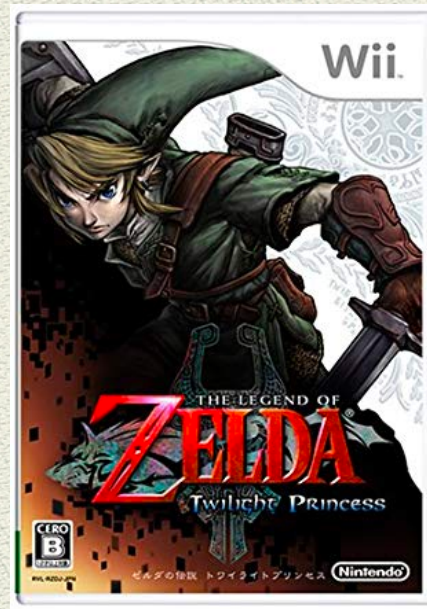


Oreo: detection of clones in the twilight zone

*Vaibhav Saini, Farima Farmahinifarahani,
Yadong Lu, Pierre Baldi, and Cristina Lopes*



発表者：作島

* 画像はイメージです

コードクロンのタイプ

- ◆ タイプ1：テキストの類似性
- ◆ タイプ2：語彙の類似性
- ◆ タイプ3：構文の類似性
- ◆ タイプ4：意味の類似性

twilight zone

- ◆ タイプ³：構文の類似性

- Very Strongly Type3

- Moderately Type3

- Weakly Type3-clone

twilight zone

- ◆ タイプ⁴：意味の類似性

背景

- ◆ Type3までのクローン検出を目的とすることが多い.
- ◆ 純粋なType4クローンを検出する試みはほとんどない.
- ◆ twilight zoneに存在するクローンを検出することが困難.

メソッド測定基準(metrics)

- メソッドがどの程度類似しているのかを判定するための基準

Table 1: Method-Level Software Metrics

Name	Description
XMET	Number of external methods called
VREF	Number of variables referenced
VDEC	Number of variables declared
NOS	Number of statements
NOPR	Total number of operators
NOA	Number of arguments
NEXP	Number of expressions
NAND	Total number of operands
MDN	Method, Maximum depth of nesting
LOOP	Number of loops (for,while)
LMET	Number of local methods called
HVOC	Halstead vocabulary
HEFF	Halstead effort to implement
HDIF	Halstead difficulty to implement
EXCT	Number of exceptions thrown
EXCR	Number of exceptions referenced
CREF	Number of classes referenced
COMP	McCabes cyclomatic complexity
CAST	Number of class casts
NBLTRL*	Number of Boolean Literals
NCLTRL*	Number of Character Literals
NSLTRL*	Number of String Literals
NNLTRL*	Number of Numerical literals
NNULLTRL*	Number of Null Literals

Oreoの概要

- トワイライトゾーンのクローンを検出

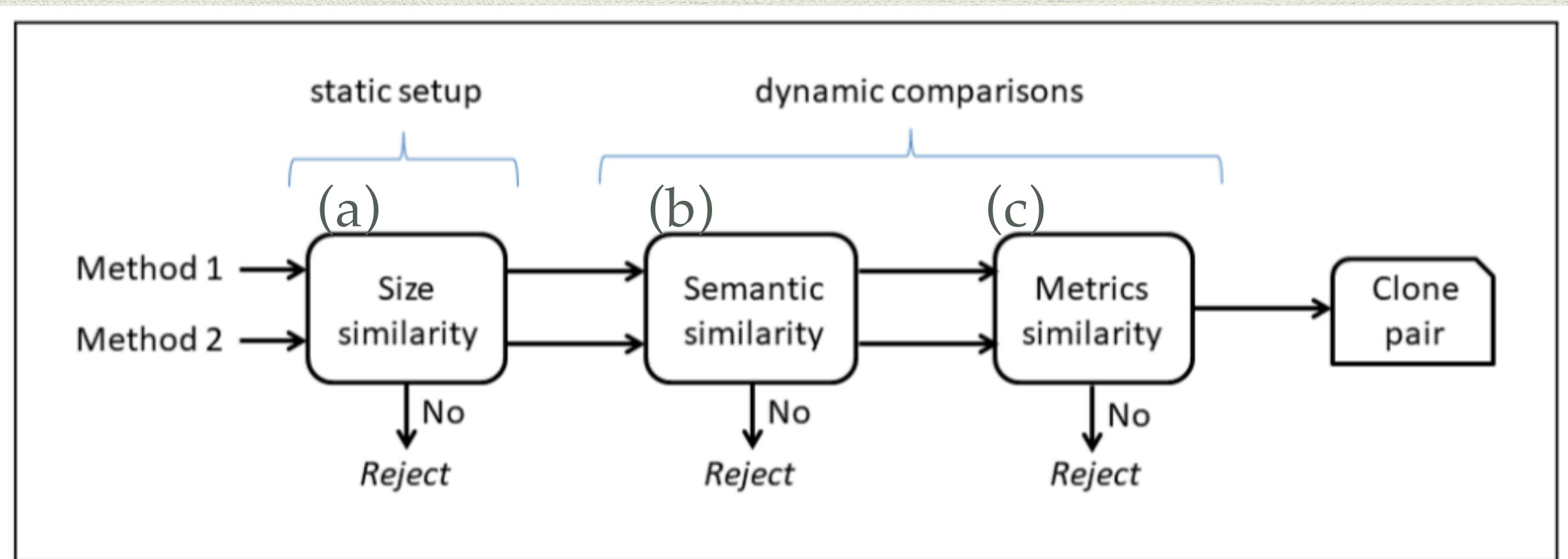


Figure 1: Overview of Oreo.

(a) 前処理

- ◆ 類似しないメソッドペアを事前に排除
 - データサイズが大きく異なるメソッドペアは機能的に類似していることが少ない
 - データサイズで分割
- ◆ Sharding (シャーディング)[8, 31, 35]
 - 1つのDBを複数に分散させて管理・運営

(b) Semantic Similarity

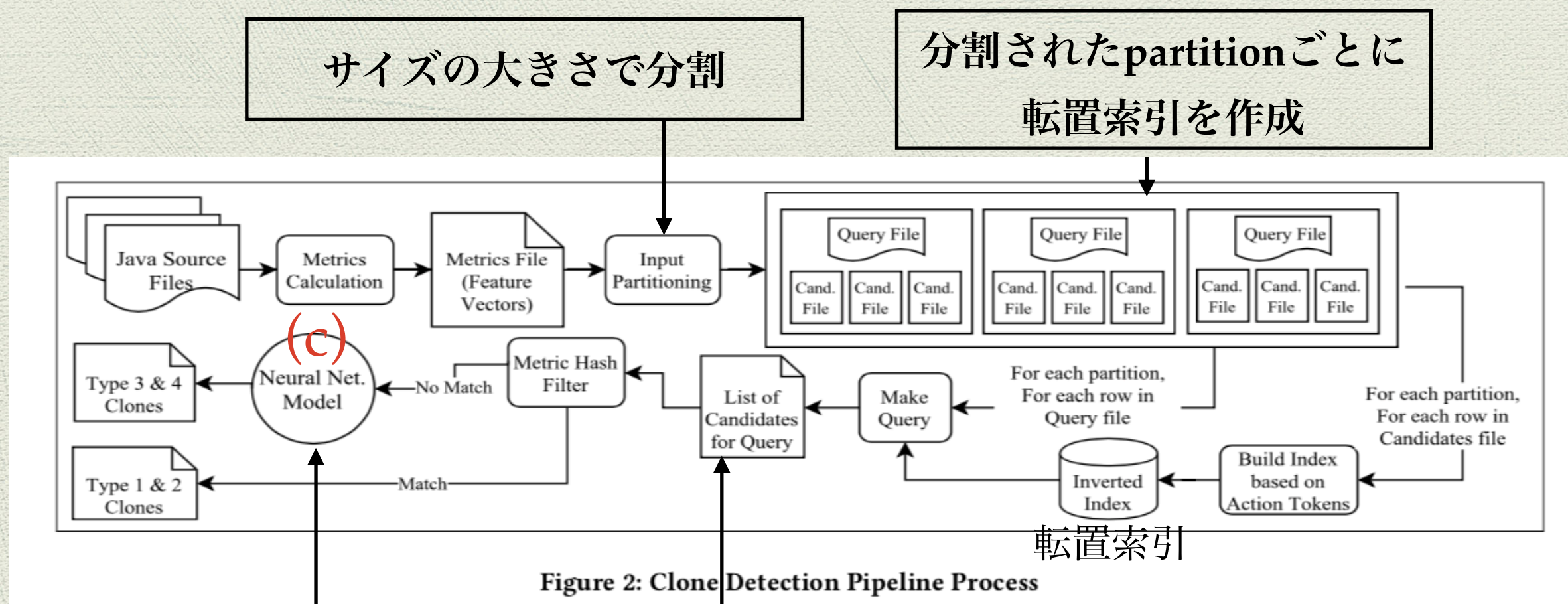
- ◆ Actionトークンという意味的シグネチャを使ってメソッドの意味を捉える

Listing 2: Action Filter Example

```
1 public static String getEncryptedPassword(String password) throws InfoException {
2     StringBuffer buffer = new StringBuffer ();
3     try {
4         byte[] encrypt = password.getBytes("UTF-8");
5         MessageDigest md = MessageDigest.getInstance("SHA");
6         md.update(encrypt);
7         byte[] hashedPasswd = md.digest();
8         for (int i = 0; i < hashedPasswd.length; i++) {
9             buffer.append(Byte.toString(hashedPasswd[i]));
10        }
11    } catch (Exception e) {
12        throw new InfoException(LanguageTraslator.translate("474"), e);
13    }
14    return buffer.toString();
15 }
```

クローン検出の一連の流れ

- ◆ 転置索引：全文探索を行う対象となる文書群から単語の位置情報を格納するための索引構造



候補とクエリをペアにして候補
ペアの特徴ベクトルを作成

クエリメソッドごとに候補クローン
メソッドのリストを返す

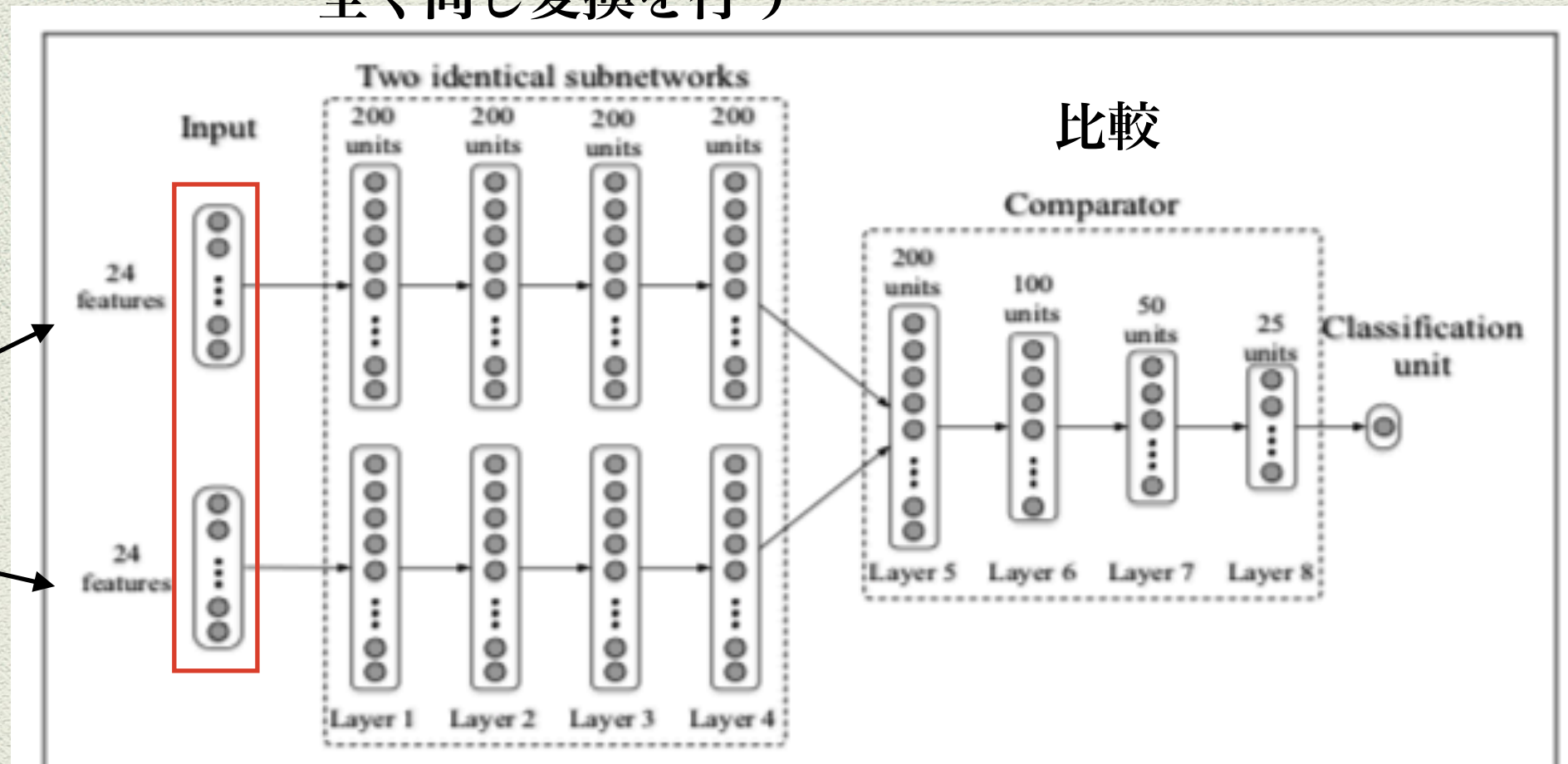
(c) Siameseモデル

- ◆ Siamese architecture neural network [4]
 - 2つの対象の類似性を正確に評価しなければならない問題に適している
 - 入力ベクトルの対象性を扱うことが可能
 - パラメータが少ない

(c) Siameseモデル

全く同じ変換を行う

24個のメトリック
から作成された
48次元ベクトル



2つのメソッドに関連した
特徴ベクトルに対応する
インスタンス

Figure 3: Siamese Architecture Model Trained for Clone Detection: The model consists of two parts: 1) two identical subnetworks, 2) a comparator network, and 3) a Classification unit.

データセット

- ◆ GitHubから50,000のJavaプロジェクトをランダムに選択
- ◆ 50個以上のトークンを持つメソッドを抽出
- ◆ 各特徴ベクトルには48個のメトリックと isClone というバイナリラベルがある

検出結果の例

Type-4の古典的な例(1/2)

◆ ソート

Listing 3: Clone Pair Example: 1

```
1 private void sortByName() {
2     int i, j;
3     String v;
4     for (i = 0; i < count; i++) {
5         ChannelItem ch = chans[i];
6         v = ch.getTag();
7         j = i;
8         while ((j > 0) && (collator.compare(chans[j - 1].getTag(), v) > 0)) {
9             chans[j] = chans[j - 1];
10            j--;
11        }
12        chans[j] = ch;
13    }
14 }
15 -----
16 public void bubblesort( String filenames []) {
17     for (int i = filenames.length - 1; i > 0; i--) {
18         for (int j = 0; j < i; j++) {
19             String temp;
20             if (filenames[j].compareTo(filenames[j + 1]) > 0) {
21                 temp = filenames[j];
22                 filenames[j] = filenames[j + 1];
23                 filenames[j + 1] = temp;
24             }
25         }
26     }
27 }
```

Type-4の古典的な例(2/2)

◆ 拡張子の抽出

Listing 4: Clone Pair Example: 2

```
1 public static String getExtension( final String filename) {
2   if (filename == null || filename.trim().length() == 0 || !filename.contains(".")) return null;
3   int pos = filename.lastIndexOf(".");
4   return filename.substring(pos + 1);
5 }
6 -----
7 private static String getFormatByName(String name) {
8   if (name != null) {
9     final int j = name.lastIndexOf('.') + 1, k = name.lastIndexOf('/') + 1;
10    if (j > k && j < name.length()) return name.substring(j);
11  }
12  return null;
13 }
```

偽陽性

Listing 5: False Positive Example

```
1 public static String getHexString(byte[] bytes) {
2     if (bytes == null) return null;
3     StringBuilder hex = new StringBuilder(2 * bytes.length);
4     for (byte b : bytes) {
5         hex.append(HEX_CHARS[(b & 0xF0) >> 4]).append(HEX_CHARS[(b & 0x0F)]);
6     }
7     return hex.toString();
8 }
9 -----
10 String sequenceUsingFor(int start, int stop) {
11     StringBuilder builder = new StringBuilder();
12     for (int i = start; i <= stop; i++) {
13         if (i > start) builder.append(',');
14         builder.append(i);
15     }
16     return builder.toString();
17 }
```

結果

結果(1/2)

- ◆ PlainDNNよりもSiameseDNNは37%少ないパラメータでトレーニング時間が短縮され、計算負荷が軽減される。

Table 2: Comparison of Precision/Recall Values for Different Models on Test Dataset

Model	Precision	Recall
Logistic Regression	0.846	0.886
Shallow NN	0.931	0.963
Random forest	0.93	0.94
Plain DNN	0.939	0.972
Siamese DNN	0.958	0.974

結果(2/2)

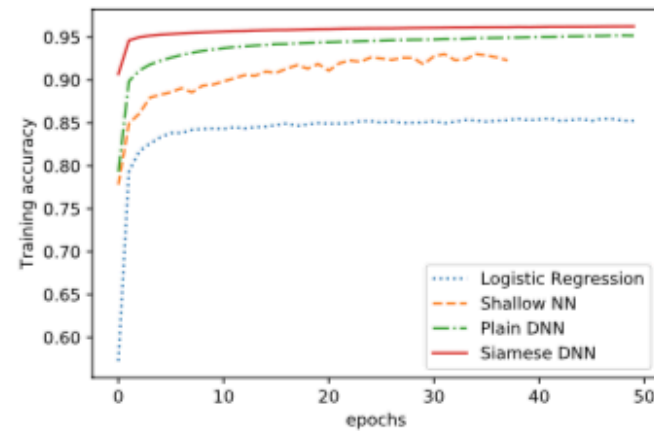


Figure 4: Training Accuracy

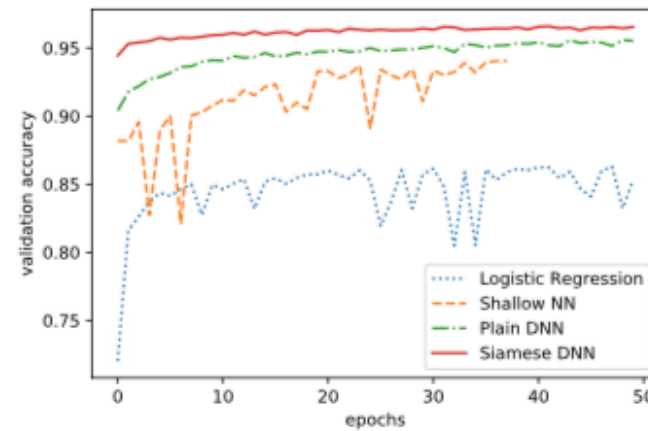


Figure 5: Validation Accuracy

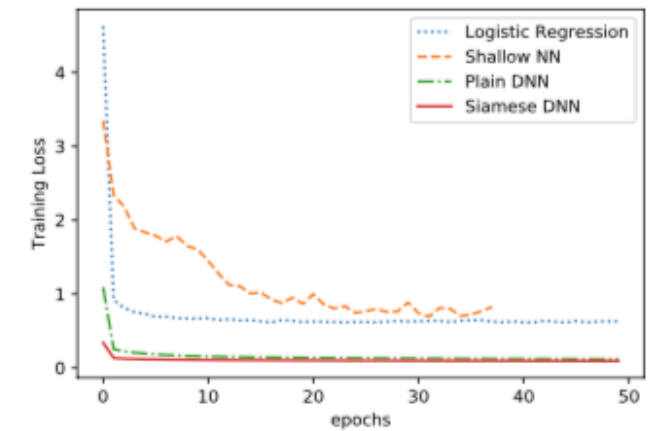


Figure 6: Training Loss

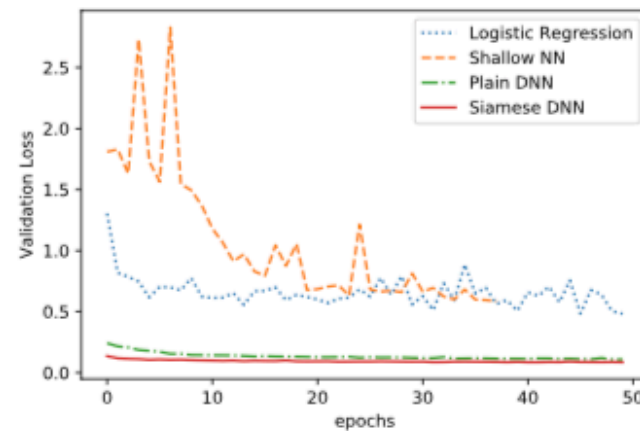


Figure 7: Validation Loss

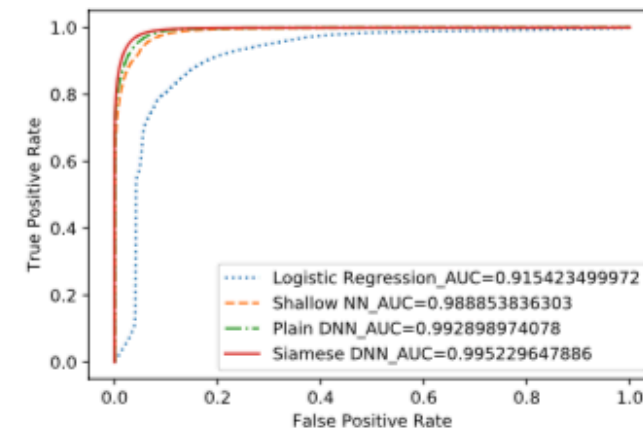


Figure 8: ROC Plot and AUC Values

最先端技術との比較

Table 3: Recall and Precision Measurements on BigCloneBench

Tool	Recall Results										Precision Results		Tool Configuration	
	T1 (35,802)		T2 (4,577)		VST3 (4,156)		ST3 (15,031)		MT3 (80,023)		WT3/T4 (7,804,868)		Sample Strength=400	Based on communications with tool authors or based on [49]
	%	#	%	#	%	#	%	#	%	#	%	#		
Oreo	100	35,798	99	4,547	100	4,139	89	13,391	30	23,834	0.7	57,273	89.5%	MIT=15, Θ = 55%, Γ = 60%
SourcererCC	100	35,797	97	4,462	93	3,871	60	9,099	5	4,187	0	2,005	97.8%	MIT=1, Θ = 70%
CloneWorks	100	35,777	99	4,544	98	4,090	93	13,976	3	2,700	0	35	98.7%	MIT=1, Θ = 70%, Mode=Aggressive
NiCad	100	35,769	99	4,541	98	4,091	93	13,910	0.8	671	0	12	99%	MIL=6, BIN=True, IA=True, Θ = 30%
Deckard*	60	21,481	58	2,655	62	2,577	31	4,660	12	9,603	1	780,487	34.8%	MIT=50, Stride=2, Θ = 85%

* Absolute numbers for Deckard are calculated based on the reported percentage values

- ❖ OreoはSourcererCCによって作成された
クローンペアを使用してトレーニングしてる
→将来的にOreoは最先端クローン検出器に

まとめと所感

- ◆ コードクロンのタイプ3について有効なコードクロン検出器の提案
- ◆ 計算負荷やトレーニング時間の少ないSiameseモデル実証
- ◆ 以前読んだコードの類似性の論文は行列を使っていたため理解が難しかったが、Actionトークンに着目する方法は理解はできた。

参考文献

- ◆ Oreoおよび使用されたデータすべて公開

<http://anonimized>

Type2-clone

```
1 // Original method
2 String sequence(int start , int stop) {
3     StringBuilder builder = new StringBuilder () ;
4     int i = start ;
5     while (i <= stop) {
6         if (i > start) builder.append(',') ;
7         builder.append(i);
8         i++;
9     }
10    return builder.toString () ;
11 }
```

```
13 // Type-2 clone
14 String sequence(int begin, int end) {
15     StringBuilder builder = new StringBuilder () ;
16     int n = begin;
17     while (n <= end) {
18         if (n > begin) builder.append(',') ;
19         builder.append(n);
20         n++;
21     }
22     return builder.toString () ;
23 }
```

Very strongly Type3-clone

```
1 // Original method
2 String sequence(int start , int stop) {
3     StringBuilder builder = new StringBuilder () ;
4     int i = start ;
5     while (i <= stop) {
6         if (i > start) builder.append(',') ;
7         builder.append(i);
8         i++;
9     }
10    return builder.toString () ;
11 }
```

```
25 // Very strongly Type-3 clone
26 String sequence(short start , short stop) {
27     StringBuilder builder = new StringBuilder () ;
28     for (short i = start ; i <= stop ; i++) {
29         if (i > start) builder.append(',') ;
30         builder.append(i);
31     }
32     return builder.toString () ;
33 }
34
```

Moderately Type3-clone

```
1 // Original method
2 String sequence(int start , int stop) {
3     StringBuilder builder = new StringBuilder () ;
4     int i = start ;
5     while (i <= stop) {
6         if (i > start) builder.append(',') ;
7         builder.append(i);
8         i++;
9     }
10    return builder.toString () ;
11 }
```

```
35 // Moderately Type-3 clone
36 String seq(int start , int stop){
37     String sep = ",";
38     String result = Integer.toString ( start ) ;
39     for (int i = start + 1; ; i++) {
40         if (i > stop) break;
41         result = String.join(sep, result , Integer.toString ( i));
42     }
43     return result ;
44 }
```

Weakly Type3-clone

```
1 // Original method
2 String sequence(int start , int stop) {
3     StringBuilder builder = new StringBuilder () ;
4     int i = start ;
5     while (i <= stop) {
6         if (i > start) builder.append(',') ;
7         builder.append(i);
8         i++;
9     }
10    return builder.toString () ;
11 }
```

```
46 // Weakly Type-3 clone
47 String seq(int begin, int end, String sep){
48     String result = Integer.toString(begin) ;
49     for (int n = begin + 1; ;n++) {
50         if (end < n) break;
51         result = String.join(sep, result , Integer.toString(n)) ;
52     }
53     return result ;
54 }
```

Type4-clone

```
1 // Original method
2 String sequence(int start , int stop) {
3     StringBuilder builder = new StringBuilder () ;
4     int i = start ;
5     while (i <= stop) {
6         if (i > start) builder.append(',') ;
7         builder.append(i);
8         i++;
9     }
10    return builder.toString () ;
11 }
```

```
56 // Type-4 clone
57 String range(short n, short m){
58     if (n == m)
59         return Short.toString (n);
60     return Short.toString (n)+ "," + range(n+1, m);
61 }
```