API parameter recommendation based on language model and program analysis

Cuong Tran Manh
Faculty of Information Technology
Vietnam National University
Ha Noi, Viet Nam
iammaytinhdibo@gmail.com

Kien Tran Trung
Faculty of Information Technology
Vietnam National University
Ha Noi, Viet Nam
kientt312@gmail.com

Tan M. Nguyen
Faculty of Information Technology
Vietnam National University
Ha Noi, Viet Nam
email address or ORCID

***Tóm tắt nội dung*—APIs are extensively and frequently used by programmers to leverage existing libraries and improve productivity. Efficiently passing suitable actual parameters to APIs, on the other hand, is still a non-trivial task due to inadequate examples and documentation. A common approach to mitigate the problem is building a parameter usage database from existing code bases. However, it stands to reason that this technique still has its inherent limitation with regard to unseen code patterns. Furthermore, suggested argument expressions may not be syntactically and semantically correct. In this paper, we introduce Flute, a program analysis and language model combined method for recommending API parameters. The source codes are first analysed to generate syntactically legal and type-valid candidates. Then, these candidates are ranked by using a language model. Our empirical results indicate that Flute achieves 80% top-1 accuracy and outperforms the state-of-the-arts on parameter recommendation.**

***Index Terms*—parameter recommendation, language model, program analysis**

I. Introduction

Việc phát triển các sản phẩm công nghệ được xem là một trong những yếu tố quan trọng trong nền công nghiệp hiện đại. Từ đó, các công cụ hỗ trợ phát triển sản phẩm được ra đời. Các công cụ này giải quyết được các vấn đề về thời gian phát triển, cũng như chất lượng của sản phẩm. Gợi ý mã được xem là một giải pháp hiệu quả trong số đó. Báo cáo này tập trung giải quyết bài toán gợi ý mã cho các biểu thức đối số của các API (Application Programming Interface).

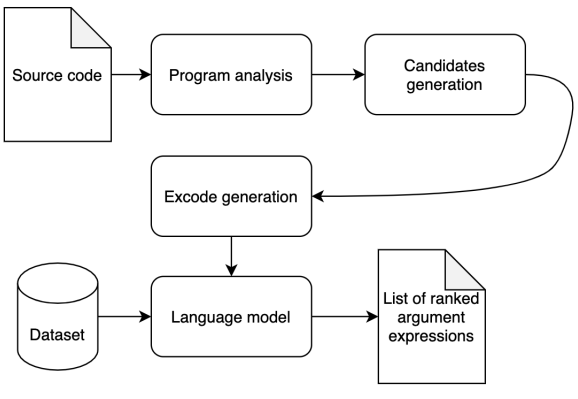
Một số phương pháp như Precise [1] và PARC [2], tập trung vào gợi ý tham số cho API sử dụng các truy vấn từ cơ sở dữ liệu có sẵn. Tuy kết quả ban đầu khả quan nhưng các phương pháp này gặp một số vấn đề không thể đưa vào sử dụng thực tiễn. Hai phương pháp này phụ thuộc vào cách biểu diễn lại thông tin của mã nguồn mà không thể xử lý khi gặp các đoạn mã sử dụng chưa từng có trước đây.

Phương pháp hiện tại mang tên Flute sử dụng mô hình ngôn ngữ trong học máy kết hợp phân tích chương trình trong software engineering để gợi ý các biểu thức đối số cho Java API.

Phân tích chương trình được sử dụng kiểm chứng các biểu thức phù hợp về cú pháp và ngữ nghĩa cho các vị trí tham số đang gợi ý. Nhiệm vụ tiếp theo được mô hình ngôn ngữ đảm nhiệm là xếp hạng lại các biểu thức theo khả năng sử dụng làm tham số của các biểu thức này. Đặc biệt, trước khi sử dụng mô hình ngôn ngữ, các biểu thức trên tiếp tục được biểu diễn dưới dạng excode, là một cách biểu diễn mã nguồn ở mức tổng quát hơn nhằm thể hiện được các mẫu sử dụng API. Các đoạn mã gợi ý dưới dạng excode được tính điểm bởi mô hình ngôn ngữ được huấn luyện trên mã nguồn ở dạng excode, dựa trên khả năng xuất hiện khi đã biết ngữ cảnh là phần mã nguồn phía trước. Tiếp theo, đoạn mã được chọn sẽ được cụ thể hóa thành các biểu thức ở dạng lexical (dạng mã nguồn Java). Cuối cùng, các đoạn mã gợi ý dạng mã nguồn được tính điểm với ngữ cảnh cho trước. Để làm được điều đó, phương pháp sử dụng đồng thời thêm một mô hình ngôn ngữ khác được huấn luyện trên mã nguồn Java. Đi kèm với đó, nghiên cứu này cũng phân tích một số đặc điểm của mã nguồn như sự tương đồng giữa tham số khai báo và biểu thức đối số nhằm tăng hiệu suất gợi ý.

II. Quy trình gợi ý tham số

Mục tiêu của bài toán là đưa ra danh sách các gợi ý phù hợp về cú pháp và ngữ nghĩa làm đối số cho các lời gọi API. Quy trình gợi ý của Flute được thể hiện qua Hình 1.



Hình 1: Tổng quan quy trình gợi ý tham số

Đầu tiên, Flute dựa vào nội dung đoạn mã người dùng đang viết để tìm ra các biến số có thể truy cập được, sau đó sinh các biểu thức từ các biến số trên sao cho phù hợp về cú pháp

và ngữ nghĩa tại vị trí tham số. Tiếp đó, các biểu thức được chuyển sang dạng excode, là một dạng biểu diễn khác của mã nguồn nhằm giữ lại các thông tin cần thiết và loại bỏ các thông tin như tên biến giúp mô hình có thể nhận biết được các mẫu sử dụng API. Như vậy, có thể có nhiều biểu thức đối số dạng mã nguồn có cùng chung mã excode. Sau đó, các biểu thức đối số được tính điểm và k biểu thức đối số có điểm cao nhất được chọn vào danh sách gợi ý. Trong đó, điểm của một biểu thức đối số là sự kết hợp giữa hai thành phần, bao gồm điểm được tính bởi mô hình ngôn ngữ excode và điểm được tính bởi mô hình ngôn ngữ dạng mã nguồn.

A. Lexical candidate generation

Bước đầu tiên trong quá trình gợi ý là sinh các biểu thức đối số. Tại bước này, kỹ thuật chủ yếu là phân tích từ mã nguồn nhằm đưa ra các gợi ý hợp lệ về mặt cú pháp lẫn ngữ nghĩa. Tại bước này sẽ trải qua các phần bao gồm: xác định thông tin ngữ cảnh, lựa chọn phương thức cần gợi ý, sinh các biểu thức từ thông tin ngữ cảnh và sau cùng là lựa chọn ra các biểu thức gợi ý.

Xác định các thông tin ngữ cảnh: trước hết, chúng tôi xác định các biến có thể sử dụng là các biến có khả năng truy cập và đã được khởi tạo giá trị khác *null*. Để làm được điều này, chương trình được phân tích theo đồ thị CFG (Control-flow graph), duyệt các đường đi có thể của chương trình các dòng lệnh để xác định được các lần khởi tạo giá trị của từng biến và phạm vi phần ngữ cảnh hiện tại.

Lựa chọn phương thức: các nghiên cứu như Precise và PARC chỉ hoạt động sau khi người dùng lựa chọn phương thức, nếu có xảy ra nạp chồng phương thức trong Java[1] thì 2 phương pháp trên không thể gợi ý. Để lựa chọn ra phương thức phù hợp, chúng tôi sẽ xét đến ngữ cảnh của đoạn mã. Ví dụ trong trường hợp trên Hình 2 phương thức có tên *run* tuy nhiên khi xét tới khai báo của nó tại lớp *Tools*, nhận thấy, có 3 phương thức cùng tên được khai báo. Đây là kỹ thuật nạp chồng phương thức khi các phương thức có cùng tên trong Java. Đầu tiên, khi xét đến khả năng truy cập và loại bỏ phương thức *private*. Sau đó, dựa vào ngữ cảnh khi lời gọi phương thức là vế phải của biểu thức trả về số nguyên, phương thức cần xác định ở đây phải trả về kiểu số *public int run(int)*.

Sinh các biểu thức từ biến đã xác định: sau khi đã xác định được các biến có thể dùng trong phạm vi của đoạn mã, phương pháp sẽ xác định các biểu thức sinh ra từ các biến trên. Ví dụ nếu chỉ sinh ra các biểu thức truy cập trường phương pháp sẽ sinh ra được các biểu thức như *tool.title*, đối với các trường không thể truy cập tại vị trí hiện tại sẽ bị loại bỏ. Ngoài ra, các biểu thức như số, chuỗi, tạo mới đối tượng nếu thoả mãn sẽ được sinh ra, ví dụ tham số truyền vào là một đối tượng *Student*, biểu thức *new Student(...)* cũng sẽ là một biểu thức có thể đưa vào gợi ý.

Sinh các biểu thức gợi ý: bước cuối cùng, dựa vào thông tin các phương thức đã chọn ở trên, nếu các biểu thức vừa sinh thỏa mãn được vị trí tham số của phương thức sẽ được gợi ý tương tự việc lựa chọn phương thức bằng cách dựa vào ngữ cảnh.

[1]https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

```
// Example.java
class Tools{
    public String title;
    private long run(boolean o){...}
    public String run(String s){...}
    public int run(String o){...}
}
public class Example{
    public void main() {
        Tools tool = new Tool();
        long position = tool.run(...
    }
}
```

Hình 2: Code example 1

B. Excode candidates generation

Extended code token (excode) là một cách biểu diễn mã nguồn bao gồm thông tin về kiểu token, thêm thông tin về kiểu dữ liệu và lược bỏ tên biến so với code token thông thường. Nhiều phương pháp có liên quan [3][4] sử dụng các khái niệm tương đương nhằm đưa ra một kiểu biểu diễn các thông tin cần thiết cho mã nguồn.

Kiểu token (token type) thể hiện chức năng của token trong mã nguồn: một lời truy cập trường, một lời gọi hàm hay câu lệnh điều kiện, v.v.. Thông tin về cú pháp được giữ lại giúp mô hình ngôn ngữ excode dự đoán token tiếp theo chính xác và chặt chẽ hơn.

Thông tin về kiểu dữ liệu (data type) được thêm vào excode phân biệt trường hợp các đoạn mã giống nhau về mặt từ vựng nhưng có chức năng khác nhau. Từ đó, ta có thể xác định ràng buộc kiểu và khả năng truy cập trường hoặc phương thức của một biến. Chẳng hạn *x.length* với *x* kiểu *String* là lời gọi phương thức *length* nhưng ở chỗ khác với *x* kiểu mảng, đoạn mã trên lại có nghĩa là truy cập trường *length*.

Việc dùng tên biến phụ thuộc nhiều vào dự án, thói quen của lập trình viên và tính địa phương của biến. Vì vậy, mô hình ngôn ngữ hoạt động trực tiếp với biểu diễn từ vựng sẽ khó nhận ra các đoạn mã có cùng ý nghĩa nhưng sử dụng tên các biến khác nhau, ví dụ *int len = s.length();* và *int l = str.length();*. Việc chỉ lưu lại thông tin về kiểu thay vì tên biến khiến các đoạn mã như vậy có cùng biểu diễn, làm tăng xác suất xuất hiện của mẫu mã nguồn đó. Điều này sẽ giúp các mô hình ngôn ngữ thống kê có thể nắm bắt được mã nguồn ở mức độ trừu tượng cao hơn và thể học được mẫu mã nguồn từ nơi này để gợi ý cho nơi khác. Mặt khác, tên của kiểu dữ liệu, phương thức và trường được giữ lại do chúng được thiết kế để có thể được tái sử dụng. Excode được sinh ra từ mã nguồn theo luật như Bảng I

Với những lợi ích trên, excode được chúng tôi sử dụng làm một thước đo đánh giá các gợi ý. Cụ thể, chúng tôi sẽ xây dựng mô hình ngram huấn luyện dữ liệu excode và sử dụng nó để tính xác suất xuất hiện của một ứng viên dưới dạng excode.

Kiểu token	Excode tương ứng	Ví dụ code → excode
Toán tử đa ngôi *o*	OP(o) [3]	+ → OP(PLUS)
Toán tử một ngôi *o*	UOP(o)	++ → UOP(posIncrement)
Toán tử gán *o*	ASSIGN(o)	+= → ASSIGN(PLUS)
Dấu phân cách *sp*	SEPA(sp)	; → SEPA(;)
Dấu ngoặc	Từ đại diện [3]	{ → OPBLK) → CLOSE_PART
Kiểu dữ liệu *T*	TYPE(T)[3]	short → TYPE(short)
Biến *v*	VAR(kiểu(v)) [3]	num (int) → VAR(int)
Literal *l*	LIT(kiểu(l)) [3]	0x01 → LIT(num), true → LIT(boolean)
Lời gọi phương thức *m*	M_ACCESS(kiểu(caller(m)), m, số tham số(m))	str.length() → M_ACCESS(String,length,0)
Lời gọi hàm khởi tạo *c*	C_CALL(lớp(c),lớp(c))	new String(...) → C_CALL(String,String)
Lời truy cập trường *f*	F_ACCESS(kiểu(caller(f)),f)	s.area → F_ACCESS(Shape,area)
Literal đặc biệt	Từ đại diện [3]	0 → LIT(zero), null → LIT(null), ? → LIT(wildcard)
Câu lệnh điều khiển *cs*	STSTM(cs) ở đầu scope, ENSTM(cs) ở cuối scope	if {...} → STSTM{IF}…ENSTM{IF}, for (...) {...} → STSTM{FOR} …ENSTM{FOR}, return → STSTM{RETURN} …ENSTM{RETURN}
Lớp *c*	CLASS{START,c} ở đầu scope CLASS{END,c} ở cuối scope	Class Car {...} → CLASS{START,Car} …CLASS{END,Car}
Khai báo phương thức *m*	METHOD{kiểu trả về, m} ở đầu scope, ENDMETHOD ở cuối scope	String toString() {...} → METHOD{String, toString} … ENDMETHOD

Bảng I: Các luật sinh mã excode từ mã nguồn của chương trình

C. Lexical Similarity

Liu và cộng sự [5] đã phân tích về sự tương đồng giữa nội dung của biểu thức đối số với tên của tham số. Họ đã đưa ra công thức lexsim (lexical similarity) tính độ tương đồng như sau:

$$lexsim(a,p) = \frac{|comterms(a,p)| + |comterms(a,p)|}{|terms(a)| + |terms(p)|} \quad (1)$$

Trong đó:

- ***terms(s)***: tập từ có nghĩa được tách ra từ xâu ký tự *s* dựa trên các chữ hoa và dấu gạch dưới, với giả sử là các xâu tuân theo camel case hoặc snake case.
- ***comterms(n1, n2):*** tập từ con chung dài nhất giữa *terms(n1)* và *terms(n2)*.

Thống kê cho thấy tham số và đối số thường hoàn toàn tương đồng hoặc hoàn toàn không tương đồng. Cụ thể, trên hai bộ mã nguồn lớn là Eclipse và Netbean thì 22.33% độ tương đồng tính được có giá trị bằng 0 và 68.24% độ tương đồng có giá trị bằng 1. Đây là một kết quả đáng được xem xét bởi phần lớn đối số rất giống với tham số. Do đó, chúng tôi đã sử dụng độ tương đồng làm một thước đo để lựa chọn gợi ý.

D. Scoring

Trong các phần trước, chúng tôi đã đề cập đến ba tiêu chí đánh giá một biểu thức ứng viên là xác suất xuất hiện tính bởi mô hình được huấn luyện trên dữ liệu lexical, xác suất xuất hiện tính bởi mô hình được huấn luyện trên dữ liệu excode và độ tương đồng giữa tham số và đối số. Điểm của một biểu thức ứng viên sẽ được tính bằng phép nhân của các điểm thành phần trên. Các cấu hình đánh giá và kết quả được thể hiện ở phần tiếp theo.

III. Kết quả sơ bộ

A. Cấu hình đánh giá

Chúng tôi sử dụng hai bộ mã nguồn lớn là Netbeans và Eclipse để thực hiện đánh giá. Với mỗi project, chúng tôi thực hiện 10-fold cross validation: lần lượt 9 fold sử dụng để huấn luyện mô hình và fold còn lại để kiểm thử. Ngoài ra, đây cũng là hai dự án được nghiên cứu PARC sử dụng để đánh giá.

B. Metrics

Về tiêu chí đánh giá, hai tiêu chí *precision* và *recall* được sử dụng để tiện so sánh với phương pháp PARC:

$$Precision = \frac{recommendations\ made \cap relevant}{recommendations\ made} \quad (2)$$

$$Recall = \frac{recommendations\ made \cap relevant}{recommendations\ requested} \quad (3)$$

Trong đó *recommendations made* là tổng số lần kỹ thuật của chúng tôi thực hiện việc gợi ý tham số, *relevant* là số lần mà các tham số thực tế có trong danh sách *k* gợi ý của Flute (với *k* $\in \{1, 3, 5, 10\}$), *recommendations requested* là số lượng tham số có trong bộ dữ liệu kiểm thử.

C. Kết quả

Đầu tiên, chúng tôi so sánh kết quả với nghiên cứu PARC. Flute lựa chọn thử nghiệm trên các API thuộc lớp SWT[2] đối với Eclipse, AWT[3] và Swing[4] đối với Netbeans để đưa ra kết quả so sánh tương đồng với thử nghiệm của PARC.

[2]https://www.eclipse.org/swt/

[3]https://docs.oracle.com/javase/8/docs/technotes/guides/awt/

[4]https://docs.oracle.com/javase/8/docs/technotes/guides/swing/index.html

Project	Top-k	Precision		Recall	
		PARC	FLUTE	PARC	FLUTE
Eclipse	1	47.65%	**71.63%**	46.65%	**54.86%**
	3	65.05%	**79.21%**	63.68%	60.66%
	5	-	**80.87%**	-	61.94%
	10	72.26%	**85.71%**	70.73%	65.64%
Netbeans	1	46.46%	**76.48%**	44.86%	**62.06%**
	3	66.20%	**86.15%**	66.75%	**69.91%**
	5	-	**87.34%**	-	**70.87%**
	10	72.06%	**88.36%**	69.57%	**71.70%**

Bảng II: So sánh kết quả giữa PARC và Flute

Tại kết quả *precision*, Flute thể hiện được sự vượt trội khi đưa ra các dự đoán so với PARC chênh lệch khoảng 13% ˜ 24%. Đối với *recall* hay trong các trường hợp thực tế, tại các gợi ý đầu tiên, Flute đưa ra được các gợi ý đa số sẽ đúng theo ý nghĩ của lập trình viên và cao hơn PARC. Khi nhìn vào *recall* top-1, Flute tốt hơn PARC 8% ˜ 17%. Một số vị trí sau kết quả xấp xỉ giữa hai nghiên cứu, tuy nhiên, nghiên cứu trước đây cho thấy các lập trình viên thường chỉ quan tâm các gợi ý ở vị trí đầu tiên [6] việc tăng thứ hạng các vị trí càng cao sẽ có hiệu quả hơn khi lập trình viên không phải tìm kiếm các gợi ý bằng cách cuộn xuống.

Để có một đánh giá tổng quát nhất, chúng tôi tiếp tục đưa ra gợi ý trên toàn bộ các API được sử dụng trên 2 bộ mã nguồn. Kết quả đánh giá tại Bảng III thể hiện Flute có độ chính xác cao. Cụ thể, tại *precision* top-1 Flute đã đạt kết quả lên tới gần 80%. Tức là trong hầu hết các trường hợp, người dùng có thể chọn ngay gợi ý đầu tiên. Độ chính xác được thể hiện khá tương đồng trên 2 bộ mã nguồn. Điều đó chứng tỏ rằng phương pháp tiếp cận của Flute thực sự có hiệu quả.

Đối tượng thử nghiệm	Netbeans	Eclipse
Top-1 precision	78.91%	79.53%
Top-3 precision	89.81%	91.11%
Top-5 precision	92.11%	93.28%
Top-10 precision	94.36%	95.14%
Top-1 recall	66.72%	66.00%
Top-3 recall	75.93%	75.61%
Top-5 recall	77.88%	77.41%
Top-10 recall	79.78%	78.95%

Bảng III: Kết quả đánh giá trên toàn bộ bộ mã nguồn

IV. Các nghiên cứu liên quan

Phần này đề cập đến các nghiên cứu liên quan đến bài toán gợi ý tham số cho API của chúng tôi. Chúng tôi tập trung khảo sát ở 3 phương pháp và chia thành 2 phân loại, thứ nhất là TabNine, thứ hai là Precise và Parc.

A. TabNine

TabNine là một trình cắm hỗ trợ cho nhiều trình soạn thảo và IDE (Integrated Development Environment - Môi trường phát triển tích hợp). Công cụ này sử dụng mô hình GPT-2[7], đây là mô hình được sử dụng rất nhiều trong việc xử lý ngôn ngữ tự nhiên. Bằng cách học các đoạn mã trên kho Github. Nó thể hiện tốt khi sinh ra được các đoạn mã theo các mẫu rất chuẩn so với tài liệu của API do có một lượng dữ liệu lớn được học trong mô hình. Tuy nhiên, có một điểm yếu ở đây là TabNine không phân tích đến tính sẵn có của các biểu thức gợi ý [8], thường xuyên gây ra tình trạng gợi ý xong khiến người dùng phải gỡ lỗi ngược lại. Trên hình 3, TabNine gợi ý phương thức không hề tồn tại thuộc lớp HttpURLConnection[5].

Hình 3: Ví dụ về trường hợp dự đoán tham số không phù hợp của TabNine

B. Các nghiên cứu tập trung gợi ý tham số

Hai nghiên cứu này đều tập trung cụ thể đến việc gợi ý tham số cho API. Về ý tưởng, cả hai đều sử dụng cơ sở dữ liệu để lưu trữ các đoạn mã mẫu kết hợp với các gợi ý từ công cụ JDT.

1) Precise: Đây được xem là nghiên cứu mở đầu tập trung về gợi ý tham số cho API. Precise cung cấp một phần mở rộng nhằm tăng cường độ chính xác của các gợi ý mà công cụ JDT trên Eclipse đề xuất bằng cách sử dụng thêm thuật toán *kNN*. Các kiểu tham số được Precise hỗ trợ bao gồm các kiểu tên các lời gọi phương thức, truy cập trường/lớp/biến, truy cập mảng, biểu thức cast và một số kiểu literal (số/null/boolean). Các trường hợp này chiếm khoảng 43% loại tham số trong các dự án thực tế là khá ít nhưng đủ đơn giản để gợi ý.

2) PARC: PARC đã hỗ trợ nhiều kiểu hơn các kiểu so với Precise tuy nhiên điều này khiến kết quả bị giảm xuống do mô hình cần dự đoán nhiều lượng gợi ý hơn, đây là một vấn đề mà mô hình dự đoán cần cân nhắc khi cân bằng giữa chi phí dự đoán và kết quả. PARC sử dụng thuật toán *simhash* và biểu diễn các đoạn mã mẫu dưới dạng vectơ để gợi ý.

3) Vấn đề chung: Cả hai nghiên cứu nêu trên đều gặp vấn đề về số lượng kiểu của biểu thức gợi ý mà không đưa ra được số lượng tối ưu nhất như trong giải pháp của chúng tôi. Ngoài ra, với cách biểu diễn dữ liệu về cách sử dụng API của 2 phương pháp này không xác định rõ trong trường hợp 2 dạng biểu thức giống nhau có cùng kiểu nhưng khác tên.

References

[1] Cheng Zhang et al. “Automatic parameter recommendation for practical API usage”. In: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE. 2012, pp. 826–836.

[2] Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. “PARC: Recommending API methods parameters”. In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2015, pp. 330–332.

[5]https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html

[3] Son Nguyen et al. “Combining program analysis and statistical language model for code statement completion”. In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2019, pp. 710–721.

[4] Tung Thanh Nguyen et al. “A statistical semantic language model for source code”. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 2013, pp. 532–542.

[5] Hui Liu et al. “Nomen est omen: Exploring and exploiting similarities between argument and parameter names”. In: *Proceedings of the 38th International Conference on Software Engineering*. 2016, pp. 1063–1073.

[6] Marcel Bruch, Martin Monperrus, and Mira Mezini. “Learning from examples to improve code completion systems”. In: *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*. 2009, pp. 213–222.

[7] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: (2019).

[8] Boris Barath, William Knottenbelt, and Thomas Heinis. “Improving Code Completion with Machine Learning”. In: (2020).