Machine Learning Based Recommendation of Method Names: How Far Are We

By Disha A Patel

Overview

- INTRODUCTION
- The paper makes the following contributions
- Recommendation for Method Name
- Recommendation for Other Identifiers
- Machine Learning based Code Recommendation
- EXPERIMENTAL SETUP
  - Evaluated Approach
  - Research Questions
  - Metrics
- RESULTS AND ANALYSIS
- Threats to Validity
- HEURISTICS BASED ALTERNATIVE APPROACH
  - Overview
  - Distinguishing Getter/Setter Methods
  - Distinguishing Delegations
  - Frequency-based Name Recommendation
  - Comparison Against code2vec
- DISCUSSION
  - Implications
  - Limitations
- CONCLUSIONS AND FUTURE WORK

INTRODUCTION

Identifiers are widely employed to identify unique software entities. According to a recent study

Identifiers account for approximately 70% of the source code in terms of characters

A well-constructed identifier not only follows languagespecific naming conventions, but also conveys intention/responsibility of its associated software entity

Consequently, high quality identifiers have significant influence on the readability of source code.

The paper makes the following contributions:

First, a comprehensive assessment and in-depth analysis of the state-of-the-art approach in ML-based method name recommendation. The analysis not only reveals the state of the art, but also discovers where and why the approach works or doesn't work.

Second, a simple and straightforward approach in recommending method names. It significantly outperforms the state-of-the-art ML-based complicated approach in the evaluation, which suggests that ML-based approaches may still have a long way to go.

Recommendation for Method Name

The quality of identifiers proved to have a significant impact on the readability and maintainability of software source code Method names, as a special kind of identifiers, are especially important because they serve as cornerstone of abstraction for aggregated behaviors.

Consequently, a series of approaches have been proposed to improve the quality of method names

Recommendation for Other Identifiers

There are also a lot of automated approaches proposed to improve the quality of identifiers

They first generate a dictionary that transforms composing words in identifiers into their associated standard terms.

Second, they infer a standard syntax from examples representative of the different forms that can be associated to the main grammatical functions for arranging those standardized terms into a sequence.

Machine Learning based Code Recommendation

- Recommender systems have a wide range of applications in software engineering to improve productivity and reliability
- Many of these systems employ machine learning techniques to assist developers in writing or maintaining software source code.
- The most popular recommender system used in integrated development environments (IDEs) is code recommendation system
- API method call such models statistically learn the probability distribution of API usage graphs extracted from source code snippets, and then recommend the next API by computing the appearance probability of each API against a given usage graph.

EXPERIMENTAL SETUP

Evaluated Approach

To evaluate the state of the art in ML-based recommendation of method names, we select code2vec for the evaluation.

code2vec is selected because of the following reasons.

First, it represents the state of the art in this field. As introduced in Section II, code2vec was proposed recently on POPL 2019, and proved significantly more accurate than alternative approaches

Second, the source code of its implementation is publicly available, which significantly facilitates the evaluation. It also facilitates other researchers to replicate the experiment. We make the replication package of the evaluation publicly available on GitHub to facilitate third-party replication and further investigation.

Research Questions

RQ1: How well does code2vec work on datasets other than the one employed by the original evaluation conducted by the authors of code2vec?

RQ2: How well does code2vec work with more realistic settings?

RQ3: Can code2vec generate method names correctly when the given method bodies do not contain method name tokens?

RQ4: Where and why does code2vec work?

RQ5: Where and why does code2vec fail?

RQ6: Is code2vec useful for developers? How often does code2vec recommend correctly for methods that are challenging to name manually?

Metrics

$$Precision@k = \frac{N_{accepted}@k}{N_{recommended}}, \quad (1)$$

$$Recall@k = \frac{N_{accepted}@k}{N_{tested}} \quad (2)$$

$$F1@k = 2 \times \frac{Precision@k \times Recall@k}{Precision@k + Recall@k} \quad (3)$$

$$MRR = \frac{1}{N_{tested}} \sum_{i=1}^{N_{tested}} \frac{1}{rank_i} \quad (4)$$

TABLE I
EVALUATION RESULTS ON DIFFERENT DATASETS

Datasets	Number of methods	Precision / Recall			MRR
		Rank 1	Rank 5	Rank 10	
Original Dataset	13,376,807	49.89%	56.99%	58.41%	52.96%
New Dataset	3,826,986	43.87%	49.90%	51.33%	46.51%

RESULTS AND ANALYSIS

- RQ1: Comparable Performance on Different Datasets
- First, code2vec is accurate in recommending method names. The top 1 recommendation is often (at a chance of 49.89% on original dataset and 43.87% on new dataset) correct. It has a great chance (58.41% on original dataset and 51.33% on new dataset) to present the correct method names on its top 10 recommendation list. High MRR (52.96% and 46.51%) also suggests that the correct names are often ranked on the top.
- Second, switching datasets does not result in significant reduction in performance. Although the precision/recall is slightly reduced (e.g., from 49.89% to 43.87% on rank 1), the major reason for the reduction is the size of new dataset: the number of methods in new dataset is only 28.6% of that in original dataset

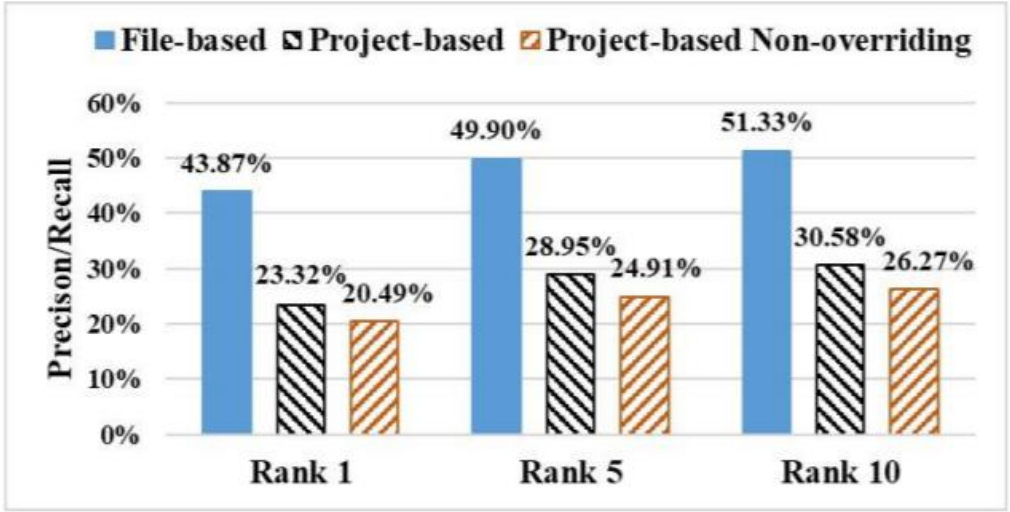


Fig. 1. Evaluation Results with Different Settings

RQ2: Realistic Settings Result in Reduced Performance

- To address research question RQ2, we evaluate code2vec on our new dataset with different settings, i.e., file-based validation, project-based validation, and project-based nonoverriding validation. File-based validation partitions dataset into training, validation and test sets at file level whereas the other two validations work at project level. The only difference between project-based validation and project-based nonoverriding validation is that overriding methods are excluded by the latter but included by the former.

TABLE II
EVALUATION RESULTS ON UNSEEN/SEEN TOKENS

Token Types	Total (N_t)	Correct (N_c)	N_c/N_t
Unseen Tokens	385,225	85,034	22.07%
Seen Tokens	581,237	152,826	26.29%

RQ3: Coining Method Names To address research

- We evaluate the performance of code2vec in generating seen and unseen method name tokens, respectively. For each of method names in new dataset, we split it into tokens according to the Camel-Case naming convention. A token t from method name mn is a seen token if t appears (case insensitive) in the body named by mn. Otherwise, it is an unseen token. We assess how often code2vec can successfully recommend seen/unseen method name tokens during project-based non-overriding validation

```
1  public static boolean contains(String str,
2    String[] array) {
3    for(String s : array)
4      if(str.equals(s))
5        return true;
6    return false;
7  }
8
9  public static boolean contains(int[] values,
10   int candidate) {
11   for(int i = 0; i < values.length; i++)
12     if(values[i] == candidate)
13       return true;
14   return false;
15 }
```

Listing 1. An Example of Coined Method Name

RQ4: Where and Why code2vec Works

```java
public int getPadding() {
  return padding;
}

public int getLength() {
  return length;
}
```

Listing 2. Getter Methods

```java
public void reset() {
  //delegation
  reset(operation);
}

public void reset(String operation) {
  //delegation and server
  reset(operation, true);
}

public void reset(String newOperation,
  boolean logOperationOnChange) {
  //server
  if (time != null)
    if(operation.equals(newOperation) ||
      logOperationOnChange)
      logOperation(operation, elapsedTime());
  time = new Date().getTime();
  if(!operation.equals(newOperation)) {
    operation = newOperation;
    log.info(String.format(
    "Operation '%s' started.", newOperation));
  }
}
```

Listing 3. Delegations

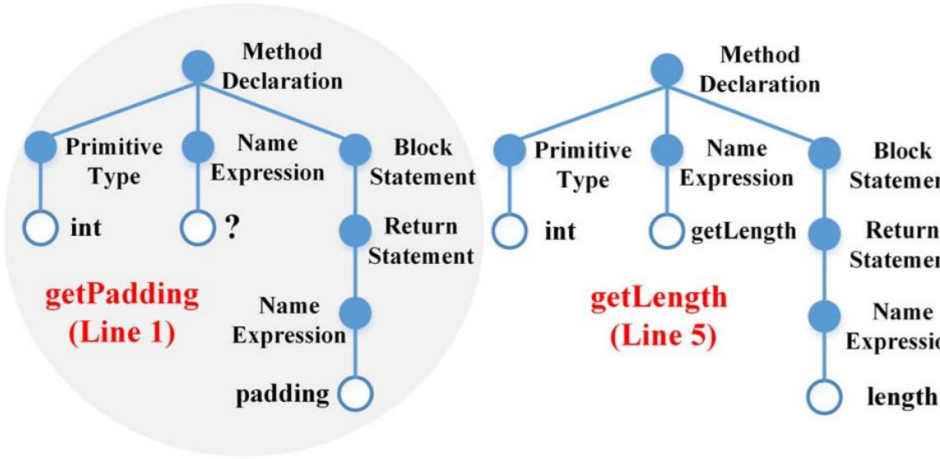
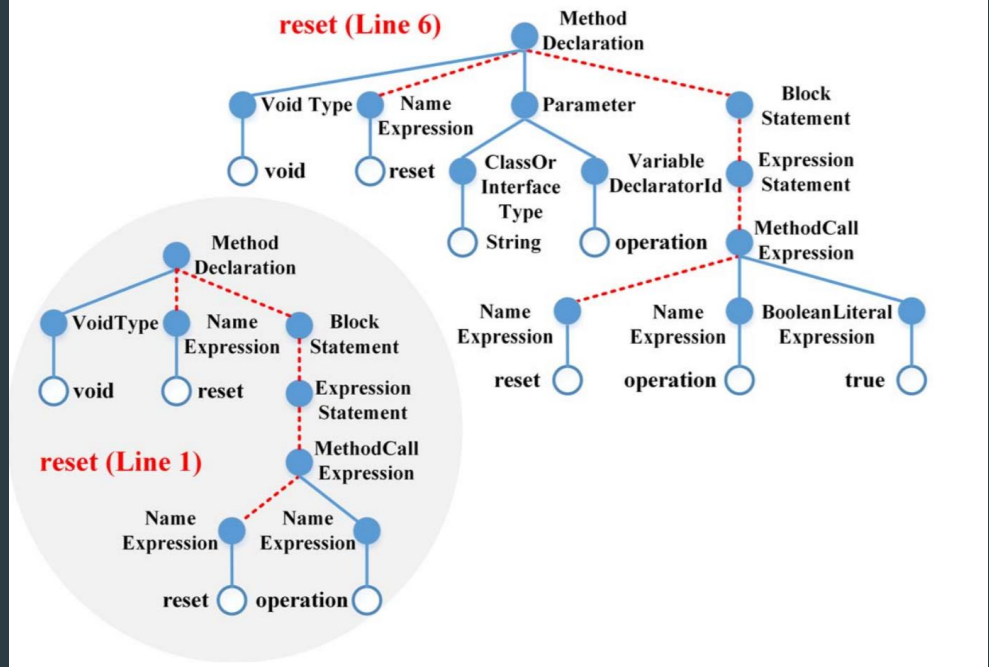


Fig. 2. Common AST Pattern of Getter Methods



. RQ5: Where and Why code2vec Fails

```
1  public TIncrement deepCopy() {
2    return new TIncrement(this);
3  }
4
5  public ReportAttributes build() {
6    return new ReportAttributes(this);
7  }
```

Listing 4. Similar Methods Named Differently

```
1  public static void err(String msg) {
2    System.err.println(msg);
3  }
4
5  public static void rawError(String msg) {
6    System.err.println(msg);
7  }
```

Listing 5. Identical Methods Named Differently

```
1  private void initializeRecyclerview() {
2    RecyclerView.LayoutManager layoutManager =
3    new GridLayoutManager(getActivity(), 2);
4    recyclerView.setLayoutManager(layoutManager);
5    recyclerView.setHasFixedSize(true);
6    adapter = new RecyclerViewAdapter(
7    getActivity().getApplicationContext());
8    recyclerView.setAdapter(adapter);
9  }
10
11 public KeycloakBuilder password(
12   String password) {
13   this.password = password;
14   return this;
15 }
```

Listing 6. Improper Method Names in Testing Set

RQ6: Limited Usefulness for Developers

- First, we invite six developers involved in a commercial project for the evaluation. The commercial project has been released recently by a giant of IT industry to conduct largescale software refactorings.
- Second, for each of the participants, we randomly select one hundred methods developed by himself/herself.
- Third, we request all participants to score the difficulty in naming sampled methods. Notably, participants do not score methods developed by other developers, and thus each of them score exactly one hundred methods. The scores rank between one and five (i.e. 5-point scale [41]-[44]) where one represents least difficulty and five represents highest difficulty.
- Fourth, we apply code2vec of project-based non-overrding validation to the scored methods, and validate the recommendations against manually constructed names.

TABLE III
EVALUATION RESULTS ON MANUALLY SCORED METHODS

Scores	Number of methods	Precision / Recall
1 (Very Easy)	287	34.84%
2 (Easy)	159	6.92%
3 (Normal)	91	5.49%
4 (Difficult)	50	2.00%
5 (Extremely Difficult)	13	0.00%
Total	**600**	**19.50%**

Threats to Validity

- A threat to the external validity is that we employ only one new dataset to validate the impact of switching datasets on the performance of code2vec.
- A threat to construct validity is that we assess the correctness of generated names based on their equivalence to the manually constructed ones.

HEURISTICS BASED ALTERNATIVE APPROACH

- First, based on a sequence of heuristics, HeMa decides whether mb is a getter/setter method. If yes, HeMa generates method name for it automatically based on another sequence of heuristics.
- Second, based on a sequence of heuristics, HeMa decides whether mb is a delegation. If yes, HeMa generates method name for it automatically based on another sequence of heuristics.
- Third, if the preceding heuristics fail, HeMa employs a sequence of heuristics to retrieve methods in a large corpus that share the same return type and parameters (both parameter names and parameter types but regardless of parameter orders) with mb. From the resulting set of methods (notated as Sm), HeMa picks up the most popular method name and suggests it to mb. If Sm = ∅, HeMa refuses to make any recommendation.

Distinguishing Getter/Setter Methods

- First, if the given method body returns nothing, i.e. the return type is void, the given method is not a getter
- Second, if the given method body contains more than one Return Statements, HeMa will not recognize it as a getter method
- Third, if the value returned by the only Return Statement is a field (notated as ${field}) declared within the enclosing class, it is a getter method.

TABLE IV
COMPARISON BETWEEN HEMA AND CODE2VEC

Scores	HeMa			code2vec
	Precision	Recall	F1	Precision
1	54.59%	41.46%	47.13%	34.84%
2	25.00%	10.69%	14.98%	6.92%
3	7.69%	3.30%	4.62%	5.49%
4	9.09%	4.00%	5.56%	2.00%
5	0.00%	0.00%	0.00%	0.00%
Total	**39.94%**	**23.50%**	**29.59%**	**19.50%**

Distinguishing Delegations

- For a given method body mb, HeMa distinguishes it as a delegation if:
- The method body contains a single statement;
- And the statement is a ReturnStatement that returns an invocation on another server method (notated as sMethod). For the potential delegation, HeMa recommends to reuse the name of invoked method (i.e. sMethod).

Frequency-based Name Recommendation

- If the given body mb is neither getter/setter nor delegation, HeMa retrieves a set of methods (Sm) that share the same return type and the same parameter list (including both parameter types and names but regardless of the order of parameters). If Sm is empty, HeMa refuses to make any recommendation. Otherwise, it selects the method name with highest frequency in Sm, and recommends to use this name for the given method body.

Comparison Against code2vec

- From the analysis in the preceding paragraphs, we conclude that the heuristics based HeMa significantly outperforms the state-of-the-art ML-based code2vec

DISCUSSION

Implications

Empirical Settings Are Critical

A Friend in Need Is A Friend Indeed

Complex Approaches Are NOT Necessarily Better than Simple and Straightforward Ones

Coining Method Names with Tokens Outside Method Body

Limitations

Implications

The definition of **empirical** is something that is based solely on experiment or experience.

Overall performance of code2vec is promising, it is not much useful. The major reason is that it frequently works when it is not strongly needed but fails when it is in need.

To investigate the possibility of designing a simple and straightforward approaches whose performance is comparable to code2vec, we propose a heuristics based approach called HeMa. Evaluation results

52% of method names in testing set are not used as method names in the large-scale training set. Consequently, method name recommendation approaches.

FIRST LIMITATIONS

THE FIRST LIMITATION OF THE EMPIRICAL STUDY IS THAT ONLY ONE STATE-OF-THE-ART APPROACH IS INVOLVED IN THE STUDY. NOTABLY, ML-BASED APPROACHES, ESPECIALLY DEEP LEARNING BASED ONES ARE OFTEN TIME AND RESOURCE CONSUMING. CONSEQUENTLY, INVOLVING MORE BASELINES IN THE STUDY COULD SIGNIFICANTLY INCREASE THE COST IN BOTH COMPUTING RESOURCE AND HUMAN RESOURCE OF MANUAL ANALYSIS.

Second limitation

- The second limitation is the limited usefulness of alternative approach proposed in Section V. We design this approach to intuitively reveal the state of the art as well as the possibility of designing simple but effective approaches. Evaluation results in Section V-E suggest that it has successfully accomplished its mission. However, it should be noted that most of method names it recommends successfully are associated with simple methods, like getter/setter and delegations. Developers rarely need help in naming such simple methods, which may suggest that usefulness of the approach is limited.

CONCLUSIONS AND FUTURE WORK

- Researches have recently achieved significant advances in machine learning techniques. As a result, many of the resulting advanced machine learning techniques are exploited to solve software engineering tasks

- Future work is needed to investigate the correlation between difficulty in naming method names and source code metrics. In this paper, we measure the difficulty subjectively by asking developers to give a number ranking from 1 to 5 to represent the difficulty. Such subjective ranking could be inaccurate. In future, it would be interesting to investigate more accurate and more objective ways to measure the difficulty in naming methods. In future, it is also interesting to investigate whether clone detection techniques could be exploited to retrieve similar methods in corpus, and whether such techniques could outperform the heuristics presented in this paper

Questions?

Thank you