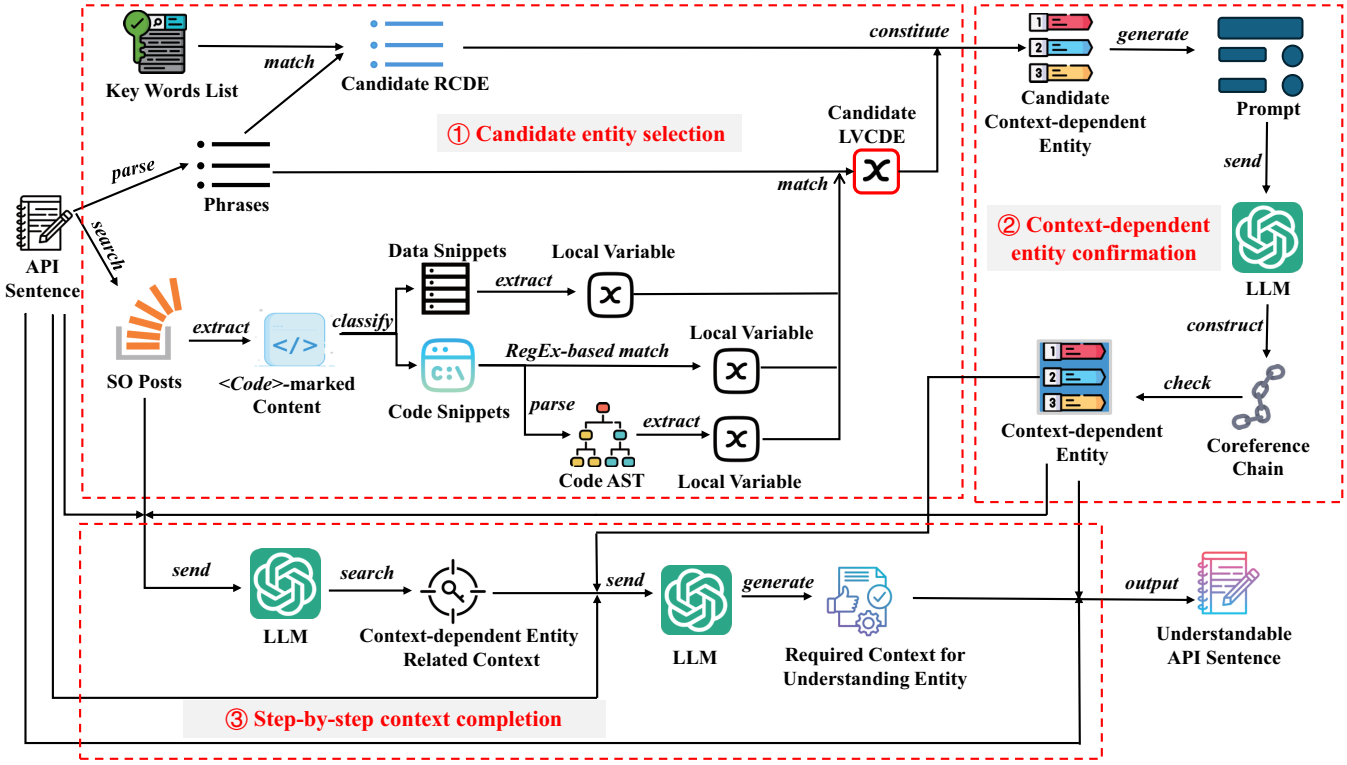

Fig. 2. Framework of Our Approach.

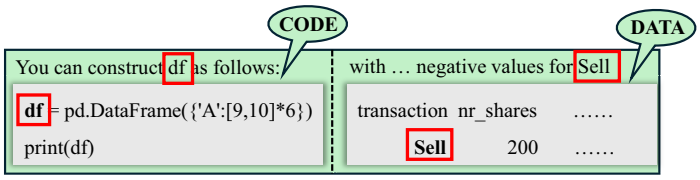

Fig. 3. Example of Two Types of Local Variables.

the libraries as research subjects in Section II), including functions, methods, and classes. These exclusions were determined based on the official API documentation). This refinement focuses our analysis on user-defined identifiers that are more likely to be context-dependent.

- From the identified data snippets, We first separate them by spaces and newlines to get all the row names, column names, and values, then filter out pure numbers and dates, and finally remove duplicates to get the local variables of the data snippets.

Through the above steps, we obtain the list of local variables defined in the SO posts where the API sentence is located. Then we select a noun phrase extracted from the API sentence as a candidate LVCDE if it appears in the local variables list.

B. Stage 2: Coreference chain construction

After obtaining the candidate context-dependent entities, we need to further determine whether these candidate entities are truly context-dependent, that is, whether understanding the entity requires the help of context outside the sentence. We can consider this task as a variant of the coreference resolution. Coreference resolution is the task of identifying and linking pronouns or noun phrases to their respective antecedents within a text [21]. In our scenario, we need to link candidate entities to their all referents within the API sentence. Based on this step, if the referent of a candidate entity is either itself or consists solely of other candidate entities, we can conclude that the entity is context-dependent.

To achieve this, we first need to construct coreference chains within the API sentence. Coreference chains a sequence of phrases in a text that refer to the same entity [11]. Some API libraries provide coreference resolution models, such as AllenNLP [4], which are used to construct coreference chains in a given text. However, some context-dependent entities refer to concrete objects (e.g., "*it*" in the example of the bottom of Fig. 4), while others refer to an abstract process, e.g., "*this*" in the example of the top of Fig. 4, which lack a concrete object. This presents a challenge for existing coreference resolution models, which often struggle to handle the latter references [22], [23]. While LLM's strong information extraction capability enables effective processing of abstract references [24], [25], making it an ideal choice for constructing coreference chains.

To reduce the complexity of inferring coreference chains and allow LLMs to focus on constructing coreference chains involving candidate entities, we designed the prompt shown in

[4]https://github.com/allenai/allennlp

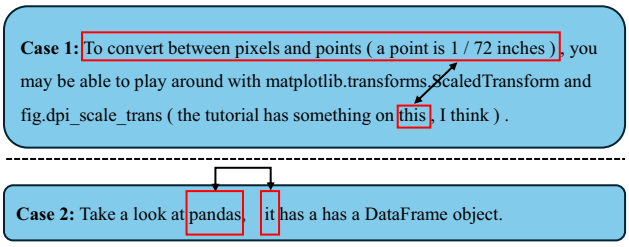

Fig. 4. The example for two types of reference.

Figure 5. Specifically, we marked context-dependent candidate entities in the sentence using special symbols, facilitating their identification by the LLM in subsequent analyses. We then instructed the LLM to generate coreference chains only for these marked entities, rather than inferring coreference for all entities within the API knowledge sentence.

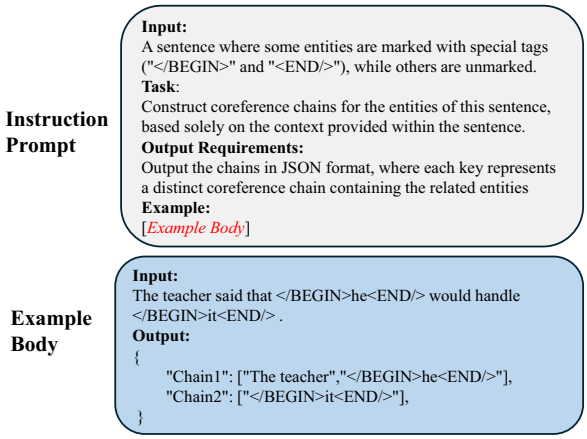

Fig. 5. The prompt for constructing coreference chain.

After generating the coreference chains, we analyze each chain by traversing through it. Specifically, if all entities in a coreference chain are candidate entities, we can conclude that understanding them requires context beyond the sentence, classifying them as context-dependent entities (e.g., "it" in Chain2 in Fig. 5). Conversely, if the coreference chain includes non-candidate entities, the candidate entities in that chain are not considered context-dependent (such as "he" in Chain1 in Fig. 5).

C. Stage 3: Step-by-step context completion

In this stage, we adopt a step-by-step strategy to instruct LLM to complete the required context for understanding context-dependent entities. This is motivated by that compared to directly utilizing LLM to perform inference tasks, breaking the tasks into multi-sub-tasks can guide the model step-by-step, ensuring a thorough understanding of the context while reducing ambiguity in understanding complex semantic relationships and improving inference quality [24], [26], [27].

As shown in the third stage of Fig. 2, the context completion task is divided into two subtasks. For the first subtask, we input the API sentences, the detected context-dependent entities, and the candidate context needed to assist in understanding these entities into the LLM. (the candidate context is defined as the SO thread to which the API sentence belongs that includes the question, all answers, and comments, a scope determined by the findings of RQ3). The LLM is then instructed to extract from this candidate context the parts relevant to the detected context-dependent entities (e.g., the sections highlighted by red boxes in Fig. 6). For the second subtask, the outputs from the first subtask, along with the API sentences and the identified context-dependent entities, are fed into the LLM. The LLM is tasked with generating the necessary context that facilitates better comprehension of the context-dependent entities (e.g., the natural language explanation of "your features" shown in Fig. 6).

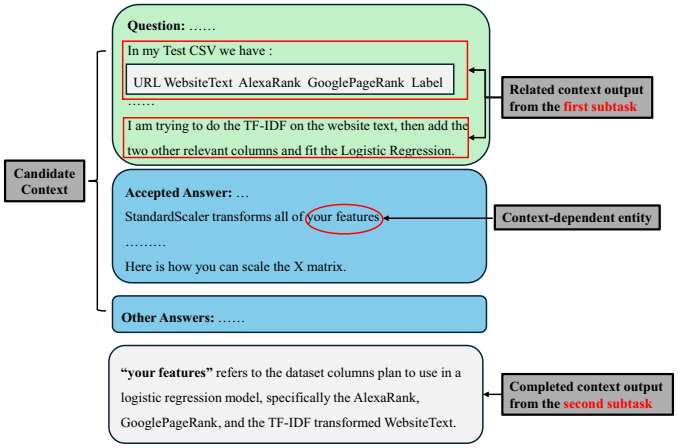

Fig. 6. An example: completing the context required for understanding entity "your features".

IV. Experiment Settings

In the experiment, we want to answer the following three research questions (RQs):

- RQ4. How well does our approach perform in detecting context-dependent entities and completing the required context?
- RQ5. What is the contribution of different components to the performance of our approach?
- RQ6. Can our approach improve developers' understanding of API sentences?

In RQ4 and RQ5, we evaluate the overall performance and the contribution of each component. In RQ6, we verify whether the API sentences processed by our approach can improve developers' understanding of API sentences.

A. Datasets

We expanded the dataset constructed in the empirical study by conducting additional sampling. Specifically, we randomly sampled 616 additional accepted answers from the 290,316 responses obtained in the empirical study, combining them with the 384 answers previously analyzed. This increased the

total number of answers in the dataset to 1,000, ensuring a 99% confidence level with a 5% confidence interval. Following the annotation methodology from the empirical study, we identified API functionality sentences and annotated context-dependent entities, yielding Cohen's Kappa coefficients of 0.87 and 0.84, respectively.

Subsequently, we collected each question corresponding to the accepted answers, along with other answers to these questions and their comments. Specifically, we gathered 1,000 questions, 1000 accepted answers, 966 other answers, and 4,703 comments. We analyzed these SO posts to identify and complete the necessary context for understanding the context-dependent entities, employing an iterative process of independent annotation followed by discussions to resolve discrepancies. The details of final dataset are presented in Table IV.

TABLE IV
THE STATISTICS OF DATASET

Sentences	Tokens	Context-Dependent Entities	
		RCDE	LVCDE
1023	16746	498	76

It is important to note that the detection task is evaluated at the entity level, meaning the number of test samples corresponds to the total number of entities, not just the context-dependent ones. This results in approximately 16,000 entities for evaluation. Based on the principle that a sample size of around 384 is sufficient to achieve a 95% confidence level with a 5% confidence interval as the total sample size increases [14], evaluating 574 context-dependent entities for the context completion task provides an effective reflection of `CEDCC`'s performance on a larger-scale test set.

B. Performance Measures

In our study, we aim to show the competitiveness of `CEDCC` via both automatic evaluation and human evaluation.

Automatic Evaluation. The detection of context-dependent entities can be considered a named entity recognition (NER) task. Therefore, we adopt commonly used evaluation metrics in NER tasks, including precision, recall, and F1 score [28]. Precision measures the proportion of predicted context-dependent entities that are correctly identified, while recall measures the proportion of true context-dependent entities that are correctly recognized. The F1 score is the harmonic mean of precision and recall, providing a balanced evaluation of the detection performance.

Given that the expressions of context necessary for understanding the entities can vary, the context completion task can be framed as a generative task. Accordingly, we employ two commonly used evaluation metrics for this task: ROUGE [29] and BERTScore [30]. Specifically, we use ROUGE-1, which measures unigram overlap, and ROUGE-L, which captures the longest common subsequence between the generated and reference texts. These metrics assess the quality of the generated context completions in terms of both lexical similarity and sequence structure. Additionally, we use BERTScore to provide a semantic-level evaluation by comparing the contextual embeddings of the generated and reference context completions, thereby evaluating how well the meaning is preserved.

Human Evaluation. Our human evaluation primarily follows the methodology of Lin et al. [31]. Specifically, we employed six Ph.D. students, none of whom are co-authors of this paper. Each participant has over five years of programming experience and is familiar with Python API libraries. To avoid bias, where participants may have prior assumptions about the method used to generate API sentences that could influence their evaluation, the processing method used for each API sentence was anonymized in the questionnaire, and each participant completed the questionnaire independently. Before conducting the evaluation, we provided detailed guidelines on detecting context-dependent entities and completing their required contexts. Each participant was asked to score each API sentence on the following four aspects:

- **Accuracy**, which reflects how correctly the completed context explains the context-dependent entities.
- **Independence**, which measures the extent to which the processed API sentences can be understood without requiring additional context.
- **Comprehensibility**, which evaluates the usefulness of the processed API sentences in aiding software development.
- **Practicality**, which evaluates the usefulness of the processed API sentences in aiding software development.

All scores are given on a scale from 1 to 5 (1 for poor, 2 for marginal, 3 for acceptable, 4 for good, and 5 for excellent).

C. Baselines

In conducting the evaluation, we faced the challenge of selecting an appropriate baseline, as this study is the first to address the detection of context-dependent entities and the completion of their context in API sentences. To reasonably evaluate `CEDCC`, we selected two baseline methods for comparison. First, we chose the coreference resolution model provided by the AllenNLP library (hereafter referred to as `AllenNLP-baseline`) as a baseline. Although this model is not specifically designed for our task, coreference resolution shares a similar objective. Additionally, we developed a simplified version of `CEDCC` as another baseline. Specifically, we utilized LLM to directly detect context-dependent entities within a given sentence and complete the required context (hereafter referred to as `LLM-baseline`).

D. Implementation Details

All experiments were conducted on a computer equipped with an Intel(R) Core(TM) i9-10900X CPU and a GeForce RTX 4090 GPU with 24 GB of memory, running Ubuntu 20.04 as the operating system. The version of the large language model (LLM) used is *gpt-4-0613*, with a temperature setting of 0.5. For SpaCy, we used version 3.7.3, with the

en_core_web_sm model as the NLP parser. For the code parser Tree-sitter, we used version 0.23.2. For the baselines, we used AllenNLP version 2.10.0, with the coreference resolution model *coref-spanbert-large-2021.03.10*.

V. Results analysis

A. RQ4

The comparison results between CEDCC and the baselines are presented in Table V. Notably, we divided the evaluation of context completion into two scopes: (1) the evaluation of the completion for all context-dependent entities, referred to as *Scope 1*, where an empty string was used as the completion result if a context-dependent entity was not correctly identified; and (2) the evaluation of the completion for only those context-dependent entities that were correctly identified, referred to as *Scope 2*. The best value in each column is highlighted in bold.

In the phase of detecting context-dependent entities, CEDCC demonstrated superior performance across precision, recall, and F1 score, achieving values of 0.912, 0.814, and 0.860, respectively. These results indicate that the CEDCC approach not only accurately identifies entities (high precision) but also comprehensively captures them (high recall). In contrast, the LLM-baseline showed a low precision of 0.164, resulting in a correspondingly low F1 score. This suggests that effective identification of context-dependent entities requires more sophisticated rules or analytical processes, such as constructing coreference chains, rather than solely relying on model predictions.

The AllenNLP-baseline achieved a recall of 0.843, demonstrating its ability to identify a greater number of context-dependent entities. However, its precision was only 0.167, reflecting a high false positive rate and leading to an F1 score of just 0.278. This issue may be attributed to the absence of candidate phrase selection and its limited capacity to handle abstract references. For example, in Case 1 (shown in Fig. 4), the entity "this" refers to the action "To convert between pixels ..." rather than a specific entity. AllenNLP-baseline failed to recognize this abstract reference relationship and incorrectly classified "this" as a context-dependent entity.

In both evaluation scopes of the context completion task, CEDCC significantly outperformed the two baseline methods in both semantic-based (BERTScore) and lexical-based (ROUGE) evaluations. While the performance of the LLM-baseline improved in *Scope 2*, it still fell short of CEDCC. Since both methods utilized LLM for context completion, this suggests that a step-by-step approach to context completion improves both lexical and semantic performance compared to a direct approach. The AllenNLP-baseline performed poorly in both scopes, likely due to its relatively weaker ability to handle long texts and abstract references compared to LLM.

Results for RQ4: CEDCC significantly outperformed both baselines in detecting context-dependent entities and completing context via automatic evaluation.

B. RQ5

Approach. In this RQ, we investigate the contribution of different components of CEDCC to its overall performance through an ablation study. Specifically, we design several variants of CEDCC by altering each component individually while keeping the others unchanged, as follows:

- Regex-based Entity Selection (**RES**): Uses regular expressions to extract candidate entities by selecting noun phrases that include pronouns or exhibit morphological characteristics typical of code naming conventions (e.g., CamelCase).
- Without Coreference Chain (**WCC**): Directly prompts the LLM to determine whether the candidate entity is context-dependent, without constructing or analyzing coreference chains.
- Targetless Coreference Chain (**TCC**): Constructs coreference chains without specifying target entities, then filters for chains that contain the target entities after construction.
- Direct Context Completion (**DCC**): Prompts the model to complete the context directly, rather than following a step-by-step strategy.

Results. Table VI presents the performance of CEDCC and its variants. It is clear that CEDCC achieves the best performance across all metrics, validating the contribution of each design component to the overall system's effectiveness. In the context-dependent entity detection stage, the low performance of **RES** suggests that entities identified through syntactic and code analysis are more accurate than those obtained via regular expression matching. This, in turn, impacts the quality of coreference chains constructed by the LLM. The discrepancy likely stems from the fact that regular expressions capture only morphological features of entities (e.g., CamelCase), without accounting for contextual relationships such as grammatical dependencies.

When comparing methods that use the same candidate entity selection process but differ in their approach to determining whether an entity is context-dependent, **WCC** shows the worst performance. This may be because identifying context dependence requires explicit rules, such as coreference chain construction and analysis, rather than relying solely on model predictions. **TCC** also underperforms compared to CEDCC, likely due to CEDCC's pre-selection of candidate entities, which enables the LLM to focus on specific entities during coreference chain construction. By clearly defining target entities, CEDCC reduces interference from unrelated entities, resulting in higher-quality coreference chains.

Regarding different strategies for completing context, CEDCC's step-by-step approach, which first narrows the context the LLM needs to analyze, allows the model to focus more effectively on completing the essential contextual details in the second step. In contrast, **DCC**'s direct context completion approach requires the LLM to handle a broader scope of information, increasing the likelihood of being distracted by irrelevant details, thereby reducing the overall quality and

TABLE V
OUR METHOD VS BASELINE

Method	Predict			Complete in Scope 1			Complete in Scope 2		
	P	R	F1	ROUGE-1	ROUGE-L	BERTScore	ROUGE-1	ROUGE-L	BERTScore
CEDCC	**0.912**	0.814	**0.860**	**0.375**	**0.351**	**0.397**	**0.404**	**0.378**	**0.428**
LLM-baseline	0.164	0.564	0.254	0.159	0.149	0.149	0.307	0.288	0.287
AllenNLP-baseline	0.167	0.843	0.278	0.076	0.076	0.053	0.200	0.198	0.139

TABLE VI
ABLATION STUDY

Method	Predict			Complete in Scenatio 1			Complete in Scenatio 2		
	P	R	F1	ROUGE-1	ROUGE-L	BERTScore	ROUGE-1	ROUGE-L	BERTScore
CEDCC	**0.912**	**0.814**	**0.860**	**0.375**	**0.351**	**0.397**	**0.404**	**0.378**	**0.428**
RES	0.438	0.650	0.523	0.210	0.189	0.207	0.327	0.294	0.322
WCC	0.491	0.193	0.277	0.054	0.048	0.059	0.272	0.239	0.295
TCC	0.890	0.807	0.846	0.238	0.207	0.238	0.290	0.252	0.289
DCC	0.912	0.814	0.860	0.274	0.257	0.266	0.330	0.310	0.321

performance of the completion process. This step-by-step strategy is a key factor contributing to CEDCC's superior performance over **DCC**.

Results for RQ5: Each component of CEDCC plays a crucial role in its overall performance.

C. RQ6

Approach. Following the sample sizes used in similar generation tasks involving manual evaluation [32], we randomly selected 50 API sentences containing context-dependent entities from the dataset. These sentences were processed using Allennlp-baseline, LLM-baseline, and CEDCC, along with the original API sentences, to form the questionnaire. Since the original API sentences lack completed context for the context-dependent entities, *Accuracy* was not evaluated for them. Each participant was required to evaluate 150 sentences for the *Accuracy* dimension and 200 API sentences for each of the other three dimensions. After collecting their scores, we used Fleiss' Kappa [33] to measure inter-rater agreement. The Kappa values for *Independence*, *Practicality*, *Comprehensibility*, and *Accuracy* were 0.746, 0.782, 0.764, and 0.8, respectively, indicating substantial agreement among participants.

Results. The results are presented in Table VII. CEDCC outperforms the baseline methods across all dimensions, demonstrating its effectiveness in identifying context-dependent entities in API sentences and providing the necessary contextual information to enhance understanding. This leads to significant improvements in the comprehensibility and practicality of API sentences. In terms of *Independence* and *Accuracy*, LLM-baseline performs relatively well, though slightly below CEDCC, while AllenNLP-baseline performs the worst. This observation aligns with our conclusions from RQ1, which evaluated the detection and completion phases using automated metrics. The original sentences received the lowest scores for *Independence* (2.0), which is consistent with the fact that the selected API sentences inherently contain context-dependent entities.

Furthermore, we observe a general trend that methods scoring highly in *Independence* and *Accuracy* also tend to score well in *Practicality* and *Comprehensibility*. This suggests that improvements in *Independence* and *Accuracy* positively impact *Practicality* and *Comprehensibility*. These findings validate the motivation behind our research, which is to enhance developers' understanding and application of API knowledge by identifying context-dependent entities in API sentences and completing the necessary context.

Case Analysis. Fig. 7 presents an example[5] of identifying and completing context-dependent entities in API sentences using CEDCC compared to LLM-baseline and AllenNLP-baseline. CEDCC produces results closely aligned with the ground truth in both vocabulary and semantics, achieving high scores for *Independence* and *Accuracy* (5 and 4.6, respectively). It accurately describes the data type of *page_id* as a column in a DataFrame (i.e., a Series) and provides additional explanation of the *pandas.factorize* API. This clarity contributes to higher scores in *Practicality* and *Comprehensibility* (4 and 4, respectively). It is worth noting that one participant gave a *Practicality* score of 2 (i.e., marginal) because this was the development knowledge he had already mastered.

In contrast, AllenNLP-baseline correctly identifies *page_id* as a context-dependent entity but fails to provide a meaningful completion, leading to low scores for *Accuracy* and *Practicality* (1.3 and 1.8, respectively). This oc-

[5]https://stackoverflow.com/questions/15829494

TABLE VII
THE RESULTS OF OUR HUMAN EVALUATION.

Method	Independence	Practicality	Comprehensibility	Accuracy
Original Sentence	2.0	2.4	2.6	-
LLM-baseline	3.5	4.0	3.8	4.0
AllenNlp-baseline	2.4	3.0	3.3	1.4
CEDCC	**4.0**	**4.4**	**4.1**	**4.6**

curs because `page_id`'s reference needs to be extracted from the data snippet within the context, a task that `AllenNLP-baseline`'s coreference resolution model is unable to perform, resulting in an empty completion.

The `LLM-baseline` incorrectly identifies *array* as a context-dependent entity, even though understanding *array* requires Python development knowledge rather than context within the SO post. Additionally, `LLM-baseline` fails to provide a clear specification of the data structure for *page_id*. We speculate that this is due to its direct context completion strategy. In contrast, `CEDCC` provides this detail through a step-by-step completion process. First, `CEDCC` locates the relevant context, specifically identifying the data snippet that defines the local variable *page_id*. In the second step, this relevant context is highlighted in the prompt provided to the LLM. This step-by-step process effectively narrows the model's search space, allowing it to focus more precisely on relevant information. In comparison, `LLM-baseline` lacks this step, resulting in unclear descriptions of the data structure and consequently lower scores in *Accuracy* and *Practicality* compared to `CEDCC` (4.2 and 3.5, respectively).

Original API sentence:
……Here I use pandas.factorize() to convert the page_id to an array ……

Related Context:
I have a pandas DataFrame …… which is structured like so:

page_id	score
2430	7.632
……	……

Ground truth:
- ***page_id***: a column in the DataFrame, representing the unique identifier of a page.

Allennlp-baseline:
- ***page_id***:

LLM-baseline:
- ***array***: a data structure that stores a collection of items
- ***page_id*** : an identifier for a specific page within the document

CEDCC:
- ***page_id***: a column in the DataFrame that serves as a unique identifier for each page, used to track individual pages across different scores.

Fig. 7. An Example of Detecting Context-dependent Entities and Completing Required Context in API Knowledge Sentence.

Results for RQ6: Through manual evaluation, it was found that `CEDCC` enhances the comprehensibility of API sentences.

VI. RELATED WORK

A. API Knowledge Extraction from SO

API documentation is crucial in software development, but its complexity and diversity often lead to hard-to-understand content. To tackle this, extensive research has focused on extracting various types of API knowledge from community platforms like SO to assist in understanding [17], [34]. Treude *et al.* [1] use a machine learning-based method to extract insightful sentences from SO to augment API documentation. Nam *et al.* [8] introduce SOREL, a machine learning tool that extracts comparative knowledge of APIs scattered in posts, improving developers' understanding by highlighting differences between comparable APIs. Ren *et al.* [5] used SO to supplement the constraints knowledge of APIs. They developed a text mining approach that extracts API misuse scenarios from SO to create demystification reports. As for API functionality knowledge, Shen *et al.* [35] perform syntactic analysis on SO posts to obtain verb-object phrases, thereby obtaining descriptions for API functionality.

While these extraction methods effectively retrieve relevant API knowledge, they can unintentionally lose context from the original Stack Overflow posts, making certain entities harder to understand and reducing the usefulness of the extracted knowledge. Our study addresses this issue by detecting and completing the missing context.

B. LLM for software engineering

LLMs, due to their powerful natural language and programming capabilities, have been widely applied across various aspects of software engineering [36]–[39]. Several researchers have focused on leveraging LLMs for code-related tasks. For example, Chen *et al.* introduced ChatUniTest [40], an LLM-based framework for automated unit test generation through an adaptive focal context mechanism and a generation-validation-repair process. Kang *et al.* [41] presented AutoFL, an LLM-based fault localization technique that enhances method-level accuracy by up to 233.3% on real-world Java and Python bugs, providing both fault locations and natural language explanations. InferFix [42], an LLM-powered program repair tool, uses a 12-billion-parameter Codex Cushman model fine-tuned on bug-fix data to generate precise fixes for critical security and performance issues, which enhances the LLM's ability to repair bugs.

Other researchers are exploring how LLMs' superior NLP capabilities can improve software-related documentation. Li *et al.* propose FSATD [43], a fusion approach that combines ChatGPT with smaller models for Self-Admitted Technical Debts detection so as to provide reliable explanations. Arora *et al.* [44] investigate how LLMs can improve requirements engineering by enhancing the efficiency and accuracy of tasks like elicitation and analysis of software requirements.

We focus on the application of LLM in assisting in API knowledge extraction. We do not use LLM to complete the tasks in one step, but decompose the tasks and then design targeted prompts in each subtask to use the LLM step by step.

VII. THREATS TO VALIDITY

Internal Threats. The first threat to internal validity is related to the subjective judgment of the annotators during data annotation. To alleviate this threat, we follow commonly used data analysis principles, e.g., multiple annotators, conflict resolution, and reporting agreement coefficients, where appropriate. Another threat pertains to the performance and potential failures of our baseline implementations. To mitigate this threat, we employed models with outstanding performance in their respective domains that were readily available for use. Specifically, we utilized the coreference resolution model *coref-spanbert-large-2021.03.10* from the AllenNLP library and LLM *gpt-4-0613*.

External Threats. The threat to external validity concerns the generalizability of our results and findings. To alleviate this threat, The number of test samples in our detection phase was approximately 16,000, while for the completion phase, the number of test samples was determined to maintain a 95% confidence level with a 5% confidence interval. In the future, we will build larger datasets covering more programming languages for further evaluation.

Construct Threats. The threat to construct validity comes from human studies, which may introduce bias. To ensure that all students can correctly understand our questionnaire, we provided a tutorial before our human study.

VIII. CONCLUSION

In this paper, we proposed CEDCC, a novel method to address the challenge of extracting API knowledge from Stack Overflow while preserving crucial context. Existing approaches that extract API-related knowledge at the sentence level often lose essential contextual information, which limits comprehension. Through an empirical study, we identified two types of context-dependent entities—referential phrases and local variables, and designed **CEDCC** to automatically identify and complete their required context. CEDCC significantly outperformed baseline methods in both context-dependent entity identification and context completion, achieving high precision, recall, and comprehensibility scores. Human evaluations confirmed CEDCC's effectiveness in improving API knowledge utility. Our work contributes to enhancing API documentation by preserving context, thus aiding developers in better understanding and utilizing APIs.

REFERENCES

[1] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 392–403.

[2] Q. Fan, Y. Yu, T. Wang, G. Yin, and H. Wang, "Why api documentation is insufficient for developers: an empirical study," *Science China. Information Sciences*, vol. 64, no. 1, p. 119102, 2021.

[3] G. Uddin, F. Khomh, and C. K. Roy, "Mining api usage scenarios from stack overflow," *Information and Software Technology*, vol. 122, p. 106277, 2020.

[4] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule, and M. Di Penta, "Focus: A recommender system for mining api function calls and usage patterns," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1050–1060.

[5] X. Ren, J. Sun, Z. Xing, X. Xia, and J. Sun, "Demystify official api usage directives with crowdsourced api misuse scenarios, erroneous code examples and patches," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 925–936.

[6] Z. Zhang, X. Mao, S. Wang, K. Yang, and Y. Lu, "Career: Context-aware api recognition with data augmentation for api knowledge extraction," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, 2024, pp. 438–449.

[7] K. Luong, M. Hadi, F. Thung, F. Fard, and D. Lo, "Arseek: identifying api resource using code and discussion on stack overflow," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, 2022, pp. 331–342.

[8] D. Nam, B. Myers, B. Vasilescu, and V. Hellendoorn, "Improving api knowledge discovery with ml: A case study of comparable api methods," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1890–1906.

[9] G. Uddin and F. Khomh, "Automatic mining of opinions expressed about apis in stack overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 522–559, 2019.

[10] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Classifying stack overflow posts on api issues," in *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2018, pp. 244–254.

[11] K. Lee, L. He, M. Lewis, and L. Zettlemoyer, "End-to-end neural coreference resolution," *arXiv preprint arXiv:1707.07045*, 2017.

[12] D. Wu, X.-Y. Jing, H. Zhang, Y. Feng, H. Chen, Y. Zhou, and B. Xu, "Retrieving api knowledge from tutorials and stack overflow based on natural language queries," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, pp. 1–36, 2023.

[13] D. Wu, X.-Y. Jing, H. Zhang, Y. Zhou, and B. Xu, "Leveraging stack overflow to detect relevant tutorial fragments of apis," *Empirical Software Engineering*, vol. 28, no. 1, p. 12, 2023.

[14] W. G. Cochran, "Sampling techniques," *John Wiley& Sons*, 1977.

[15] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[16] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[17] M. Liu, X. Peng, A. Marcus, S. Xing, C. Treude, and C. Zhao, "Api-related developer information needs in stack overflow," *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4485–4500, 2021.

[18] R. Snow, B. O'connor, D. Jurafsky, and A. Y. Ng, "Cheap and fast–but is it good? evaluating non-expert annotations for natural language tasks," in *Proceedings of the 2008 conference on empirical methods in natural language processing*, 2008, pp. 254–263.

[19] Y. Vasiliev, *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press, 2020.

[20] K. Chowdhary and K. Chowdhary, "Natural language processing," *Fundamentals of artificial intelligence*, pp. 603–649, 2020.

[21] R. Mitkov, *Anaphora resolution*. Routledge, 2014.

[22] R. Liu, R. Mao, A. T. Luu, and E. Cambria, "A brief survey on recent advances in coreference resolution," *Artificial Intelligence Review*, vol. 56, no. 12, pp. 14 439–14 481, 2023.

[23] N. Stylianou and I. Vlahavas, "A neural entity coreference resolution review," *Expert Systems with Applications*, vol. 168, p. 114466, 2021.

[24] X. Wei, X. Cui, N. Cheng, X. Wang, X. Zhang, S. Huang, P. Xie, J. Xu, Y. Chen, M. Zhang *et al.*, "Zero-shot information extraction via chatting with chatgpt," *arXiv preprint arXiv:2302.10205*, 2023.

[25] J. Zhao, N. Xue, and B. Min, "Cross-document event coreference resolution: Instruct humans or instruct gpt?" in *Proceedings of the 27th Conference on Computational Natural Language Learning (CoNLL)*, 2023, pp. 561–574.

[26] B. Li, G. Fang, Y. Yang, Q. Wang, W. Ye, W. Zhao, and S. Zhang, "Evaluating chatgpt's information extraction capabilities: An assessment of performance, explainability, calibration, and faithfulness," *arXiv preprint arXiv:2304.11633*, 2023.

[27] S. Ekin, "Prompt engineering for chatgpt: a quick guide to techniques, tips, and best practices," *Authorea Preprints*, 2023.

[28] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE transactions on knowledge and data engineering*, vol. 34, no. 1, pp. 50–70, 2020.

[29] C.-Y. Lin and F. Och, "Looking for a few good metrics: Rouge and its evaluation," in *Ntcir workshop*, 2004.

[30] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.

[31] B. Lin, S. Wang, Z. Liu, Y. Liu, X. Xia, and X. Mao, "Cct5: A code-change-oriented pre-trained model," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1509–1521.

[32] K. Liu, G. Yang, X. Chen, and C. Yu, "Sotitle: A transformer-based post title generation approach for stack overflow," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 577–588.

[33] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[34] C. Chen, Z. Xing, and Y. Liu, "What's spain's paris? mining analogical libraries from q&a discussions," *Empirical Software Engineering*, vol. 24, pp. 1155–1194, 2019.

[35] Q. Shen, Y. Qian, Y. Zou, S. Wu, and B. Xie, "Fusing code and documents to mine software functional features," *Journal of Software*, vol. 32, no. 4, pp. 1023–1038, 2021.

[36] R. Khojah, M. Mohamad, P. Leitner, and F. G. de Oliveira Neto, "Beyond code generation: An observational study of chatgpt usage in software engineering practice," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1819–1840, 2024.

[37] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[38] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," in *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*. IEEE, 2023, pp. 23–30.

[39] M. Watanabe, Y. Kashiwa, B. Lin, T. Hirao, K. Yamaguchi, and H. Iida, "On the use of chatgpt for code review: Do developers like reviews by chatgpt?" in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, 2024, pp. 375–380.

[40] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, "Chatunitest: A framework for llm-based test generation," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 2024, pp. 572–576.

[41] S. Kang, G. An, and S. Yoo, "A quantitative and qualitative evaluation of llm-based explainable fault localization," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1424–1446, 2024.

[42] M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, and A. Svyatkovskiy, "Inferfix: End-to-end program repair with llms," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1646–1656.

[43] J. Li, L. Li, J. Liu, X. Yu, X. Liu, and J. W. Keung, "Large language model chatgpt versus small deep learning models for self-admitted technical debt detection: Why not together?" *Software: Practice and Experience*.

[44] C. Arora, J. Grundy, and M. Abdelrazek, "Advancing requirements engineering through generative ai: Assessing the role of llms," in *Generative AI for Effective Software Development*. Springer, 2024, pp. 129–148.