ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO BÀI TẬP LỚN

ĐỀ XUẤT PHƯƠNG THỨC API QUA SO SÁNH CỤM ĐỘNG TỪ CHỨC NĂNG

Giảng viên hướng dẫn	:	**ThS. Nguyễn Thu Trang**
Lớp học phần	:	**INT3110 1**
Môn học	:	**Các vấn đề hiện đại công nghệ thông tin**
Sinh viên thực hiện	:	**Nhóm 1**
		Nguyễn Ninh Chi – 18020220
		Trịnh Thị Kim – 18020740
		Đào Minh Hoàn – 18020535
		Trần Thị Nhung – 18020983

HÀ NỘI - 2021

MỤC LỤC

1.Giới thiệu

Việc tìm kiếm các API phù hợp cung cấp các chức năng mong muốn là điều cần thiết trong nhiều nhiệm vụ phát triển phần mềm. Các thư viện API phổ biến như JDK và Android cung cấp tài liệu tham khảo bao gồm các mô tả chức năng cho các lớp và phương thức API. Tuy nhiên, do khoảng cách từ vựng giữa mô tả chức năng do nhà phát triển API cung cấp và truy vấn tìm kiếm của người dùng API, việc truy xuất API dựa trên tài liệu thường tạo ra kết quả kém.

Các nhà nghiên cứu [27, 44] đã cố gắng sử dụng kỹ thuật nhúng từ để thu hẹp khoảng cách từ vựng bằng cách tìm hiểu mức độ liên quan thống kê giữa các từ, chẳng hạn như “convert” và “transform”, “image” và “color”, “JSON” và “XML”. Tuy nhiên, các phương pháp này không phân tích cú pháp rõ ràng cấu trúc chi tiết của mô tả chức năng và truy vấn người dùng, chúng cũng không khớp rõ ràng với vai trò ngữ nghĩa của các phần khác nhau của mô tả chức năng và truy vấn người dùng. Do đó, các phương pháp này có thể dẫn đến kết quả đối sánh kém khi phải tính đến các chi tiết ngôn ngữ chi tiết về mô tả chức năng và truy vấn của người dùng để tạo ra đối sánh thỏa đáng. Ví dụ: các phương pháp hiện có không thể phân biệt “convert Integer to String” với “convert String to Integer”, bởi vì chúng không hiểu vai trò gốc và mục tiêu [11] của hai mô tả.

Để giải quyết vấn đề này, một số phương pháp [21] sử dụng các mô hình học sâu nâng cao hơn (ví dụ: Recurrent Neural Network) để tìm hiểu các mẫu mô tả ngôn ngữ tự nhiên và chuỗi lệnh gọi API tuần tự. Họ ánh xạ vấn đề đối sánh truy vấn-API như một tác vụ dịch máy. Tuy nhiên, các phương pháp này là phương pháp học có giám sát, yêu cầu dữ liệu đào tạo quy mô lớn (các cặp nhận xét về phương pháp và chuỗi lệnh gọi API), và chỉ có thể nắm bắt các truy vấn thường xuyên nhất và các mẫu sử dụng API. Do đó, đối với phần đuôi dài của các API ít được sử dụng hơn, các nhà phát triển vẫn phải sử dụng các phương tiện khác, chẳng hạn như truy xuất dựa trên tài liệu.

Khoảng cách từ vựng giữa mô tả chức năng API và truy vấn của người dùng rộng hơn nhiều so với chỉ không khớp trình tự. Ví dụ: API java.lang.Integer.parseInt (String) là API chính xác cho truy vấn "convert String to Integer". Thật không may, mô tả chức năng của API này là "Parses the string argument as a signed decimal integer". Không có phương pháp học sâu nào có thể xử lý khoảng cách rộng này giữa mô tả chức năng và truy vấn của người dùng. Để khớp với mô tả chức năng và truy vấn của người dùng ở hai bên của khoảng trống này, chúng ta phải hiểu cấu trúc chi tiết của câu và vai trò ngữ nghĩa của các bộ phận của chúng, ví dụ, các cụm động từ chính và vai trò ngữ nghĩa "{source}" và "{goal}" liên quan đến các cụm từ "convert { source } to { goal }" và "phân tích cú pháp {source} thành {goal}". Một số nhà nghiên cứu đã cố gắng giải quyết khoảng cách này bằng cách yêu cầu các nguồn lực bên ngoài (ví dụ: các cuộc thảo luận về Stack Overflow) để tăng cường truy vấn của người dùng, ví dụ như Biker [27] và QECK [35], nhưng những điều này chỉ có thể nắm bắt các API được thảo luận nhiều nhất trong các nguồn lực bên ngoài và bị từ nhiễu thông tin trong các nguồn bên ngoài này.

Trong công việc này, nhóm tác giả cho rằng các cụm động từ và các mẫu cụm từ của chúng là rất cần thiết trong cả việc mô tả chức năng của các phương thức API và giải thích các truy vấn của người dùng. Ta gọi các động từ được sử dụng để mô tả chức năng phương thức là động từ chức năng. Do đó, nhóm tác giả tranh luận rằng các truy vấn của người dùng và mô tả API nên được khớp về mặt chức năng của các động từ và mẫu cụm từ, theo ngữ nghĩa mà chúng diễn đạt hơn là sự giống nhau về mặt từ vựng của chúng. Nghiên cứu trước đây cũng đã xem xét tầm quan trọng của cấu trúc động từ-đối tượng trong định danh mã nguồn [22, 24–26, 28, 29], và đối với tài liệu phần mềm, Treude và các cộng sự [40] tập trung vào việc trích xuất các nhiệm vụ phát triển, được mô tả bằng các động từ từ một danh sách xác định trước, được kết hợp với tân ngữ trực tiếp và / hoặc một cụm giới từ. Tuy nhiên, ít người biết về cách các cụm động từ được sử dụng trong mô tả chức năng API và truy vấn của người dùng cũng như sự không nhất quán trong việc sử dụng chúng ảnh hưởng như thế nào đến việc truy xuất API.

Ta giả thuyết rằng hầu hết các chức năng API có thể được mô tả với một số giới hạn các động từ chức năng thường được sử dụng và các chức năng có thể được phân loại thêm thành một số lượng nhỏ các danh mục chức năng. Ví dụ: động từ “return”, khi được sử dụng trong cụm từ “return {source} as {goal}”, diễn tả các chức năng của transformation and conversion, và các chức năng này có thể được xếp vào cùng một loại. Ta giả thuyết thêm rằng các danh mục chức năng và mẫu cụm từ của truy vấn người dùng và mô tả API có thể được nhận dạng tự động. Cuối cùng, tác giả lập luận rằng đối sánh chi tiết có thể được thực hiện giữa các truy vấn của người dùng và mô tả API bằng cách căn chỉnh các danh mục chức năng và các mẫu cụm từ của chúng. Ví dụ: cả truy vấn “convert String to Integer” và mô tả API “parses the string argument as a signed decimal integer” đều có thể được phân loại thành cùng một danh mục chức năng “convert/transform/turn” và những người tham gia (tức là “{ source} ”và“ {goal} ”) trong cụm từ của họ có thể được căn chỉnh bằng cách đối sánh rõ ràng.

Để xác minh giả thuyết của mình, tác giả tiến hành một nghiên cứu thực nghiệm để điều tra các động từ chức năng và danh mục chức năng cũng như các mẫu cụm từ có trong mô tả chức năng API từ tài liệu tham khảo của JDK và Android.

Nhóm tác giả đã phân tích thủ công các mô tả chức năng của 14.733 phương thức JDK và Android API. Các mô tả này chứa 356 động từ chức năng khác nhau và các động từ này có thể được nhóm thành 87 loại chức năng dựa trên ngữ nghĩa của chúng trong ngữ cảnh mô tả. Mỗi danh mục chức năng chứa từ 1 đến 28 (trung bình 4,71 và 2,5 trung bình) các động từ chức năng khác nhau. Đối với tất cả các danh mục chức năng, 80% mô tả API được mô tả bằng 1-5 mẫu cụm từ (giá trị trung bình là 2,3, trung vị 2).

Dựa trên những phát hiện thực nghiệm của về các động từ chức năng, nhóm đề xuất phương pháp tiếp cận đề xuất phương thức API dựa trên sự đối sánh rõ ràng của các cụm động từ chức năng trong truy vấn của người dùng và mô tả API, được gọi là PreMA. Phương pháp này đào tạo một bộ phân loại để dự đoán các danh mục chức năng

(được tóm tắt trong nghiên cứu thực nghiệm) của các mô tả API. Sau đó, nó đối sánh các câu mô tả với các mẫu cụm từ của các danh mục chức năng tương ứng để xác định các mẫu cụm từ được chấp nhận. Để đề xuất các phương thức API, PreMA phân tích cú pháp truy vấn API do người dùng đưa ra theo cách tương tự. Sau đó, nó đối sánh truy vấn của người dùng đã được phân tích cú pháp với câu mô tả API đã được phân tích cú pháp bằng cách căn chỉnh các danh mục chức năng và các mẫu cụm từ của chúng. Cuối cùng PreMA trả về hoàn toàn hoặc một phần (ví dụ: các danh mục chức năng giống nhau nhưng các đối tượng hoạt động khác nhau) các API phù hợp dưới dạng kết quả truy vấn, cùng với giải thích ngôn ngữ về đối sánh.

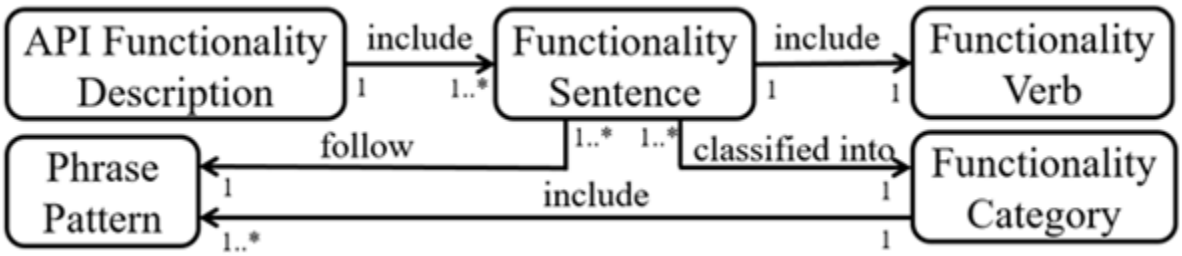
Nhóm đánh giá độ chính xác phân tích câu của PreMA dựa trên dữ liệu được chú thích trong nghiên cứu thực nghiệm. Kết quả cho thấy độ chính xác của phân loại chức năng và nhận dạng mẫu cụm từ cao (tương ứng là 92,8% và 90,4%). Nhóm cũng đánh giá hiệu suất của PreMA trong việc truy xuất API dựa trên tài liệu bằng cách so sánh nó với cách tiếp cận dựa trên Word2Vec [33]. Kết quả cho thấy những người tham gia sử dụng PreMA hoàn thành nhiệm vụ của họ chính xác hơn (0,77 so với 0,54) với ít lần thử lại hơn (2,16 so với 3,30) và sử dụng ít thời gian hơn (98,29 giây so với 113,42 giây). Nhìn chung, bài báo này có những đóng góp sau:

- Nhóm đã tiến hành một nghiên cứu thực nghiệm quy mô lớn về các mô tả chức năng của 14.733 phương pháp API JDK và Android. Nhóm đã xác định 356 động từ chức năng thường được sử dụng, 87 danh mục chức năng và 523 mẫu cụm từ từ mô tả.
- Nhóm tác giả đề xuất phương pháp tiếp cận đề xuất phương pháp API dựa trên sự đối sánh rõ ràng của các cụm động từ chức năng trong truy vấn của người dùng và mô tả API.
- Nhóm đã đánh giá cách tiếp cận được đề xuất về độ chính xác của phân tích mô tả chức năng API và hiệu suất của việc truy xuất API. Dữ liệu và kết quả phân tích của nghiên cứu và đánh giá thực nghiệm được đưa vào replication package [7].

2.Định nghĩa

Các khái niệm và mối quan hệ chính được sử dụng để mô hình hóa rõ ràng các mô tả chức năng API như thể hiện trong Hình 1.

Tài liệu tham khảo sẽ thể hiện mô tả về mục đích chính của mỗi phương thức API. Ta gọi mô tả này là mô tả chức năng API (hoặc viết tắt là f_description). f_description có thể bao gồm một số câu mô tả các chức năng của phương thức, ta gọi là câu chức năng (hoặc f_sentences). Mỗi f_sentence có thể bao gồm một số động từ hoặc cụm động từ, nhưng chính xác là một động từ chức năng (hoặc f_verb), biểu thị hành động chính của chức năng. Ví dụ, f_sentence "Cố gắng hủy bỏ việc thực hiện nhiệm vụ này" bao gồm hai động từ ("cố gắng" và "hủy bỏ"), trong khi chỉ có "hủy bỏ" là f_verb. Đối với một câu ghép mô tả nhiều chức năng, chúng ta có thể chia nó thành nhiều f_sentences, mỗi f_sentence chỉ mô tả một chức năng.



Hình 1: Mô hình thuật ngữ

Một f_sentence có thể được phân loại thành một loại chức năng (hoặc f_category), dựa trên ý nghĩa của f_verb của nó trong ngữ cảnh hiện tại. Ví dụ: API org.omg.CosNaming.NamingContextExtOperations.to_name (String) chứa f_sentence này trong f_description của nó: "This operation converts a Stringfield Name into an equivalent array of Name Components."(Tạm dịch: Thao tác này chuyển đổi Tên trường chuỗi thành một mảng các Thành phần Tên tương đương). Nó bao gồm f_verb "convert" (được gạch dưới) và thuộc về f_category "convert / transform / turn", được định nghĩa là "chuyển đổi thứ gì đó sang dạng khác".

Hai f_sentences có thể được phân loại thành hai f_categories riêng biệt, ngay cả khi chúng sử dụng cùng một f_verb. Đó là bởi vì f_verb có thể có các ý nghĩa khác nhau trong các ngữ cảnh khác nhau. Ví dụ: “return” được sử dụng rộng rãi trong f_sentences như là f_verbs với các ý nghĩa khác nhau, biểu thị các f_categories khác nhau, chẳng hạn như: “get / return / obtain” (ví dụ: “Returns the state of this thread”), hoặc “check/test/determine”(ví dụ:“ Return if the Type is a cube map ”) hoặc “convert/transform/parse ”(ví dụ:“ Returns the BigDecimal as a character array ”). Đồng thời, một f_category có thể tương ứng với một số f_verbs riêng biệt có chung ý nghĩa trong các ngữ cảnh khác nhau. Ví dụ: trong tài liệu tham khảo JDK và Android có nhiều f_verbs với ý nghĩa được nắm bắt bởi f_category “create / build / construct”, chẳng hạn như: “create”, “build”, “produce”, “construct”, “generate”, “establish”, “make”, “instantiate”, v.v.

Các f_sentences từ một f_category chia sẻ một tập hợp các mẫu cụm từ (hoặc p_patterns) và mỗi câu tuân theo một trong số chúng. Một p_pattern bao gồm các phần tử sau:

1) một f_category, ví dụ: “convert/transform/turn”;

2) giới từ, ví dụ, “in”, “at”, “from”;

3) các vai trò ngữ nghĩa mô tả các quan hệ khái niệm giữa những bên tham gia trong p_pattern, ví dụ: “location”, “patient”, “source”, “goal”;

4) các từ dẫn đến mệnh đề (clause leader), ví dụ, “that”, “whether”;

5) các mệnh đề, ví dụ, một mệnh đề được dẫn đầu bởi “whether”;

6) động từ nguyên thể có to (ví dụ: "to be called") và Ving (ví dụ: "reading a file").

Ví dụ, f_sentence “This operation converts a Stringfield Name into an equivalent array of Name Components” theo “V {source} đến / as / into {goal}” p_pattern, trong đó “V” biểu thị f_category, trong khi “{source}” và “{goal}” biểu thị vai trò ngữ nghĩa. Lưu ý rằng các vai trò ngữ nghĩa là nhất quán mặc dù có sự thay đổi trong cú pháp [11], do đó có thể sắp xếp các vai trò ngữ nghĩa giữa các p_patterns khác nhau.

VerbNet [10] là một từ điển về động từ độc lập theo miền và có phạm vi rộng cho tiếng Anh. Nó chứa khoảng 5.800 động từ. Chúng được phân loại thành hơn 270 lớp và mỗi lớp chứa một tập hợp các khung cú pháp mà các thành viên của lớp thường sử dụng. Nhóm tác giả không sử dụng các lớp và khung VerbNet làm f_categories và p_patterns vì hai lý do sau. Đầu tiên, một số động từ được sử dụng trong API f_sentences (ví dụ: “unmarshal”, “iterate”) không có trong VerbNet. Thứ hai, các lớp VerbNet không được định nghĩa bởi ý nghĩa của các động từ. Ví dụ: “open” và “close” đều thuộc cùng một lớp “crane” trong VerbNet, nhưng được phân loại thành các f_categories khác nhau. Nhóm quyết định chú thích f_categories và p_patterns của riêng mình, nhưng sử dụng 12 vai trò ngữ nghĩa được xác định trong VerbNet: “duration”, “location”, “goal”, “source”, “patient”, “instrument”, “beneficiary”, “attribute”, “theme”, “material”, “topic”, “product”.

3.Nghiên cứu thực nghiệm

Nhóm tác giả đã tiến hành một nghiên cứu thực nghiệm để hiểu f_verbs, p_patterns và f_categories nào biểu diễn trong API f_sentences. Đặc biệt, nhóm tập trung trả lời các câu hỏi nghiên cứu sau:

RQ1: Các verb nào được sử dụng trong API f_sentences?

RQ2: Các f_categories nào mà các f_sentence có thể được phân loại vào?

RQ3: Các p_pattern nào được sử dụng trong các f_sentence từ mỗi f_category?

Nhóm tập trung vào API JDK và Android dựa trên tài liệu tham khảo của JDK 1.8 [6] và Android 27 [1] cũng như trình bày và phân tích kết quả của nghiên cứu thực nghiệm, để trả lời ba câu hỏi nghiên cứu.

3.1.Thiết kế nghiên cứu

3.1.1.Phân tích động từ (RQ1)

Từ các khai báo API bán cấu trúc trong tài liệu tham khảo JDK và Android, nhóm đã trích xuất 38,819 và 29,125 phương thức API (bao gồm cả các hàm tạo) tương ứng bằng cách sử dụng BeautifulSoup [4]. Nhóm đã lọc ra các phương thức không có f_description hoặc có f_description đáp ứng một trong các điều kiện sau: (1) nêu rõ phương thức ghi đè, ví dụ: “Overrides hashCode”; (2) nêu rõ một phương thức không được chấp nhận, ví dụ: “Deprecated. Use getTimeToLive instead”; (3) đề xuất kiểm tra mô tả của một API khác, ví dụ: “See getenv()”; (4) đề xuất một API khác có cùng chức năng, ví dụ: “Same as charCount(int)”. Sau khi lọc, nhóm thu được 54.256 phương thức (31.618 JDK và 22.638 Android). Đối với mỗi phương thức còn lại, nhóm trích xuất câu đầu tiên của mô tả của nó từ tài liệu tham khảo dưới dạng f_sentence của nó. Nếu câu đó là câu ghép, chúng ta chia nó thành nhiều f_sentences.

Nhóm đã sử dụng SpaCy [8] (một thư viện mã nguồn mở để xử lý ngôn ngữ tự nhiên) để phân tích cú pháp f_sentence của mỗi phương thức API và xác định các động từ được sử dụng trong câu. Nếu một f_sentence bao gồm nhiều hơn một động từ thì xem xét tất cả chúng để trả lời câu hỏi nghiên cứu đầu tiên. Nhóm đã sử dụng SpaCy để bổ sung các động từ đã xác định về dạng bình thường của chúng. Sau khi trích xuất, nhóm phân tích tần suất và sự phân bố của các động từ.

3.1.2.Phân tích danh mục chức năng (RQ2)

Việc xác định f_categories và f_verbs từ JDK và Android được thực hiện thông qua phân tích định tính của f_sentences, sử dụng mã hóa mở. Việc mã hóa được thực hiện trong hai giai đoạn: giai đoạn xác định giao thức mã hóa và giai đoạn chú thích. Trong giai đoạn đầu, các chuyên gia đã xác định và định nghĩa một tập hợp các f_categories (tức là các mã) dựa trên một tập hợp con các mô tả API. Trong giai đoạn chú thích, một nhóm lớn hơn các nhà chú thích đã được đào tạo để sử dụng giao thức mã hóa được phát triển trong giai đoạn đầu tiên để mã hóa một bộ f_sentences lớn hơn.

Việc chọn f_sentences ngẫu nhiên cho chú thích có thể dẫn đến một số lượng lớn các câu có động từ "get / set", với mức độ phổ biến của chúng. Thay vào đó, đối với giai đoạn mã hóa đầu tiên, nhóm lấy mẫu ngẫu nhiên 10 f_sentences cho mỗi 116 động từ được sử dụng thường xuyên nhất ở gốc của cây phân tích cú pháp (tức là các động từ của mệnh đề chính). Nhóm tập trung vào những động từ này, vì chúng có nhiều khả năng là f_verbs (hãy nhớ rằng RQ1 đã xem xét tất cả các động từ trong f_sentences). Số lượng động từ này (tức là 116) được chọn dựa trên kết quả của RQ1 (xem Phần 3.2). Những động từ này bao gồm 84% f_sentences được sử dụng trong RQ1 - tất cả 116 động từ thường gặp nhất được xác định trong RQ1 đều là động từ nút gốc. Sau khi loại bỏ các bản sao, chúng tôi thu được 1.139 f_sentences (590 từ JDK và 549 từ Android), được sử dụng cho giai đoạn mã hóa ban đầu.

Đối với giai đoạn chú thích, nhóm lấy mẫu ngẫu nhiên 20.000 API f_sentences bất kể tần suất xuất hiện của các động từ. Để có một tập dữ liệu cân bằng hơn (tức là không có quá nhiều câu "get / set"), nhóm đã tinh chỉnh thêm các mẫu, dựa trên động từ nút gốc của chúng. Nếu một động từ nút gốc xuất hiện trong hơn 1.000 f_sentences được lấy mẫu, thì nhóm chọn ngẫu nhiên và giữ lại 1.000 f_sentences đó. Nhóm đã giữ tất cả các f_sentences cho các động từ nút gốc xuất hiện trong ít hơn 1.000 mẫu. Bằng cách này, các f_sentences sử dụng các động từ không phổ biến cũng có thể được đưa vào. Cuối cùng, chúng tôi đã thu được 13.635 f_sentences cho giai đoạn chú thích (7.716 từ JDK và 5.919 từ Android).

Việc viết mã ban đầu được thực hiện bởi ba trong số các tác giả, họ là những chuyên gia trong lĩnh vực phát triển Java và Android, như sau. Đầu tiên, một API f_sentence được phân bổ ngẫu nhiên cho một trình chú thích. Thứ hai, người chú thích kiểm tra API và f_sentence của nó và xác định f_verb được sử dụng trong đó. Thứ ba, người chú thích cố gắng phân loại (tức là chú thích) API f_sentence thành một f_category hiện có (tức là mã hiện có). Nếu không tìm thấy mã nào, thì mã mới (tức là f_category) được tạo và cung cấp định nghĩa. Một mã đặc biệt "Unknown" được tạo để chứa các câu không thực sự là f_sentences (ví dụ: "Equivalent to the codePointCount(char[], int, int) method, for convenience"). Mỗi f_sentence được mã

hóa bởi hai chú thích độc lập và nếu chú thích của chúng (f_verb hoặc f_category) khác nhau, thì người chú thích thứ ba sẽ được chỉ định để giải quyết xung đột. Quá trình trên được lặp lại cho đến khi tất cả các mẫu được chú thích.

Chú thích được thực hiện bởi 10 sinh viên (2 tiến sĩ và 8 sinh viên MS), những người đã quen thuộc với phát triển Java và Android. Trước khi chú thích, các sinh viên đã được đào tạo bởi các chuyên gia xác định công cụ mã hóa. Khóa đào tạo được thực hiện theo nhóm và kéo dài hơn một giờ. Tài liệu thêm về đào tạo, bao gồm các ví dụ, có sẵn trong replication package [7]. Chú thích được thực hiện bằng cách sử dụng các mã (tức là f_categories) được xác định trong giai đoạn xác định giao thức mã hóa và tuân theo quy trình tương tự. Như trong giai đoạn đầu tiên, người chú thích có thể tạo mã mới khi cần thiết. Mã cho mỗi f_category là một trong những f_verbs (được đặt tên là nhãn f_verb), được sử dụng thường xuyên (nhưng không nhất thiết là được sử dụng thường xuyên nhất), phản ánh tốt nhất ý nghĩa của f_category và không được sử dụng để gắn nhãn cho f_category khác.

Để tạo thuận lợi cho việc viết mã, chúng tôi đã phát triển một công cụ chú thích dựa trên web. GUI của công cụ chú thích có sẵn trên web [2].

3.1.3.Phân tích mẫu cụm từ (RQ3)

Như đã đề cập ở trên, để nắm bắt được chức năng của một phương thức, f_verb là chưa đủ, và p_patterns là điều quan trọng để thiết lập ngữ cảnh. Nhóm điều tra các p_patterns được sử dụng trong mỗi f_category, được xác định trong RQ2, bằng cách chú thích tất cả 14.774 (giai đoạn mã hóa ban đầu 1.139 và giai đoạn chú thích 13.635) f_sentences. Đối với mỗi f_sentence, hai tác giả đã chú thích p_pattern một cách độc lập dựa trên các nguyên tắc chú thích VerbNet [11]. Nếu các chú thích khác nhau, một tác giả thứ ba đã được chỉ định để giải quyết xung đột bằng chiến lược bỏ phiếu đa số.

3.2.Kết quả phân tích động từ (RQ1)

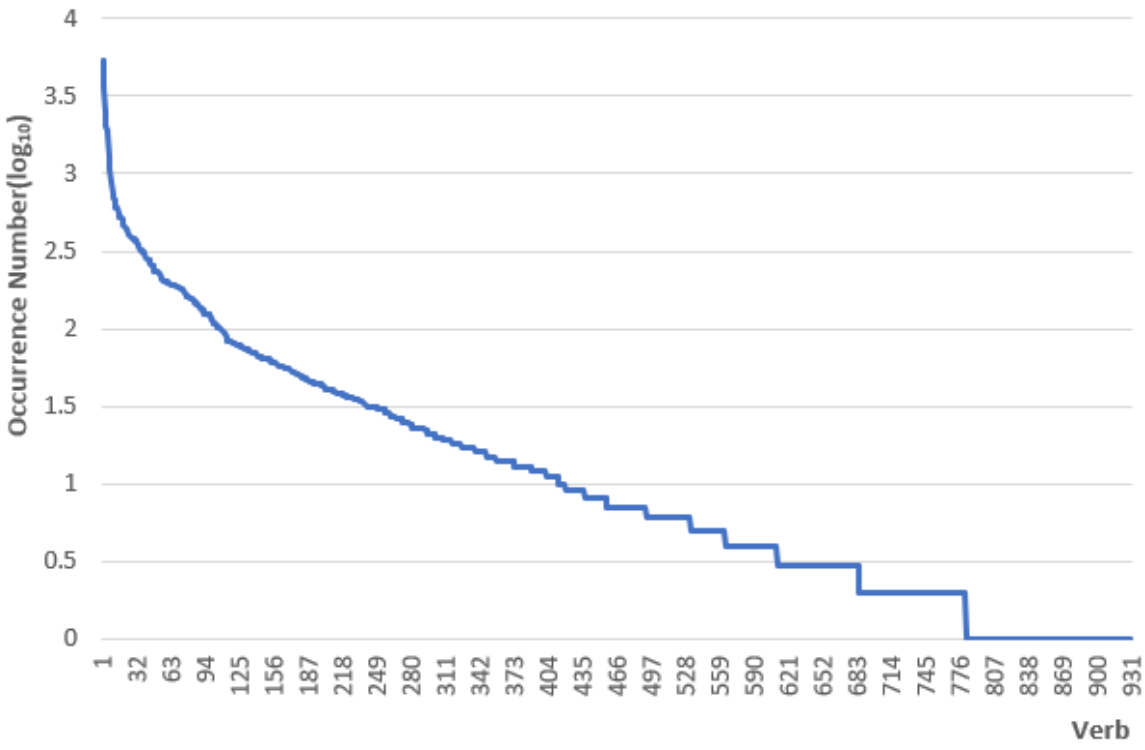
Nhóm tác giả xác định 931 động từ khác nhau từ f_sentences của 54.256 phương thức API JDK và Android. Bảng 1 cho thấy 30 động từ được sử dụng thường xuyên nhất và sự xuất hiện của chúng. “Return” cho đến nay là động từ được sử dụng nhiều nhất, theo sau là “set” và “get”. Đây không phải là điều đáng ngạc nhiên, dựa trên kinh nghiệm của nhóm với mã và tài liệu. Một quan sát thú vị là 30 động từ hàng đầu này không phải là miền cụ thể, điều này ngụ ý rằng nhóm mong đợi chúng xuất hiện trong các thư viện khác, từ các miền khác nhau. Nhóm đã kiểm tra tất cả các động từ và nhận thấy rằng hầu hết chúng không thuộc miền cụ thể (dành cho Android hoặc Java). Ví dụ: “dial” là một động từ dành riêng cho miền dành cho Android. Tiếp theo là một số động từ miền cụ thể được xác định và tần suất của chúng: (“mute”, 3), (“dengrate”, 7), (“suffix”, 7), (“prefix”, 5), (“negotiate”, 3), (“dial”, 3), (“snooze”, 1), (“absorb”, 1), (“advertise", 7), (“roam”, 6), (“introspect”, 5)

Bảng 1: 30 động từ hàng đầu được sử dụng nhiều nhất

Động từ	Lần xuất hiện	Động từ	Lần xuất hiện	Động từ	Lần xuất hiện
return	16,268	indicate	730	do	478
set	5,406	determine	691	retrieve	458
get	3,624	write	676	give	446
call	2,472	change	602	contain	436
have	2,034	obtain	599	support	403
create	1,898	read	558	specify	400
use	1,837	check	524	convert	396

add	1,259	insert	520	update	390
remove	1,067	perform	518	describe	387
invoke	877	start	508	notify	383

Hình 2 cho thấy sự phân bố của 931 động từ (trục Y trong thang log). Phân tích phân phối cho thấy rằng 87 (9,34%) động từ xuất hiện thường xuyên nhất trong 80% API f_sentences. Nếu chúng ta loại trừ “return” là ngoại lệ, thì 115 (12,37%) động từ thường gặp nhất sẽ xuất hiện trong 80% API f_sentences không bao gồm “return”. Nếu chúng ta bao gồm "return", thì 116 (12,46%) động từ thường gặp nhất sẽ xuất hiện trong 84,28% API f_sentences. Nói cách khác, một số lượng tương đối nhỏ các động từ bao hàm hầu hết tất cả các API f_sentences.



Hình 2: Sự phân bố của các lần xuất hiện động từ

3.3.Các danh mục chức năng (RQ2)

Với số lượng chú thích và mã, ta sử dụng hệ số Cohen’s Kappa [32] để đo tỷ lệ giống nhau giữa các chú thích. Đối với giai đoạn mã hóa ban đầu, Kappa là 0,724 và đối với giai đoạn chú thích thứ hai là 0,700. Những người chú giải đã xác định được 87 f_categories (không bao gồm "Unknown"), trong đó 50 được xác định trong giai đoạn mã hóa đầu tiên và 37 trong giai đoạn thứ hai. Trong số 87 f_categories, 65 bao gồm cả API JDK và API Android, 10 chỉ bao gồm các API JDK và 12 chỉ bao gồm các API Android.

87 f_categories chứa 356 f_verbs, tám trong số đó xuất hiện trong hơn hai f_categories. “Return” xuất hiện trong bảy f_categories, “determine” xuất hiện trong sáu f_categories, “indicate” xuất hiện trong năm, “tell”, “retrieve” và “give” xuất hiện trong bốn, trong khi "get" và "notify" xuất hiện trong ba . Có 26 f_verbs xuất hiện trong hai f_categories. Tất cả 322 f_verb khác xuất hiện trong một f_category duy nhất. Trung bình, một f_category chứa 4,71 f_verbs (trung vị là 2,5). f_category “convert/transform/...” chứa nhiều f_verb nhất (tức là 28), trong khi 30 (34,48%) f_categories chứa một f_verb duy nhất. Lưu ý rằng các động từ không phổ biến có thể được đưa vào một f_category cùng với các động từ phổ biến

Nhóm tác giả đã so sánh 356 f_verbs với danh sách 202 hành động lập trình do Treude và các cộng sự công bố [40]. Các tác vụ lập trình của chúng có ngữ nghĩa tương tự như f_verbs của nhóm, nhưng dựa trên một tập dữ liệu nhỏ hơn nhiều. Nhóm thấy rằng 121 trong số 356 f_verbs của nhóm (34,0%) cũng được họ xác định là các hành động lập trình, trong khi nghiên cứu thực nghiệm của nhóm xác định thêm 235 f_verbs.

Nhóm xác định nhãn f_verb là f_verb đại diện của một f_category. Đối với mỗi f_category, ba trong số các đồng tác giả đã chọn một f_verb từ tất cả các f_verbs trong f_category này làm nhãn f_verb của f_category này thông qua thảo luận. Hai phương pháp heuristics được sử dụng để chọn nhãn f_verb: (1) Các đồng tác giả kiểm tra f_verbs trong f_category theo tần suất từ cao xuống thấp cho đến khi xác định được nhãn f_verb.

(2) Một f_verb chỉ được coi là nhãn f_verb nếu ý nghĩa của nó bao hàm ý nghĩa của f_category và không có sự nhầm lẫn với f_category khác.

10 f_categories hàng đầu dựa trên số lượng f_sentences và nhãn f_verbs của chúng được hiển thị trong Bảng 2.

Bảng 2: 10 f_categories được sử dụng thường xuyên nhất

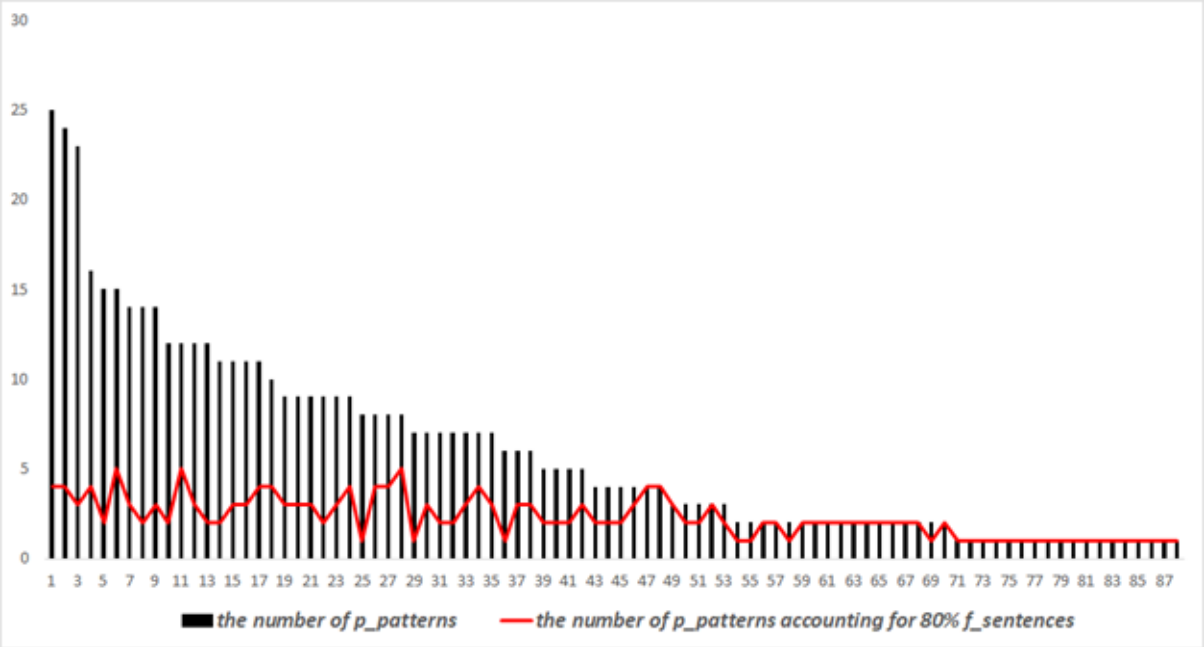
f_categories	Nhãn *f_verb*	Số *f_sentences*
get/return/obtain/...	get	3021
set/control/configure/...	set	1303
check/test/determine/...	check	977
create/build/construct/...	create	784
append/add/insert/...	append	777
call/invoke/notify/...	call	762
perform/execute/run/...	perform	409
convert/transform/parse/...	convert	393
remove/delete/exclude/...	remove	348
write/record/output /..	write	293

3.4.Mẫu cụm từ (RQ3)

Hai người chú thích được coi là đạt được thỏa thuận nếu các chú thích p_pattern của họ cho một f_sentence giống nhau. Kết quả là, tỷ lệ đồng thuận cho chú thích p_pattern là 90,2% (tức là đồng thuận gần như hoàn hảo). Các p_patterns được xác định từ f_sentences của cùng một f_category được tổng hợp để lấy p_patterns của mỗi

f_category. Xin lưu ý rằng các p_patterns thuộc cùng một f_category và chỉ khác nhau về giới từ hoặc giới từ đứng đầu mệnh đề (clause leader) đã được hợp nhất thành một, ví dụ: “V {source} to/as/into {goal}” cho f_category “convert/transform/ parse”.

Hình 3 cho thấy sự phân bố của số p_pattern trên f_categories. Số lượng p_patterns cho mỗi 87 f_categories thay đổi từ 1 đến 25 (trung bình 6, trung vị 4). 3 f_categories hàng đầu có nhiều p_patterns nhất là “set / control / configure” (25), “append / put / add” (24) và “get / return / get” (23). Có 18 f_categories chỉ có một p_pattern, ví dụ: “lock”, “touch / press”, “collect/recycle/sample”.



Hình 3: Phân bố của p_pattern qua f_categories

Đối với mỗi f_category, nhóm tác giả đã phân tích số lượng p_patterns chiếm 80% f_sentences, được biểu thị bằng đường màu đỏ trong Hình 3. Nhóm nhận thấy rằng, đối với tất cả f_categories, 80% f_sentences được mô tả bởi 1-5 p_patterns (trung bình 2.3, trung vị 2). Ví dụ: f_category “append / add / insert” có 24 p_patterns, trong khi 4 trong số chúng chiếm 80% f_sentences.

Các p_patterns được xác định có từ 0-4 vai trò ngữ nghĩa. Số lượng p_patterns có vai trò ngữ nghĩa 0, 1, 2, 3, 4 lần lượt là 28 (5,4%), 166 (31,7%), 216 (41,3%), 110 (21,0%) và 3 (0,6%). Chúng ta có thể thấy rằng 73% p_patterns là những p_patterns

đơn giản với 1 hoặc 2 vai trò ngữ nghĩa. Ví dụ về p_patterns có 3 vai trò ngữ nghĩa là "V {patient} from {source} as/into/ to {goal}" cho danh mục "convert/transform/parse" f_category; f_sentence theo sau p_pattern này là "Convert a long datetime from the given time scale to the universal time scale.".

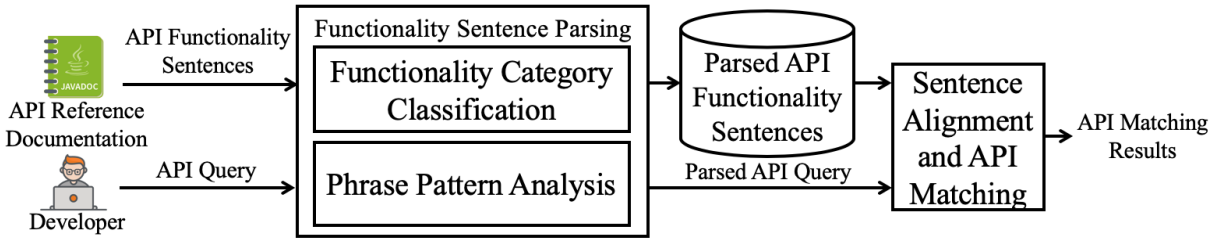
4. Cách tiếp cận

Nghiên cứu thực nghiệm của cho thấy rằng hầu hết các f_sentence API của JDK và Android có thể được phân loại thành một tập hợp giới hạn gồm 87 f_categories với 1-5 p_patterns (trung bình là 2,33) được sử dụng cho mỗi f_category. Những phát hiện này ngụ ý cách phân tích động từ có thể được sử dụng để đối sánh giữa truy vấn API và f_sentence: trước tiên hãy nhận biết f_categories và p_patterns của chúng; sau đó sắp xếp chúng dựa trên f_categories và p_patterns để đối sánh chi tiết giữa những bên tham gia tương ứng.

Dựa trên ý tưởng này, nhóm tác giả đề xuất một phương pháp tiếp cận PreMA để khớp các mô tả chức năng API như thể hiện trong Hình 4. Với f_sentences từ tài liệu tham khảo API, phương pháp này phân tích cú pháp các câu bằng cách phân tích f_categories và p_patterns của chúng. Các f_sentences phân tích sau đó được lưu trữ để phân tích thêm. Khi được sử dụng để tìm kiếm API, PreMA phân tích cú pháp truy vấn API do nhà phát triển phát hành theo cách tương tự. Sau đó, nó khớp với truy vấn API được phân tích cú pháp với f_sentences được phân tích cú pháp bằng cách căn chỉnh chúng dựa trên f_categories và p_patterns. Kết quả đối sánh API bao gồm toàn bộ hoặc một phần (ví dụ: các f_categories giống nhau nhưng không có người tham gia khác nhau) các API phù hợp và f_sentences của chúng, cùng với các giải thích về đối sánh.

Việc triển khai sử dụng BeautifulSoup để phân tích cú pháp các trang HTML của tài liệu tham khảo API. Nó trích xuất tất cả các phương thức API với f_description của chúng và lọc ra các quy tắc sử dụng phương pháp API không hợp lệ (giống như trong Phần 3.1.1). Đối với mỗi phương thức còn lại, nhóm trích xuất câu đầu tiên của mô tả

của nó từ tài liệu tham chiếu dưới dạng f_sentence của nó để phân tích cú pháp câu chức năng.



Hình 4: Tổng quan về PreMA

4.1 Phân loại danh mục chức năng

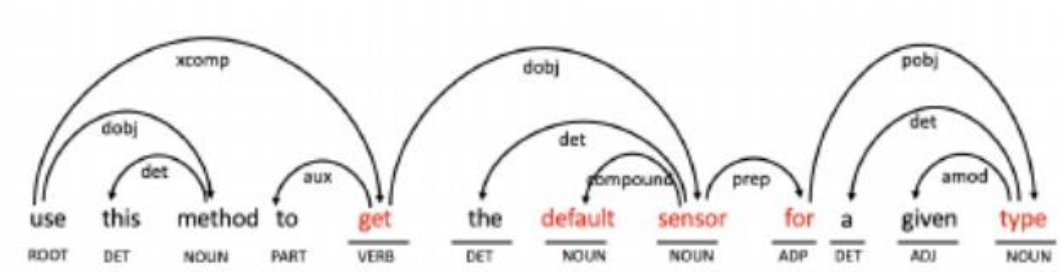
87 f_categories và 14.774 f_sentences được chú thích do nghiên cứu thực nghiệm cung cấp cho phép phân loại tự động các f_sentences vào f_categories. Nhóm coi phân loại f_category như một nhiệm vụ phân loại văn bản và sử dụng dữ liệu chú thích f_sentence để đào tạo trình phân loại cho nhiệm vụ. Bộ phân loại lấy một câu (truy vấn f_sentence or) làm đầu vào và trả về một trong 87 f_categories hoặc danh mục "Unknown" làm đầu ra.

Nhóm triển khai trình phân loại dựa trên BERT [18] (Biểu diễn bộ mã hóa hai chiều từ Máy biến áp), một mô hình ngôn ngữ hiện đại. Mô hình được sử dụng để học các biểu diễn của các từ: nó lấy đầu vào là một chuỗi các từ và đầu ra là biểu diễn vectơ phân bố của chuỗi từ [41]. Google cung cấp hai mô hình BERT được đào tạo trước (BERT-base, BERT-Large), được đào tạo trên một kho ngữ liệu không có nhãn quy mô lớn để nắm bắt các đặc điểm ngữ nghĩa phong phú. Các mô hình được đào tạo trước có thể được tùy chỉnh bằng cách thêm một lớp đầu ra và được tinh chỉnh dựa trên dữ liệu được gắn nhãn cho các tác vụ NLP (Xử lý ngôn ngữ tự nhiên) cụ thể như phân loại văn bản và trả lời câu hỏi. Nhóm sử dụng mô hình cơ sở BERT đã được đào tạo trước và thêm một lớp phân loại (lớp được kết nối đầy đủ) với các f_categories được xác định trong nghiên cứu thực nghiệm và sau đó tinh chỉnh mô hình dựa trên tập hợp dữ liệu đào tạo bao gồm các cặp f_sentence-f_category.

4.2 Phân tích mẫu cụm từ

Đưa ra một câu (f_sentence hoặc truy vấn) và f_category của nó, phân tích cụm từ xác định p_pattern được sử dụng trong đó. Vì nghiên cứu thực nghiệm đã xác định một tập hợp p_patterns cho mỗi f_category, phân tích chỉ cần đối sánh câu với p_patterns của f_category mà nó thuộc về. Nhóm sử dụng Spacy [8], hoạt động tốt trên tài liệu Java API [13], để gắn thẻ POS (Part of Speech) và phân tích cú pháp phụ thuộc của câu. Sau đó, nhóm xác định f_verb được sử dụng trong câu và sau đó là p_pattern.

4.2.1 *Nhận dạng động từ chức năng*. Phân loại danh mục chức năng không xác định f_verb được sử dụng trong câu, vì vậy chúng ta cần xác định f_verb dựa trên việc gắn thẻ POS và phân tích cú pháp phụ thuộc. Nghiên cứu thực nghiệm đã xác định một tập hợp các f_verbs cho mỗi f_category và nhiều trong số đó có thể xuất hiện trong câu. Nhóm sử dụng cách tiếp cận dựa trên heuristic để chọn từ các động từ ứng viên. Với một câu, nhóm duyệt qua cây phụ thuộc của nó theo thứ tự trước. Động từ ứng cử viên đầu tiên được duyệt được coi là f_verb của câu. Nếu không tìm thấy động từ ứng cử viên nào sau khi duyệt qua toàn bộ cây phụ thuộc, động từ đầu tiên đi qua trong câu được coi là f_verb, cho biết một f_verb mới cho f_category không được xác định trong nghiên cứu thực nghiệm. Hình 5 là cây phụ thuộc được sử dụng làm ví dụ về nhận dạng động từ chức năng. Cây thuộc về f_category “get/return/obtain”. Ví dụ: “dobj”, “pobj” và “xcomp” lần lượt đề cập đến tân ngữ trực tiếp, tân ngữ của giới từ và phần bổ sung mệnh đề mở. Câu này có hai động từ (tức là “use” và “get”). Khi duyệt qua cây phụ thuộc theo thứ tự trước, “use” là động từ được duyệt đầu tiên vì nó là nút gốc, nhưng nó không nằm trong bộ f_verb của f_category “get/return/obtain”; "get" là động từ được đảo ngược thứ hai và nó nằm trong bộ f_verb của f_category, vì vậy hãy chọn nó làm f_verb của câu.



Hình 5. Ví dụ về phân định động từ chức năng

4.2.2 *Nhận dạng cụm từ tìm kiếm.* Để xác định p_pattern cần phải đối sánh câu đã cho với các p_pattern của f_category mà câu đó thuộc về.

Để làm như vậy, trước tiên chúng ta cần xác định phần cốt lõi của f_sentence đã cho bởi mô tả chức năng API method. Điều này có thể được thực hiện bằng cách tìm cây con của cây phụ thuộc bắt nguồn từ f_verb. Ví dụ, đối với f_sentence được hiển thị trong Hình 5, phần "get" (được gạch dưới) là phần cốt lõi mô tả chức năng. Sau đó, từ phần cốt lõi, nhóm trích xuất cú pháp SP bằng cách phân tích cây phụ thuộc của phần cốt lõi. Nhóm thay thế các từ trong phần cốt lõi bằng một trình giữ chỗ cho các thành phần cú pháp Hoa Kỳ bằng cách sử dụng các quy tắc sau:

1)động từ bằng "V";

2) thay thế danh từ và cụm danh từ bằng "NP";

3) thay thế mệnh đề đối tượng và mệnh đề trạng ngữ bằng "S";

4) danh động từ = "S_ING";

5) thay thế động từ nguyên thể bằng "S_INF".

Clause leaders (ví dụ: "that", "whether", "if") và các giới từ (ví dụ: "in", "at", "to") trong mệnh đề chính được giữ lại. Việc thay thế được thực hiện bằng cách truy cập đệ quy vào cây con của mệnh đề cốt lõi. Ví dụ: mô hình cú pháp được xác định cho "This method will start proling if isProling() returns true." là "V S_ING if S" (Quy tắc

1, 3 và 4); và mô hình cú pháp được xác định cho "Registers the parameter named parameter Name to be of JDBC type sqlType" là “V NP S_INF” (Quy tắc 1, 2 và 5).

Sau khi có được mẫu cú pháp SP của một câu, nhóm tìm thấy p_pattern tương tự nhất trong số p_patterns ứng cử viên là tất cả p_patterns của f_category. Nhóm chia SP và p_patterns thành các thành phần nhỏ để so sánh này. Các giới từ có vai trò “NP” hoặc ngữ nghĩa tiếp theo được coi là một thành phần. Ví dụ: “V {patient} for {beneficiary}” có thể được chia thành ba thành phần “V”, “{patient}” và “for {beneficiary}”. Nhóm so sánh từng p_pattern với SP theo thứ tự của các thành phần để có được số lượng thành phần phù hợp trong SP. “NP” có thể khớp với bất kỳ vai trò ngữ nghĩa nào trong p_pattern. Nếu không phải tất cả các thành phần trong p_pattern đều có thể khớp trong SP thì xóa p_pattern này khỏi ứng cử viên p_patterns. Cuối cùng, chọn p_pattern từ ứng cử viên p_patterns có số lượng thành phần phù hợp cao nhất làm p_pattern của câu. Ví dụ: đối với câu trong Hình 5, sau khi đối sánh “V NP for NP” với tất cả p_patterns trong f_category “get/return/obtain”, ta chọn được hai ứng cử viên “V {patient}” và “V {patient} for {beneficiary}” và số thành phần phù hợp tương ứng là 2 và 3. Do đó, nhóm chọn “V {patient} for {beneficiary}” với tư cách là p_pattern.

4.3 Căn chỉnh câu và đối sánh API

Đưa ra một truy vấn API được phân tích cú pháp nhóm tác giả đối sánh nó với từng f_sentence FS được phân tích cú pháp và tính điểm phù hợp của chúng bằng cách căn chỉnh chúng dựa trên f_categories và p_patterns. Điểm phù hợp được tính bằng phương trình 1 bằng cách kết hợp ba điểm tương đồng khác nhau: độ tương đồng về f_category, độ giống nhau về vai trò ngữ nghĩa và độ giống nhau về văn bản. Sau đó, xếp hạng các phương pháp API bằng cách so khớp điểm và tạo lời giải thích cho từng phương pháp.

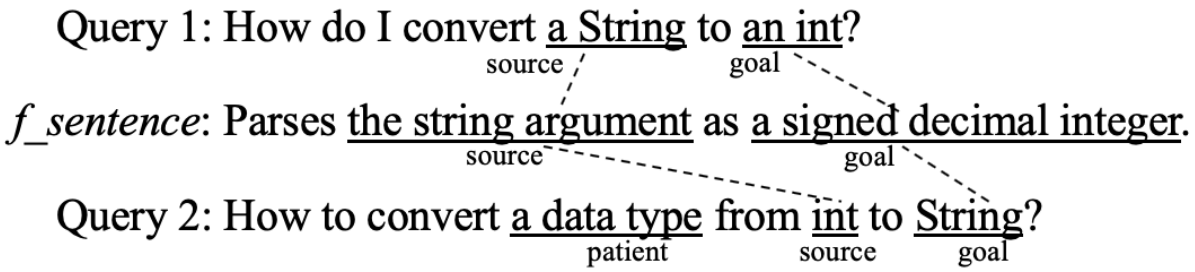
$$Score\ (Q, FS) = Sim_C(Q, FS) + Sim_R(Q, FS) + Sim_T(Q, FS) \quad (1)$$

4.3.1 Tính toán tính tương tự của danh mục chức năng. *F_category similarity SimC(Q, FS)* đo lường liệu f_categories của Q và FS có giống nhau hay không. Nếu Q

và FS được phân loại thành cùng một f_category, *SimC (Q, FS) = 1*; ngược lại, *SimC (Q, FS) = 0.*

4.3.2 Tính toán sự giống nhau về vai trò ngữ nghĩa. Sự tương tự về vai trò ngữ nghĩa SimR (Q, FS) đo lường sự giống nhau giữa các vai trò ngữ nghĩa tương ứng của đối sánh dựa trên thực thể Q và FS Sử dụng. Nếu Q và FS được phân loại thành f_categories khác nhau, *SimR (Q, FS) = 0*. Một cách khôn ngoan khác, chúng tôi tính toán *SimR (Q, FS)* theo ba bước, tức là liên kết thực thể, liên kết thực thể và đối sánh thực thể.

Đầu tiên, nhóm sắp xếp các thực thể tương ứng giữa Q và FS dựa trên vai trò ngữ nghĩa. Thực thể là một cụm danh từ trong Q hoặc FS. Một thực thể E_Q trong Q và một thực thể E_{FS} trong FS có thể được căn chỉnh nếu và chỉ khi chúng có cùng vai trò ngữ nghĩa trong Q và FS. Hình 6 cho thấy một ví dụ về sự căn chỉnh thực thể. Trong ví dụ này, "the string argument" trong f_sentence được căn chỉnh với "a String" trong Truy vấn 1 và "int" trong Truy vấn 2; "a signed decimal integer" trong f_sentence được căn chỉnh với "an int" trong Truy vấn 1 và "String" trong Truy vấn 2. Bằng cách này, có thể dễ dàng xác định rằng Truy vấn 1 phù hợp với f_sentence hơn Truy vấn 2. Nếu một thực thể E_Q trong Q có hai thực thể tương ứng E_{FS1} và E_{FS2} trong FS, E_Q được căn chỉnh với cả E_{FS1} và E_{FS2}, và ngược lại.



Hình 6: một ví dụ về sự căn chỉnh thực thể

Thứ hai, nhóm căn chỉnh các thực thể trong Q và FS với các thực thể tương ứng trong một biểu đồ kiến thức chung (Wikidata [42] trong tài liệu hiện tại). Dựa trên sự liên kết, chúng ta có thể sử dụng kiến thức trong đồ thị kiến thức chung để tính độ tương

đồng giữa các thực thể trong Q và FS. Ví dụ, Wikidata cung cấp kiến thức như "chuỗi là một chuỗi các ký tự và một kiểu dữ liệu" và "bí danh str isan của chuỗi". Để xem xét sự liên kết giữa một thực thể E_S trong Q hoặc FS và một thực thể E_W trong biểu đồ kiến thức chung, trước tiên chúng ta tiến hành tiền xử lý (mã hóa, dừng loại bỏ từ và bổ sung) trên cụm danh từ của E_S và sau đó tính hai điểm tương đồng sau đây giữa E_S và E_W: 1) sự giống nhau về hình thái có thể được đo lường dựa trên khoảng cách chỉnh sửa tối thiểu giữa cụm danh từ được xử lý trước của E_S và bất kì bí danh nào của E_W: 2) sự giống nhau về văn bản có thể được đo lường bằng sự giống nhau về văn bản giữa câu mà E_S xuất hiện trong (tức là Q hoặc FS) và các định nghĩa của E_W được cung cấp bởi biểu đồ kiến thức chung. Cuối cùng E_S được liên kết với một thực thể trong biểu đồ kiến thức chung có độ tương đồng kết hợp cao nhất với E_S.

Thứ ba, nhóm đối sánh giữa các thực thể được căn chỉnh dựa trên kết quả căn chỉnh thực thể. Đối với một thực thể E_Q trong Q và một thực thể căn chỉnh E_{FS} trong FS, sau đó tính điểm phù hợp của chúng theo cách sau: 1) nếu E_Q và E_{FS} được căn chỉnh với cùng một thực thể trong sơ đồ kiến thức chung (ví dụ: “string” và “char sequence ”), chúng bằng nhau và điểm phù hợp của chúng là 1; 2) nếu E_Q và E_{FS} Sare được căn chỉnh với hai thực thể có từ ghép nghĩa (ví dụ: “instance of”, “subclass of” hoặc “part of”) trong biểu đồ kiến thức chung (ví dụ: “int” và “primarytype”), điểm phù hợp là 1; 3) nếu một trong E_Q và E_{FS} là tiền tố tiếp đầu của cái còn lại (ví dụ: “Integer value” và “Integer”), chúng ở trong mối quan hệ từ ghép đôi và điểm phù hợp của chúng là 1; 4) nếu không, điểm phù hợp là 0.

Cuối cùng, Sim_R (Q, FS) được tính bằng cách tổng các điểm so khớp của tất cả các cặp thực thể được căn chỉnh giữa Q và FS. Lưu ý rằng nếu một thực thể E trong Q (hoặc FS) có một số thực thể được liên kết trong FS (hoặc Q), ta chỉ xem xét cặp thực thể có điểm phù hợp cao nhất.

4.3.3 Tính toán độ tương tự của văn bản. Sự tương tự về văn bản SimT (Q, FS) đo lường sự tương tự về văn bản tổng thể giữa Q và FS để so sánh với các phần câu khác nhau (ví dụ: mệnh đề, V-ing và động từ nguyên thể có to). Tương tự được tính toán dựa

trên mô hình Word2Vec [33]. Nhóm sử dụng mô hình Word2Vec 100 chiều trên Wikipediacorpus [12] và điều chỉnh mô hình dựa trên kho dữ liệu của tất cả f_sentences được rút ra trong nghiên cứu thực nghiệm bằng cách sử dụng gensim [5]. Sau đó, nhóm tính toán sự giống nhau theo cách sau: 1) tiền xử lý Q và FS bằng cách mã hóa, dừng loại bỏ từ và lemmatization; 2) vectơ chung cho Q và FS tương ứng bằng cách lấy trung bình các vectơ của tất cả các từ của chúng được tạo ra bởi mô hình Word2Vec; 3) tính toán độ tương tự cosin chuẩn hóa giữa các vectơ của Q và FS.

4.3.4 Tạo Kết quả khớp. Với một truy vấn, nhóm xếp hạng các f_sentences theo điểm phù hợp của chúng từ cao đến thấp. Đối với mỗi f_sentence, nhóm tạo giải thích ngôn ngữ cho việc mô tả bằng cách đối sánh: 1) danh mục f_category mà truy vấn và f_sentence thuộc về, 2) vai trò ngữ nghĩa trong truy vấn và f_sentence, và 3) tất cả các cặp thực thể phù hợp và mức độ phù hợp của chúng . Ví dụ: một kết quả phù hợp cho truy vấn "How do you crash a JVM?" sẽ là “Terminates the currently running Java Virtual Machine” của java.lang.System.exit (int). Nhóm có thể giải thích sự phù hợp này như sau: cả hai đều thuộc về f_category “stop/quit/terminate”; “JVM” đối sánh với “Java Virtual Machine” ở cấp độ “Equal” và chúng chia sẻ vai trò ngữ nghĩa “patient”.

5. Đánh giá

Đánh giá gồm hai phần. Phần thứ nhất đánh giá độ chính xác của phân tích cú pháp f_sentence, bao gồm phân loại f_category và phân tích p_pattern. Phần thứ hai đánh giá hiệu suất của PreMA trong các tác vụ truy xuất API dựa trên tài liệu bằng cách so sánh nó với phương pháp tiếp cận dựa trên học sâu.

5.1 Độ chính xác của chức năng phân tích cú pháp câu

Để đánh giá độ chính xác của phân loại f_category, nhóm đã sử dụng 14.774 f_sentences được chú thích trong nghiên cứu thực nghiệm để thực hiện kỹ thuật 5-fold cross validation. Độ chính xác trung bình của nghiên cứu là 92,8% (độ chính xác tối

thiểu là 92,6%). Phân tích cho thấy rằng hầu hết các f_sentences bị phân loại sai thường sử dụng các động từ ít gặp như “pin” và “compile”.

Để đánh giá độ chính xác của phân tích p_pattern, nhóm đã xóa những f_category “không xác định” khỏi 14.774 f_sentences, dữ liệu còn lại 11.074 f_sentences và chú thích tập dữ liệu này là p_pattern. Đối với mỗi f_sentence,nhóm đã sử dụng cách tiếp cận của mình để xác định p_pattern và so sánh nó với chú thích. Kết quả cho thấy độ chính xác của phân tích p_pattern là 90,4%. Phân tích cho thấy rằng hầu hết các sai lầm là do việc phân loại từ và phân tích cú pháp. Ví dụ: danh động từ đôi khi được nhận dạng là cụm danh từ (ví dụ: “Starts looping playback from the current position”) và "to" trong một động từ nguyên thể đôi khi được nhận dạng là giới từ (ví dụ: “Marshals to output the value in the Holder”).

Đánh giá trên dựa trên f_sentences được trích xuất từ tài liệu tham khảo JDK và Android. Để xác nhận liệu trình phân loại và phân tích có thể được áp dụng cho các thư viện khác hay không, nhóm đánh giá thêm tính chính xác của việc phân tích cú pháp f_sentence trên Apache POI (một thư viện Java để xử lý tài liệu Microsoft Office) [3]. Nhóm chọn ngẫu nhiên 100 f_sentences từ tài liệu tham khảo POI và mời ba sinh viên MS chú thích f_categories và p_patterns của họ theo cách tương tự như nghiên cứu thực nghiệm. Quá trình chú thích không tạo ra f_categories mới. Nhóm đã sử dụng tất cả 14.774 f_sentences được chú thích trong nghiên cứu thực nghiệm để đào tạo một bộ phân loại f_category và sử dụng nó để phân loại 100 câu POI. Độ chính xác phân loại f_category trên các câu POI là 97%. Độ chính xác phân loại f_category trên các câu POI là 97%. Nhóm tiếp tục sử dụng PreMA để phân tích p_patterns của 100 câu POI và độ chính xác là 95%. Kết quả cho thấy trình phân loại f_category và trình phân tích p_pattern được đào tạo trên tài liệu tham khảo JDK và Android cũng hoạt động tốt cho POI

Bảng 3 cho thấy kết quả phân tích cú pháp câu chức năng do PreMA tạo ra, trong đó các từ in nghiêng đậm và các chỉ số con trong f_sentences biểu thị cấu trúc và vai trò ngữ nghĩa (mệnh đề) của p_patterns. Chúng ta có thể thấy rằng các f_sentences có

cùng một f_verbs (ví dụ: “return”) có thể được phân loại thành các f_categories khác nhau. Các p_patterns của cùng một f_categories có thể có số lượng vai trò ngữ nghĩa khác nhau, ví dụ như f_category “convert/transform/turn” có cả p_patterns với 2 và 3 vai trò ngữ nghĩa. Dựa trên các p_patterns được công nhận, những người tham gia các f_sentences khác nhau của cùng một f_categories có thể được căn chỉnh dựa trên các vai trò ngữ nghĩa (ví dụ: source, goal) và các mệnh đề.

Bảng 3: Ví dụ về phân tích cú pháp câu chức năng

API Method	*f_sentence*	*f_category*	Library
javax.xml.transform.Transformer.transform()	***Transform*** {the XML Source}$_s$ ***to*** {a Result}$_g$.	convert/transform/turn	JDK
android.graphics.Color.HSVToColor()	***Convert*** {HSV components}$_s$ ***to*** {an ARGB color}$_g$.	convert/transform/turn	Android
org.apache.poi.hpsf.PropertySet.toInputStream()	***Returns*** {the contents of this property set stream}$_s$ **as** {an input stream}$_g$.	convert/transform/turn	POI
android.icu.util.UniversalTimeScale.from()	***Convert*** {a long datetime}$_p$ ***from*** {the given time scale}$_s$ ***to*** {the universal time scale}$_g$.	convert/transform/turn	Android
java.text.CharacterIterator.getBeginIndex()	***Returns*** {the start index of the text}$_p$.	get/return/obtain	JDK
java.sql.Statement.getResultSetType()	***Retrieves*** {the result set type}$_p$ ***for*** {ResultSet objects generated by this Statement object}$_b$.	get/return/obtain	JDK
org.apache.poi.ss.formula.eval.AreaEval.containsColumn()	***Returns*** {true}$_p$ ***if*** {the specified col is in range}$_c$.	check/test/determine	POI
org.apache.poi.xssf.usermodel.XSSFWorkbook.isSheetHidden()	***Check whether*** {a sheet is very hidden}$_c$.	check/test/determine	POI

Participants of *p_patterns*: s (source), g (goal), p (patient), b (beneficiary), c (clause).

5.2 Hiệu suất của phương pháp truy xuất API

Nhóm tác giả đã sử dụng phương pháp học sâu(Deep Learning) làm công cụ nền tảng để truy vấn API method. Công cụ này sử dụng Word2Vec [33] để biểu diễn vector của các từ và câu. Khi người dùng thực hiện truy vấn, công cụ sẽ tìm một API phù hợp nhất với truy vấn đó mà không cần phân tích cú pháp câu. Nó sử dụng mô hình Word2Vec 100 chiều được tạo từ trước trên kho dữ liệu Wikipedia [12] và điều chỉnh mô hình dựa trên kho dữ liệu f_sentences bằng cách sử dụng gensim [5]. Nó tạo ra một vectơ cho truy vấn của người dùng bằng cách lấy trung bình các vectơ của tất cả các từ trong câu truy vấn sau khi xử lý (tức là mã hóa, dừng loại bỏ từ và lemmatization) và một vectơ cho mô tả API theo cách tương tự. Cuối cùng, nó tính toán sự giống nhau về cosin giữa vectơ của truy vấn và vectơ của mô tả của từng phương thức API và xếp hạng các phương thức API ứng viên theo độ tương tự. Nhóm đã thử và kiểm tra hai chiến lược để đối sánh API (sử dụng mô tả đầy đủ của từng phương thức API hoặc chỉ câu đầu tiên của mô tả), dựa trên tập dữ liệu được xây dựng thủ công gồm các cặp

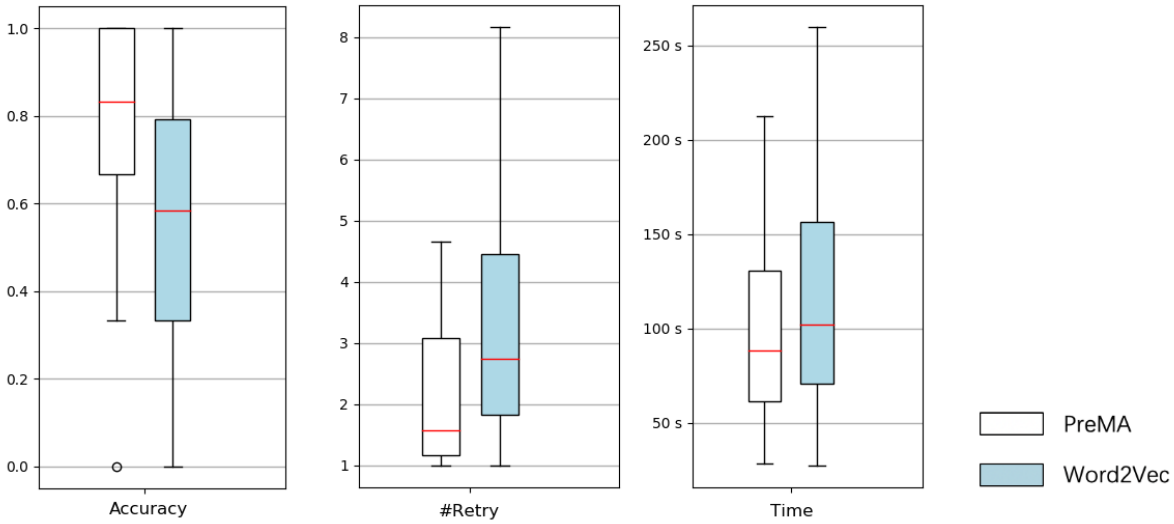
phương thức API và truy vấn của người dùng. Kết quả cho thấy việc triển khai sử dụng câu đầu tiên của mỗi mô tả có hiệu suất tốt hơn, do đó nhóm chọn nó làm đường cơ sở. Nhóm không coi cách tiếp cận dựa trên BERT là cơ sở, vì nhóm cần đào tạo một bộ phân loại nhị phân về mức độ liên quan của tài liệu truy vấn dựa trên mô hình BERT đã được tinh chỉnh và dữ liệu đào tạo bổ sung của các cặp tài liệu truy vấn có liên quan / không liên quan.

Nhóm đã chọn các câu hỏi truy xuất API trên Stack Overflow được gắn thẻ "Java" hoặc "Android" dựa trên các tiêu chí sau: các câu hỏi yêu cầu API triển khai các chức năng cụ thể và có ít nhất một câu trả lời được chấp nhận đề xuất API JDK hoặc Android . Sau đó sắp xếp các câu hỏi trên dựa theo phiếu bầu nhận được và chọn 30 câu hỏi được xếp hạng cao. Đối với mỗi câu hỏi đã chọn, nhóm đã tạo một nhiệm vụ truy xuất API sử dụng tiêu đề và nội dung câu hỏi làm mô tả nhiệm vụ.

Nhóm mời hai tiến sĩ và mười sinh viên MS, những người quen thuộc với phát triển Java và Android để hoàn thành 30 nhiệm vụ. Đối với mỗi nhiệm vụ, họ có thể xây dựng và thử các truy vấn khác nhau dựa trên sự hiểu biết của họ về mô tả nhiệm vụ. Cả PreMA và phương pháp cơ sở đều trả về 10 phương thức API hàng đầu cùng với mô tả lớp của chúng làm bối cảnh để người dùng chọn. Những người tham gia hoàn thành một nhiệm vụ khi họ tìm thấy một phương thức API phù hợp với mô tả nhiệm vụ. Họ được chia thành hai nhóm (G1 và G2) dựa trên một cuộc khảo sát trước khi thử nghiệm về kinh nghiệm lập trình của họ, cân bằng kinh nghiệm trong cả hai nhóm. 30 nhiệm vụ được chia ngẫu nhiên thành hai nhóm (T1 và T2). Thí nghiệm được tiến hành trong hai giai đoạn. Trong giai đoạn đầu, những người tham gia G1 và G2 được yêu cầu hoàn thành các nhiệm vụ trong T1 với PreMA và công cụ cơ sở tương ứng. Trong giai đoạn hai, hai nhóm trao đổi công cụ để hoàn thành nhiệm vụ trong T2. Tất cả những người tham gia được yêu cầu chạy trình ghi toàn màn hình để ghi lại các quy trình truy xuất API của họ.

Đối với mỗi nhiệm vụ, nhóm ghi lại độ chính xác, số lần thử lại, thời gian hoàn thành của mỗi người tham gia, tính toán độ chính xác (tức là tỷ lệ người tham gia đã

chọn đúng API cho nhiệm vụ), số lần thử lại trung bình và thời gian hoàn thành của hai người tham gia các nhóm. Hình 7 cho thấy hiệu suất của các nhóm sử dụng PreMA và công cụ cơ sở thực hiện 30 nhiệm vụ. Những người tham gia sử dụng PreMA đã hoàn thành các nhiệm vụ chính xác hơn, yêu cầu ít lần thử lại hơn và sử dụng ít thời gian hơn những người sử dụng công cụ cơ sở. Trung bình, độ chính xác, số lần nghỉ việc và thời gian hoàn thành (tính theo giây) của những người tham gia sử dụng PreMA và công cụ cơ sở lần lượt là 0,77, 2,16, 98,29 so với 0,54, 3,30, 113,42. Nhóm đã sử dụng Welch's T-test để xác minh ý nghĩa thống kê của sự khác biệt và độ chính xác, số lần nghỉ việc và thời gian lần lượt là 0,0032, 0,0064 và 0,2832. Chúng ta có thể thấy sự khác biệt về độ chính xác và số lần việc là có ý nghĩa thống kê ($p << 0,05$), trong khi sự khác biệt về thời gian là không có ý nghĩa.



Hình 7: Hiệu suất của việc truy xuất API

Việc phân tích các bản ghi màn hình cho thấy rằng công cụ của nhóm đã hoạt động đặc biệt tốt cho các nhiệm vụ nhạy cảm với xếp hạng. Ví dụ: đối với nhiệm vụ "converting array to list in Java", người tham gia có thể nhanh chóng tìm thấy các API chính xác bằng công cụ của nhóm trong khi công cụ cơ sở trả về kết quả bao gồm các API không chính xác (biến danh sách thành mảng - thứ tự ngược lại). Đối với các nhiệm vụ liên quan đến các khái niệm software domain, cách tiếp cận của nhóm cũng có thể đề xuất kết quả tốt hơn. Ví dụ: đối với nhiệm vụ "how do you crash a jvm", công cụ của

chúng tôi có thể đề xuất chính xác các API java.lang.System.exit(int) và java.lang.Runtime.halt(int) trong khi kết quả được trả về bởi công cụ cơ sở không liên quan đến "jvm". Nhóm cũng nhận thấy f_categories rất hữu ích để giải quyết các công việc. Ví dụ: đối với nhiệm vụ “check whether a string is not null and not empty”, f_sentences được đề xuất bởi công cụ của chúng tôi nằm trong cùng một f_category “check/test/determine” như câu hỏi. Tuy nhiên, f_sentences được đề xuất bởi công cụ cơ sở có xu hướng chứa từ khóa “null” và động từ được sử dụng trong câu hỏi bị bỏ qua.

Một vấn đề cần quan tâm là cách tiếp cận này có thể không tìm kiếm được các API sử dụng các động từ không phổ biến. Một số động từ không phổ biến thể hiện ý nghĩa miền cụ thể (ví dụ: “mute”, “roam”, “dial”) và có thể dễ dàng tự phân biệt. Một số động từ không phổ biến khác có thể được phân loại cùng với các động từ phổ biến hơn và do đó cũng có thể phân biệt. Trong số 30 tác vụ, có 3 nhiệm vụ có các API tương ứng (java.util.Collections.sort(), java.lang.System.exit(), java.lang.String.split()) sử dụng các động từ không phổ biến (“sort”, “terminate”, “split”). Những động từ này đã được bao gồm trong các f_categories đã được xác định, ví dụ “terminate” được bao gồm trong một f_category với các động từ phổ biến “stop” and “end”.

Từ việc phỏng vấn những người tham gia sau thử nghiệm, họ nhận thấy công cụ của mình cung cấp câu trả lời chính xác và phù hợp dễ dàng hơn trong khi công cụ cơ sở yêu cầu nhiều truy vấn hơn. Một người tham gia nói rằng công cụ của nhóm có thể hiểu các truy vấn tốt hơn công cụ cơ sở. Ví dụ: khi người tham gia nhập một truy vấn có từ “check”, công cụ có thể đề xuất các API có f_sentences chứa các động từ như “test” and “determine”. Những người tham gia cũng đề xuất các cải tiến cho công cụ, chẳng hạn như cần xếp hạng kết quả tốt hơn và hiển thị tất cả các biến thể của hàm nạp chồng.

6.Các mối đe dọa đến tính hợp lý (THREATS TO VALIDITY)

6.1.Tính hợp lý nội tại (Internal validity)

Một mối đe dọa tiềm ẩn đối với tính hợp lý nội tại bắt nguồn từ việc sử dụng thư viện xử lý ngôn ngữ tự nhiên, SpaCy. Một số phân tích của chúng tôi dựa trên quá trình xử lý câu bằng ngôn ngữ tự nhiên của SpaCy (ví dụ: RQ1 và RQ3). Không có thư viện xử lý ngôn ngữ tự nhiên nào đạt được độ chính xác 100% trên bất kỳ tập dữ liệu lớn nào và hiệu suất của SpaCy được đánh giá là ngang bằng với hiện đại [17] và vượt trội hơn các thư viện khác khi áp dụng cho tài liệu phần mềm [13]. SpaCy không được thiết kế đặc biệt cho văn bản phần mềm, tức là văn bản chứa các phần tử mã, câu chưa hoàn chỉnh hoặc lỗi ngữ pháp thường gặp trên Stack Overflow. Hiện tại không có công cụ xử lý ngôn ngữ tự nhiên nào chuyên dùng để phân tích văn bản liên quan đến phát triển phần mềm và chúng ta phải dựa vào các công cụ xử lý ngôn ngữ tự nhiên có mục đích chung. Để giảm thiểu mối đe dọa này, chúng tôi sử dụng các quy tắc heuristic (dựa trên kinh nghiệm) để sửa một số lỗi phổ biến. Quá trình này tương tự như phần công việc liên quan [40].

Một mối đe dọa khác có thể phát sinh từ quy mô của mã hóa mở (open coding). Để phân tích f_categories trong RQ2, chúng tôi chỉ mã hóa f_sentences của 14.733 phương thức JDK và Android API, không phải tất cả 54.256 f_sentences. Một mối quan tâm là f_verbs và f_categories mà chúng tôi đã xác định có thể không bao gồm tất cả các f_verbs và f_categories trong JDK và Android. Mã hóa mở của toàn bộ tập hợp vượt quá khả năng của chúng tôi và chúng tôi cố gắng hết sức để bao gồm nhiều f_verbs và f_categories nhất có thể bằng chiến lược lấy mẫu của mình. Dựa trên những phát hiện của RQ1, 116 động từ thường gặp nhất xuất hiện trong khoảng 84% f_sentences. Do đó, trong giai đoạn mã hóa ban đầu, chúng tôi lấy mẫu 10 câu cho mỗi 116 động từ thường gặp nhất để bao gồm nhiều f_verbs và f_categories phổ biến nhất có thể với kích thước lấy mẫu tương đối nhỏ (1.139). Trong giai đoạn thứ hai, nhóm tạo một mẫu ngẫu

nhiên lớn hơn (13,635) để bao gồm các f_verbs và f_categories không phổ biến không được bao gồm trong mẫu đầu tiên.

Một mối đe dọa bổ sung liên quan đến chất lượng của mã hóa mở. Nhóm giảm thiểu điều này bằng cách tách hai giai đoạn mã hóa và đào tạo tất cả các lập trình viên trước khi viết mã. Nhóm báo cáo Cohen's Kappa cho tất cả các mã mở để cung cấp bằng chứng cho thấy kết quả mã hóa của chúng tôi là đáng tin cậy.

6.2.Tính hợp lý ngoại tại (External validity)

Mối quan tâm chính là mức độ tổng quát hóa các công cụ phát hiện tự động. Nhóm tác giả cung cấp bằng chứng về khả năng tổng quát hóa của chúng bằng cách đánh giá chúng trên JDK và Android f_sentences, câu hỏi Stack Overflow và POI f_sentences. Các f_categories và f_patterns mới có thể cần được tiết lộ bằng cách sử dụng quy trình nghiên cứu thực nghiệm tương tự khi sử dụng cách tiếp cận của nhóm cho các thư viện khác.

7. Công việc liên quan

7.1. Kiến thức về chức năng

Chức năng là một loại kiến thức quan trọng cần thiết cho các nhiệm vụ phát triển phần mềm như triển khai và bảo trì các tính năng. Kirk và cộng sự[30] đã nghiên cứu "các vấn đề tái sử dụng" phải đối mặt khi phát triển các ứng dụng dựa trên một framework và xác định bốn loại chính của các vấn đề tái sử dụng framework - "Chức năng" là một trong số đó. Erdos và Sneed [19] đã xác định bảy câu hỏi mà các nhà phát triển cần hỏi trong các nhiệm vụ bảo trì phần mềm. Tất cả các câu hỏi là về việc hiểu hành vi của chương trình và do đó về kiến thức chức năng.

Công việc liên quan khác hướng đến các mô tả chức năng trong tài liệu phần mềm. Maalej và Robillard [31] đã báo cáo về một nghiên cứu về các mẫu kiến thức trong tài liệu API, chẳng hạn như Chức năng, Khái niệm và Hướng dẫn. Các tác giả nhận thấy

rằng chức năng chiếm một phần lớn trong tài liệu API, nhưng họ không cung cấp phân loại thêm về các mô tả chức năng trong tài liệu API. Dựa trên kết quả của Maalej và Robillard, Fucci và cộng sự. [20] đã cố gắng sử dụng các kỹ thuật học máy để phân loại các loại kiến thức của câu trong tài liệu API. Họ báo cáo rằng các loại kiến thức thường gặp nhất là Chức năng và Phi Thông tin. Trong công việc của mình, nhóm đã tập trung vào loại kiến thức phổ biến nhất — Chức năng — và phân loại và phân tích các mô tả chức năng API dựa trên các động từ chức năng.

7.2.Cụm động từ trong kỹ thuật phần mềm

Trong nhiều ngôn ngữ lập trình, tên phương thức được sử dụng để mô tả việc thực hiện một phương thức ở mức cao. Nghiên cứu trước đây đã phát hiện ra rằng mã nguồn sẽ dễ đọc hơn nếu mọi phương pháp đều có tên thích hợp [25]. Vì tên phương thức thường bao gồm các cụm động từ, Høst và Østvold [24] đã xây dựng một từ vựng chứa các động từ thường được sử dụng trong tên phương thức Java và các đặc điểm được báo cáo của tên phương thức dựa trên động từ của chúng. Hayase và cộng sự [22] đã xây dựng một từ điển theo miền cụ thể về quan hệ động từ-đối tượng từ các định danh xuất hiện trong các tệp mã nguồn. Kashiwabara và cộng sự [29] tập trung vào việc đề xuất các động từ tương tự cho tên phương thức để các nhà phát triển có thể sử dụng các động từ nhất quán cho tên phương thức. Tuy nhiên, họ tập trung vào tên phương thức thay vì mô tả phương thức bằng ngôn ngữ tự nhiên.

Shepherd và cộng sự [38] đề xuất một cách tiếp cận để mở rộng truy vấn và tìm kiếm mã. Phương pháp này sử dụng các cặp <động từ, đối tượng trực tiếp> (<verb, direct object>) (V-DO) từ các ký hiệu và chú thích của phương thức để tìm các hành động cắt ngang các hệ thống hướng đối tượng. Hill và cộng sự [23] đề xuất một cách tiếp cận để tự động trích xuất và tạo ra các cụm danh từ, động từ và giới từ từ các ký hiệu phương thức và trường, nắm bắt ngữ cảnh từ của các truy vấn ngôn ngữ tự nhiên để duy trì và sử dụng lại.

Treude và cộng sự [40] tập trung vào mô tả ngôn ngữ tự nhiên và các cụm từ nhiệm vụ phát triển được trích xuất từ tài liệu phần mềm. Tuy nhiên, họ đã trích xuất tất cả các cụm từ nhiệm vụ từ các câu. Trong công việc của họ, một câu có thể chứa nhiều hơn một cụm từ nhiệm vụ và họ đã không phân biệt chúng dựa trên mức độ quan trọng. Ngoài ra, họ chỉ sử dụng một tập hợp nhỏ các động từ xác định trước để xác định các cụm từ nhiệm vụ và không xem xét các từ đồng nghĩa một cách có hệ thống. Trong công việc này, nhóm tập trung vào mô tả chức năng của các phương thức API Java và Android từ tài liệu tham khảo API và phân loại các động từ chức năng thành các danh mục chức năng, do đó cung cấp một cách có hệ thống để khám phá các từ đồng nghĩa trong danh mục chức năng.

7.3.Đề xuất API

Các phương pháp tiếp cận đề xuất API hiện tại thường sử dụng thông tin ngữ cảnh để đề xuất các API, ví dụ: đồ thị phụ thuộc API [16], lịch sử yêu cầu tính năng [39], các trang web và tài liệu hỏi đáp [27, 37]. Các phương pháp tiếp cận hiện tại không chỉ có thể đề xuất các phương pháp và lớp API từ các thư viện của bên thứ ba [27, 37] mà còn hỗ trợ các API dành riêng cho dự án [43]

Rahman và cộng sự [37] đề xuất một cách tiếp cận gọi là RACK và cũng xây dựng một kho tài liệu để ánh xạ các từ khóa từ câu hỏi Stack Overflow đến tài liệu API. Dựa trên kho tài liệu này, RACK có thể đề xuất các API cho một truy vấn nhất định. Huang và cộng sự. [27] đã kết hợp kiến thức về Stack Overflow với tài liệu API và BIKER được đề xuất, cũng có thể đề xuất API cho một truy vấn nhất định. Tuy nhiên, những cách tiếp cận này chỉ có thể hoạt động đối với các API đã được thảo luận nhiều trên các trang web như Stack Overflow và bị nhiễu thông tin trong các tài nguyên bên ngoài này. Nghiên cứu trước đây đã chỉ ra rằng Stack Overflow có xu hướng chậm bao phủ các API mới [36] và có thể bỏ qua các phần quan trọng của API. Ngược lại, cách tiếp cận của nhóm không dựa vào các nguồn lực bên ngoài.

Cách tiếp cận của Hill cùng cộng sự [23] có thể tự động phân loại các cụm từ được trích xuất thành một hệ thống phân cấp dựa trên đối sánh từng phần của cụm từ, để giúp người bảo trì phần mềm nhanh chóng phân biệt giữa các kết quả tìm kiếm có liên quan và không liên quan và định dạng lại các truy vấn. Tuy nhiên, cách tiếp cận của họ không thể giải quyết vấn đề về khoảng cách từ vựng giữa truy vấn và tài liệu.

Các cách tiếp cận khác trong lĩnh vực đề xuất API không tập trung vào việc đề xuất các phương pháp, mà thay vào đó cho các ví dụ trên các đoạn mã [15, 34, 35, 45] hoặc các tham số [14].

8. Kết luận

Trong bài báo này, nhóm tác giả đã tiến hành một nghiên cứu thực nghiệm quy mô lớn về các mô tả chức năng của 14.733 phương thức JDK và Android API. Nhóm đã xác định 356 động từ chức năng khác nhau từ các mô tả và các động từ này có thể được nhóm thành 87 loại chức năng dựa trên ngữ nghĩa của chúng trong ngữ cảnh mô tả. Nhóm cũng trích xuất 523 mẫu cụm từ từ các cụm động từ của các mô tả. Dựa trên những phát hiện này, Nhóm đề xuất phương pháp tiếp cận đề xuất phương thức API dựa trên sự so sánh của các cụm động từ chức năng trong mô tả chức năng và truy vấn của người dùng, được gọi là PreMA. Nhóm đã tiến hành các nghiên cứu thử nghiệm để đánh giá độ chính xác của phân tích chức năng và hiệu suất truy xuất API của PreMA. Kết quả cho thấy PreMA có thể nhận dạng chính xác các danh mục chức năng (92,8%) và các mẫu cụm từ (90,4%) của các câu mô tả chức năng; và những người tham gia sử dụng PreMA đã hoàn thành nhiệm vụ của họ chính xác hơn (0,77 so với 0,54) với ít lần thử lại hơn (2,16 so với 3,30) và sử dụng ít thời gian hơn (98,29 giây so với 113,42 giây).

Công việc trong tương lai sẽ được dành để áp dụng các cách tiếp cận để tự động nhận dạng các danh mục chức năng và các động từ chức năng liên quan và các mẫu cụm từ cho các vấn đề kỹ thuật phần mềm khác, chẳng hạn như chất lượng tài liệu và truy xuất thông tin. Ngoài ra, nhóm sẽ cải thiện hơn nữa khả năng phân tích ngữ cảnh (ví dụ:

bằng cách xem xét mô tả lớp và mô tả phương thức ngoài chức năng) PreMA để đạt được đối sánh API chính xác hơn. Tất cả dữ liệu từ công việc này sẽ được chuyển thành dữ liệu mở lưu trữ sau khi được chấp nhận.

9. Dựng lại nghiên cứu

9.1. Thư viện sử dụng

Nghiên cứu thực nghiệm của nhóm tác giả cùng cấp 87 f_categories và 14.774 f_sentences đã được chú thích để đào tạo trình phân loại tự động. Để dựng lại bài nghiên cứu nhóm đã sử dụng thư viện funcverbnet của nghiên cứu và thư viện Spacy bằng ngôn ngữ python.

Thư viện funcverbnet cung cấp hệ thống dữ liệu đã được xây dựng từ nghiên cứu thực nghiệm, gồm các danh mục chức năng (f_category), động từ(f_verb), mẫu cụm từ (p_pattern) và các hàm có sẵn để phân tích chi tiết các f_description dựa trên hệ thống dữ liệu đó. Funverbnet sử dụng fasttext để học vector từ giúp phân loại f_sentence vào f_category

Thư viện Spacy là một thư viện mã nguồn mở miễn phí dùng để xử lý ngôn ngữ tự nhiên (NLP) bằng python. Bản dựng nghiên cứu này sử dụng chức năng chú thích ngôn ngữ của Spacy để phân tích cấu trúc ngữ pháp của câu.

9.2. Cách dựng

Dựa trên bài báo nghiên cứu, Nhóm đã dựng lại phần xác định động từ chức năng (f_verb) của câu. Sau đây là mô tả cách thức hoạt động của bản dựng:

Sử dụng Thư viện funverbnet (hệ thống dữ liệu đã phân tích và các hàm) của nghiên cứu để phân loại câu chức năng f_sentency (hoặc câu truy vấn api của người dùng) vào f_category (Hình 9.1 hàm find_f_category)

Từ f_category tìm được, xác định được tập verb thuộc về category. Sử dụng Spacy để phân tích ngữ pháp câu:

Phân đoạn câu thành từ, dấu câu, …

Gán nhãn POS và tìm cây phụ thuộc của các từ

Dựa vào nhãn POS => duyệt cây phụ thuộc, động từ đầu tiên tìm thấy thuộc tập verb của category sẽ được chọn làm f_verb. Trường hợp không tìm thấy động từ nào trong câu thuộc tập verb của category thì động từ đầu tiên tìm thấy được chọn làm f_verb. Hình 5 của bài nghiên cứu mô tả cách xác định f_verb (Hình 9.1 hàm find_f_verb)

Dựng đối tương PreMA có các thuộc tính, f_sentency, f_category, f_verb, p_pattern.

```python
class PreMA:
    def __init__(self, f_sentence):
        self._f_sentence = sentence
        self._f_category = self.find_f_category()
        self._f_verb, self._vector_doc = self.find_f_verb()
        self._p_pattern = self.find_p_pattern()

    def find_f_category(self):
        category_id = FUNC_VERB_NET.find_category_by_any_sentence(self._f_sentence)
        cate = FUNC_VERB_NET.find_cate_by_id(category_id)
        return cate

    def find_f_verb(self):
        if self._f_category.id == -1:
            return None
        nlp = spacy.load("en_core_web_sm")
        doc = nlp(self._f_sentence)
        f_verb = None
        include_verb = self._f_category.included_verb
        for token in doc:
            if token.tag_ == 'VB' and token.text in include_verb:
                f_verb = token.text
                break
        if f_verb is None:
            for token in doc:
                if token.tag_ == 'VB':
                    f_verb = token.text
                    break
        return f_verb, doc
```

Hình 9.1 Tạo đối tượng PreMA mô phỏng

9.3. Kết quả bản dựng

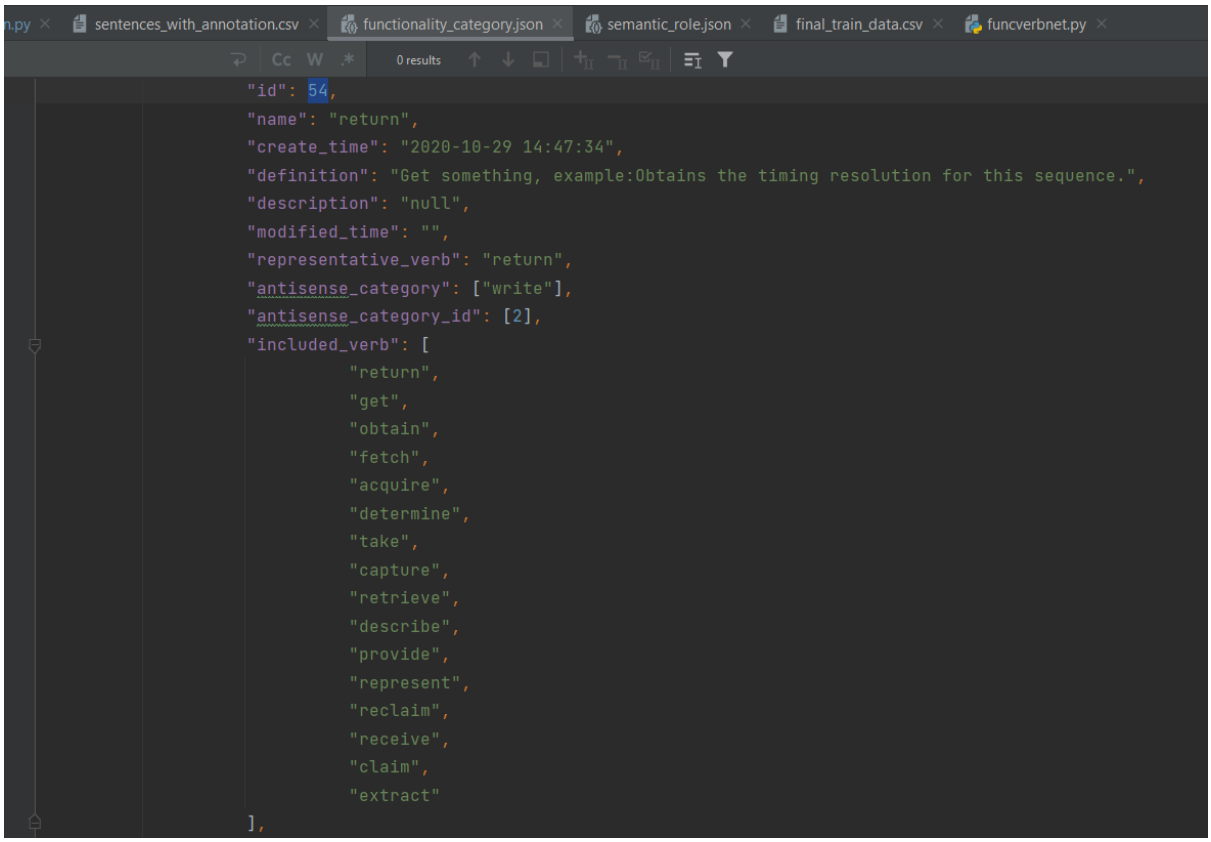
link git bản dựng:

https://github.com/nguyen-chi123/APIMethodRecommendation.git

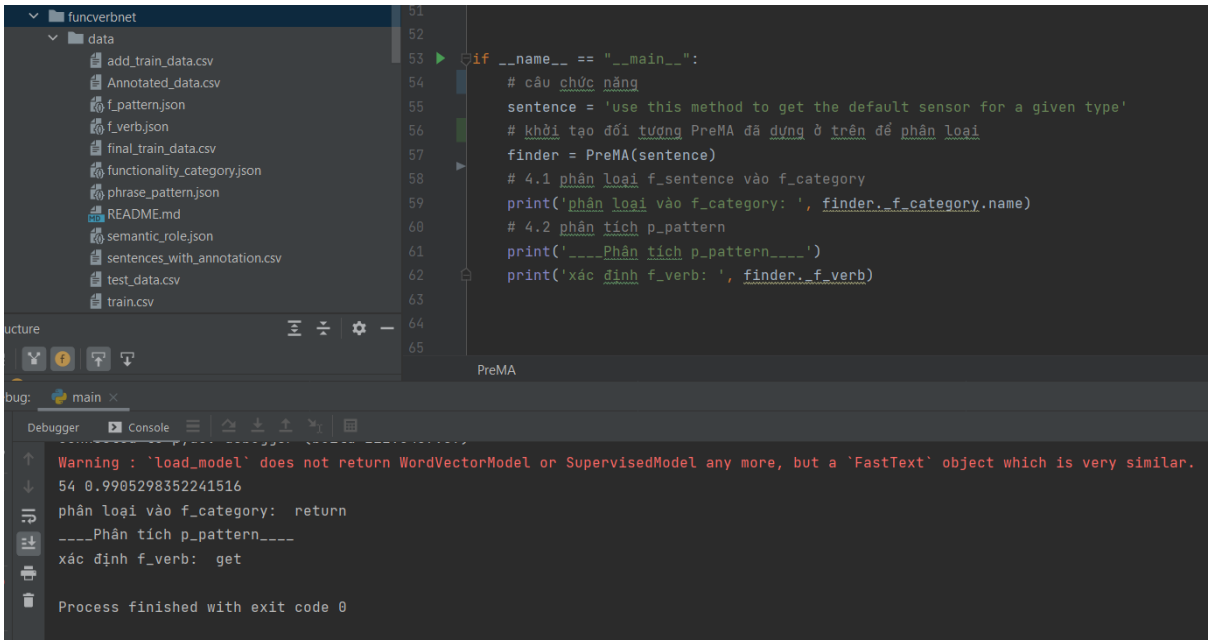
Ví dụ:

với câu chức năng: “use this method to get the default sensor for a given type”

lớp PreMA đã xác định được f_category: id = 54, name = “return” với độ chính xác khoảng 0.9905298352241516, tìm ra f_verb = “get” (Hình 9.2 và Hình 9.3)



Hình 9.2 category “return” chứa động từ “get”



Hình 9.3 kết quả chạy bản dựng

10.Tài liệu tham khảo

[1] 2020. *Android 27 Reference Documentation*. Retrieved September 10, 2020 from https://developer.android.com/reference/packages

[2] 2020. *Annotation Tool GUI*. Retrieved September 10, 2020 from https://i.loli.net/2019/05/12/5cd7d3ed9a2e6.png

[3] 2020. *Apache POI*. Retrieved September 10, 2020 from https://poi.apache.org/

[4] 2020. *BeautifulSoup*. Retrieved September 10, 2020 from https://www.crummy.com/software/BeautifulSoup/

[5] 2020. *gensim*. Retrieved September 10, 2020 from https://radimrehurek.com/gensim/

[6] 2020. *JDK 1.8 Reference Documentation*. Retrieved September 10, 2020 from https://docs.oracle.com/javase/8/docs/api/overview-summary.html

[7] 2020. *Replication Package*. Retrieved September 9, 2020 from https://fudanselab.github.io/Research-FSE2020-FuncVerb/

[8] 2020. *SpaCy*. Retrieved September 10, 2020 from https://spacy.io

[9] 2020. *Stack Overflow Question 65200*. Retrieved September 10, 2020 from https://stackoverflow.com/questions/65200/

[10] 2020. *VerbNet*. Retrieved September 10, 2020 from http://verbs.colorado.edu/~mpalmer/projects/verbnet.html

[11] 2020. *VerbNet Annotation Guidelines*. Retrieved September 10, 2020 from https://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf

[12] 2020. *word2vec-api*. Retrieved September 10, 2020 from https://github.com/3Top/word2vec-api

[13] Fouad Nasser A Al Omran and Christoph Treude. 2017. Choosing an NLP Library for Analyzing Software Documentation: A Systematic Literature Review and a Series of Experiments. In *Proceedings of the International Conference on Mining Software Repositories*. 187–197.

[14] Muhammad Asaduzzaman, Chanchal K. Roy, Samiul Monir, and Kevin A. Schneider. 2015. Exploring API method parameter recommendations. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, Rainer Koschke, Jens Krinke, and Martin P. Robillard (Eds.). IEEE Computer Society, 271–280.

[15] Brock Angus Campbell and Christoph Treude. 2017. NLP2Code: Code Snippet Content Assist via Natural Language Tasks. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*. IEEE Computer Society, 628–632.

[16] Wing-Kwan Chan, Hong Cheng, and David Lo. 2012. Searching connected API subgraph via text phrases. In *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012*, Will Tracz, Martin P. Robillard, and Tevfik Bultan (Eds.). ACM, 10.

[17] Jinho D. Choi, Joel R. Tetreault, and Amanda Stent. 2015. It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. 387–396.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018).

[19] Katalin Erdös and Harry M. Sneed. 1998. Partial Comprehension of Complex Programs (enough to perform maintenance). In *6th International Workshop on Program Comprehension (IWPC '98), June 24-26, 1998, Ischia, Italy*. IEEE Computer Society, 98–105.

[20] Davide Fucci, Alireza Mollaalizadehbahnemiri, and Walid Maalej. 2019. On using machine learning to identify knowledge in API reference documentation. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 109–119.

[21] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*. 631–642.

[22] Yasuhiro Hayase, Yu Kashima, Yuki Manabe, and Katsuro Inoue. 2011. Building Domain Specific Dictionaries of Verb-Object Relation from Source Code. In *15th European Conference on Software Maintenance and Reengineering, CSMR 2011, 1-4 March 2011, Oldenburg, Germany*. 93–100.

[23] Emily Hill, Lori L. Pollock, and K. Vijay-Shanker. 2009. Automatically capturing source code context of NL-queries for software maintenance and reuse. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*. IEEE, 232–242.

[24] Einar W. Høst and Bjarte M. Østvold. 2007. The Programmer's Lexicon, Volume I: The Verbs. In *Seventh IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2007), September 30 - October 1, 2007, Paris, France*. 193–202.

[25] Einar W. Høst and Bjarte M. Østvold. 2009. Debugging Method Names. In *ECOOP 2009 - Object-Oriented Programming, 23rd European Conference, Genoa, Italy, July 6-10, 2009. Proceedings*. 294–317.

[26] Einar W. Høst and Bjarte M. Østvold. 2010. Canonical Method Names for Java - Using Implementation Semantics to Identify Synonymous Verbs. In *Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*. 226–245.

[27] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. 293–304.

[28] Yuki Kashiwabara, Takashi Ishio, Hideaki Hata, and Katsuro Inoue. 2015. Method Verb Recommendation Using Association Rule Mining in a Set of Existing Projects. *IEICE Transactions* 98-D, 3 (2015), 627–636.

[29] Yuki Kashiwabara, Yuya Onizuka, Takashi Ishio, Yasuhiro Hayase, Tetsuo Yamamoto, and Katsuro Inoue. 2014. Recommending verbs for rename method using association rule mining. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*. 323–327.

[30] Douglas Samuel Kirk, Marc Roper, and Murray Wood. 2007. Identifying and addressing problems in object-oriented framework reuse. *Empirical Software Engineering* 12, 3 (2007), 243–274.

[31] Walid Maalej and Martin P. Robillard. 2013. Patterns of Knowledge in API Reference Documentation. *IEEE Trans. Software Eng.* 39, 9 (2013), 1264–1282.

[32] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica: Biochemia medica* 22, 3 (2012), 276–282.

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.* 3111–3119.

[34] Anh Nguyen, Peter C. Rigby, Thanh Van Nguyen, Dharani Palani, Mark Karanfil, and Tien N. Nguyen. 2018. Statistical Translation of English Texts to API Code Templates. In *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 194–205.

[35] Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. 2016. Query Expansion Based on Crowd Knowledge for Code Search. *IEEE Trans. Services Computing* 9, 5 (2016), 771–783.

[36] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. 2012. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. *Georgia Institute of Technology, Tech. Rep* 11 (2012).

[37] Mohammad Masudur Rahman, Chanchal Kumar Roy, and David Lo. 2016. RACK: Automatic API Recommendation Using Crowdsourced Knowledge. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. IEEE Computer Society, 349–359.

[38] David C. Shepherd, Zachary P. Fry, Emily Hill, Lori L. Pollock, and K. VijayShanker. 2007. Using natural language program analysis to locate and understand action-oriented concerns. In *Proceedings of the 6th International Conference on Aspect-Oriented Software Development, AOSD 2007, Vancouver, British Columbia, Canada, March 12-16, 2007 (ACM International Conference Proceeding Series, Vol. 208)*, Brian M. Barry and Oege de Moor (Eds.). ACM, 212–224.

[39] Ferdian Thung, Shaowei Wang, David Lo, and Julia L. Lawall. 2013. Automatic recommendation of API methods from feature requests. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, Ewen Denney, Tevfik Bultan, and Andreas Zeller (Eds.). IEEE, 290–300.

[40] Christoph Treude, Martin P. Robillard, and Barthélémy Dagenais. 2015. Extracting Development Tasks to Navigate Software Documentation. *IEEE Trans. Software Eng.* 41, 6 (2015), 565–581.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008.

[42] Denny Vrandecic. 2013. The Rise of Wikidata. *IEEE Intelligent Systems* 28, 4 (2013), 90–95.

[43] Rensong Xie, Xianglong Kong, Lulu Wang, Ying Zhou, and Bixin Li. 2019. HiRec: API Recommendation using Hierarchical Context. In *30th IEEE International Symposium on Software Reliability Engineering, ISSRE 2019, Berlin, Germany, October 28-31, 2019*, Katinka Wolter, Ina Schieferdecker, Barbara Gallina, Michel Cukier, Roberto Natella, Naghmeh Ivaki, and Nuno Laranjeiro (Eds.). IEEE, 369–379.

[44] Xin Ye, Hui Shen, Xiao Ma, Razvan C. Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, Laura K. Dillon, Willem Visser, and Laurie Williams (Eds.). ACM, 404–415.

[45] Hongyu Zhang, Anuj Jain, Gaurav Khandelwal, Chandrashekhar Kaushik, Scott Ge, and Wenxiang Hu. 2016. Bing developer assistant: improving developer productivity by recommending sample code. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane ClelandHuang, and Zhendong Su (Eds.). ACM, 956–961.