

# Deep Learning Similarities from Different Representations of Source Code

M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, D. Poshyvanyk

# Source Code

# Source Code



# Source Code

- Plain Text
- Abstract Syntax Tree
- Control Flow Graph
- Bytecode



# Source Code

- Plain Text ( domain )
- Abstract Syntax Tree
- Control Flow Graph
- Bytecode



# Source Code

- Plain Text ( domain )
- Abstract Syntax Tree ( structure )
- Control Flow Graph
- Bytecode



# Source Code

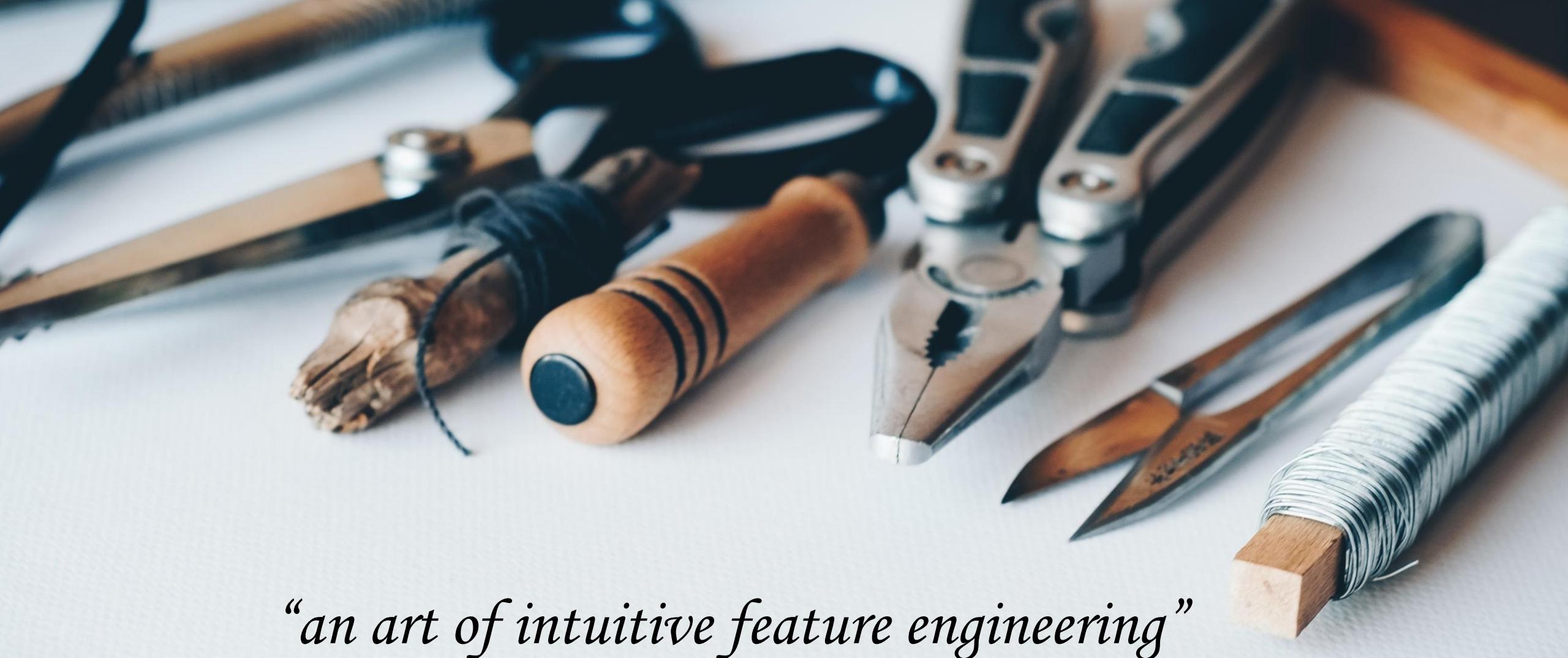
- Plain Text ( domain )
- Abstract Syntax Tree ( structure )
- Control Flow Graph ( execution )
- Bytecode



# Source Code

- Plain Text ( domain )
- Abstract Syntax Tree ( structure )
- Control Flow Graph ( execution )
- Bytecode ( instruction )





*“an art of intuitive feature engineering”*

## **REPRESENTATION (or FEATURE) LEARNING**

Learn vector representation of Source Code

Different Representations + Feature Learning





## APPROACH

- 1. Extract multiple representation from code**
- 2. Learn embeddings for each representation**
- 3. Compute similarities**
- 4. Assemble a combined model**
- 5. Reusability and Transfer Learning**

# 1. Extract multiple representation from code

Identifiers

Abstract Syntax Tree

Control Flow Graph

Bytecode

# 1. Extract multiple representation from code

Identifiers

## Extraction

Leaf nodes of the AST

Stream of identifiers and constants in code

Abstract Syntax Tree

## Normalization

Replace constant values with their type

- < int >
- < float >
- < char >
- < string >

Control Flow Graph

Bytecode

# 1. Extract multiple representation from code

Identifiers

## Extraction

Pre-order visit of the AST  
Stream of AST Node Types

Abstract Syntax Tree

## Normalization

Remove AST node types:

- SimpleName
- QualifiedName

Control Flow Graph

Bytecode

# 1. Extract multiple representation from code

Identifiers

Abstract Syntax Tree

Control Flow Graph

Bytecode

## Extraction

Use Soot to extract CFG

### Graph:

- Nodes (statements)
- Directed edges (control flow)

# 1. Extract multiple representation from code

Identifiers

Abstract Syntax Tree

Control Flow Graph

Bytecode

## Extraction

`javap -c -private <classname>`  
Stream of bytecode mnemonic opcodes  
(e.g., iload, invokevirtual)

## Normalization

Remove references to constants  
`putfield #2 -> putfield`

# Embedding Learning

**AutoenCODE** NN Language Model + Recursive AutoEncoder (RvNN)

# AutoenCODE NN Language Model + Recursive AutoEncoder (RvNN)

$w_1$

$w_2$

$w_3$

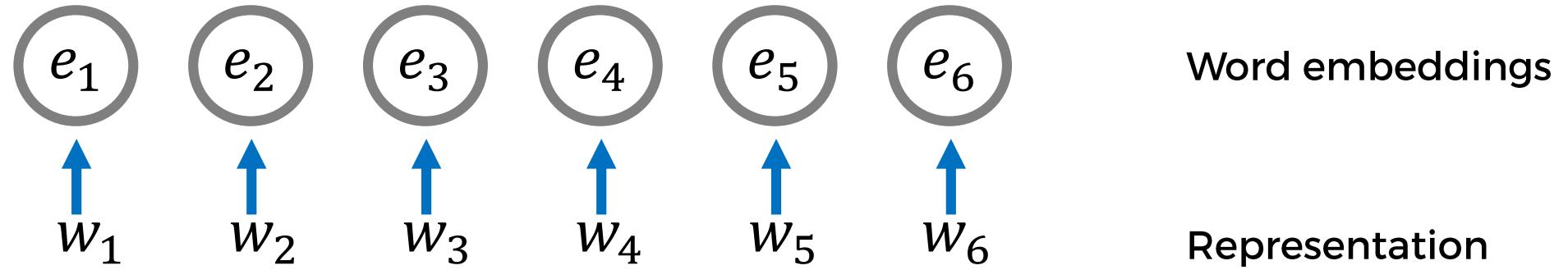
$w_4$

$w_5$

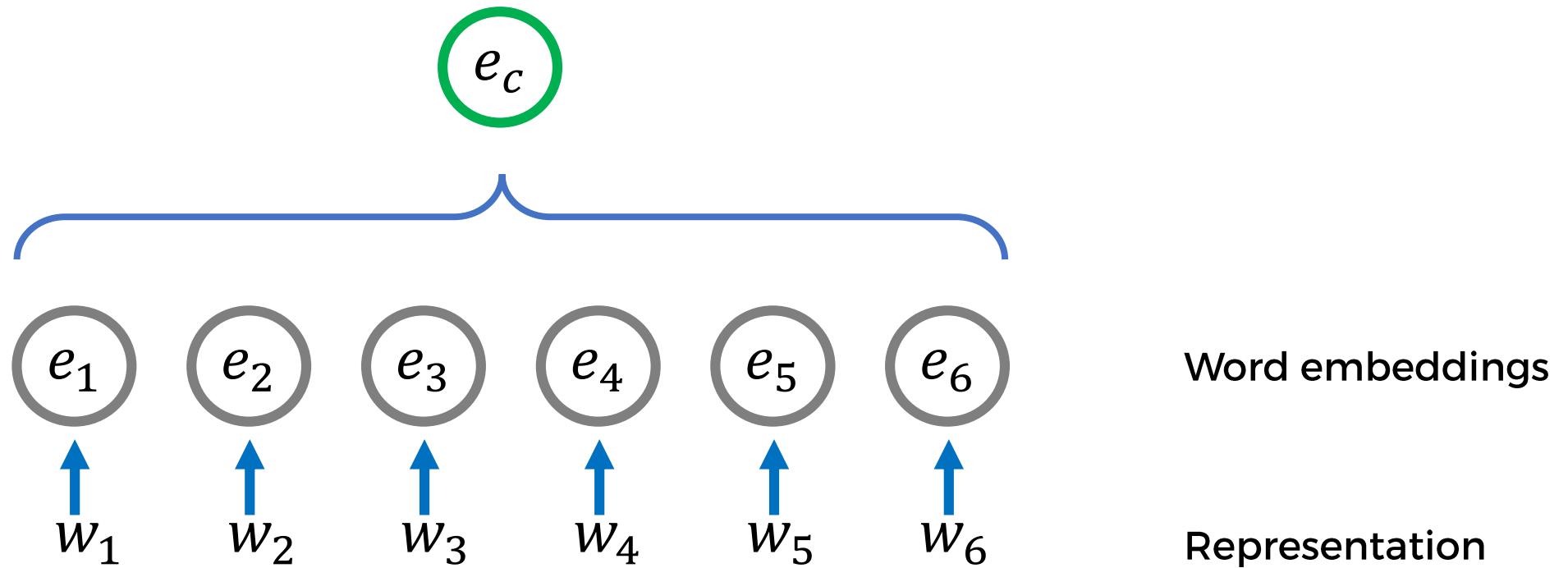
$w_6$

Representation

# AutoenCODE NN Language Model + Recursive AutoEncoder (RvNN)



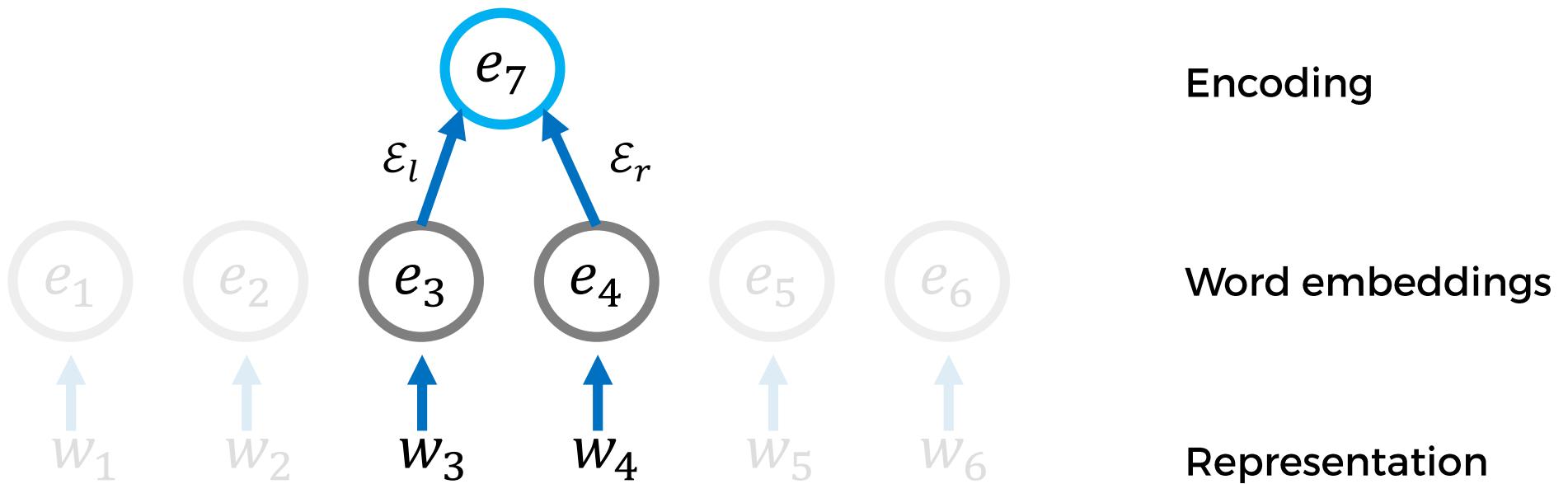
# AutoenCODE NN Language Model + Recursive AutoEncoder (RvNN)



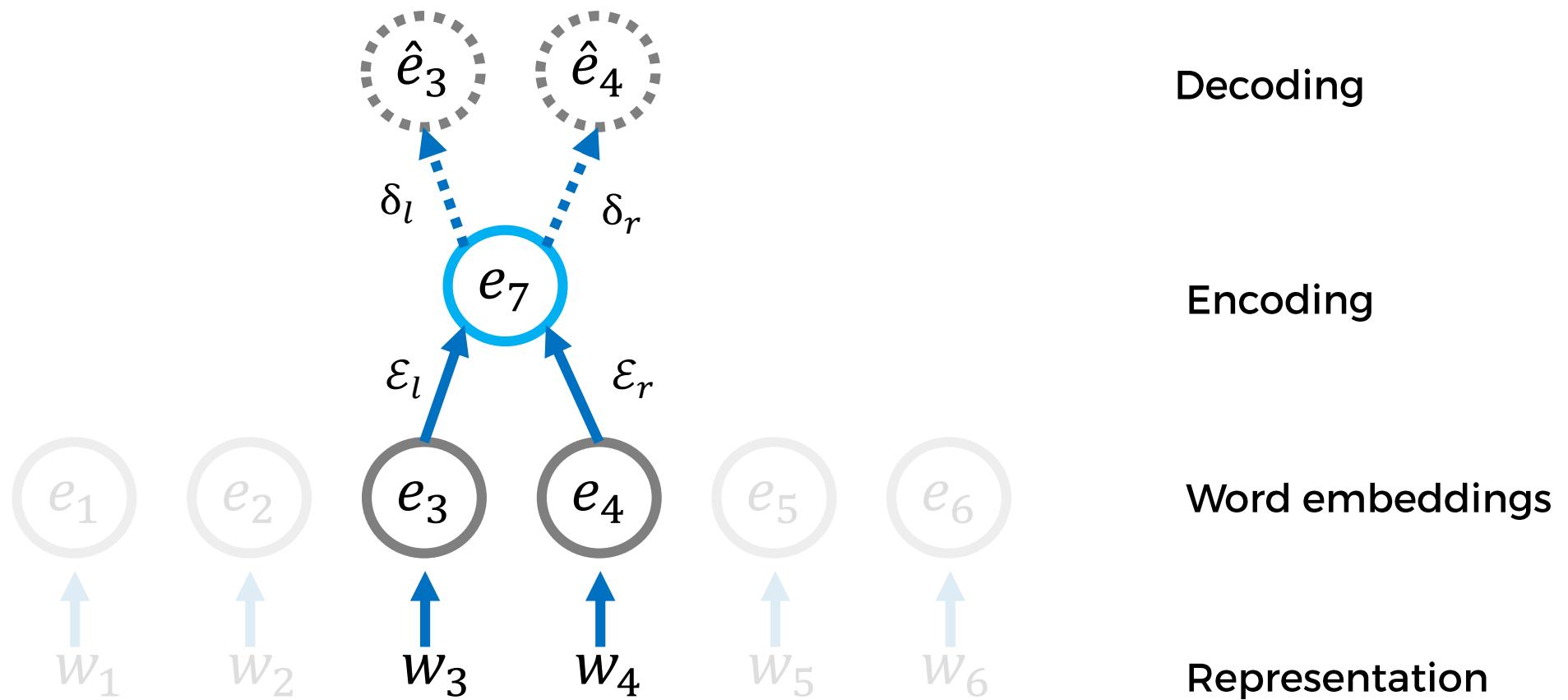
# AutoenCODE NN Language Model + Recursive AutoEncoder (RvNN)



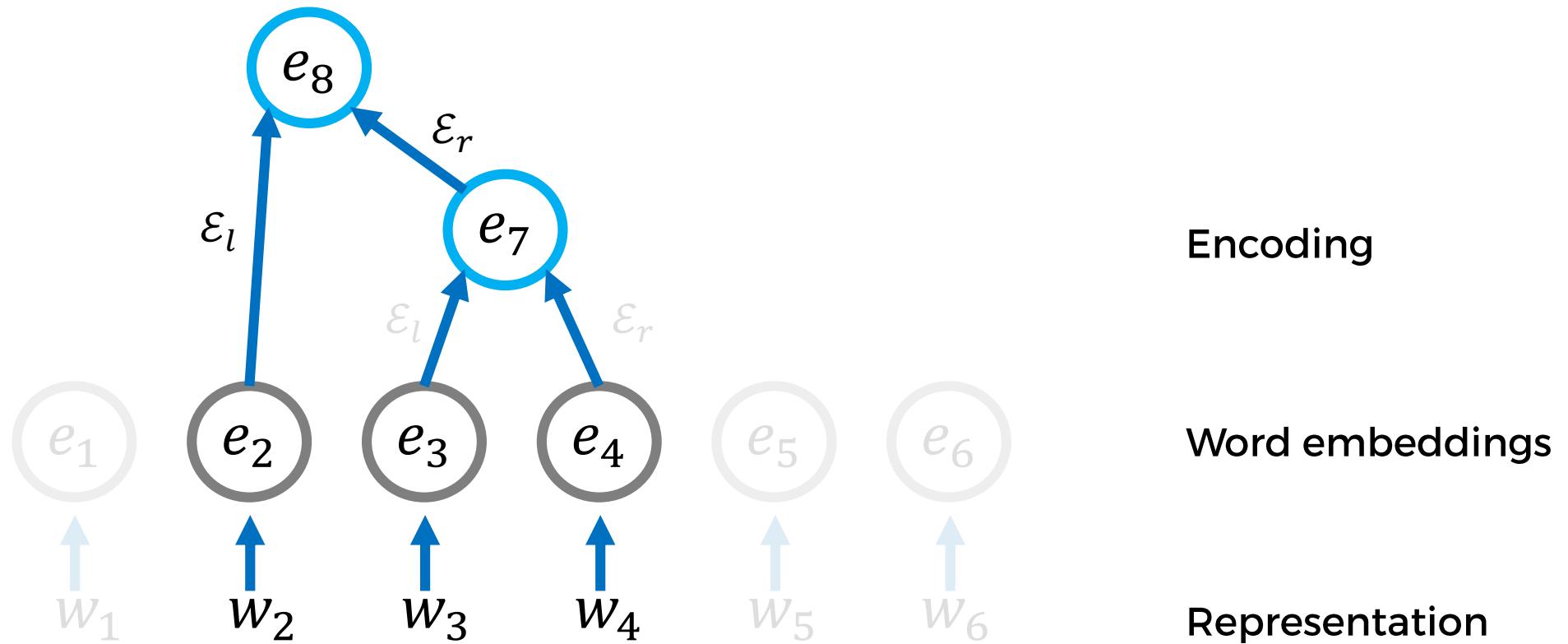
# AutoenCODE NN Language Model + Recursive AutoEncoder (RvNN)



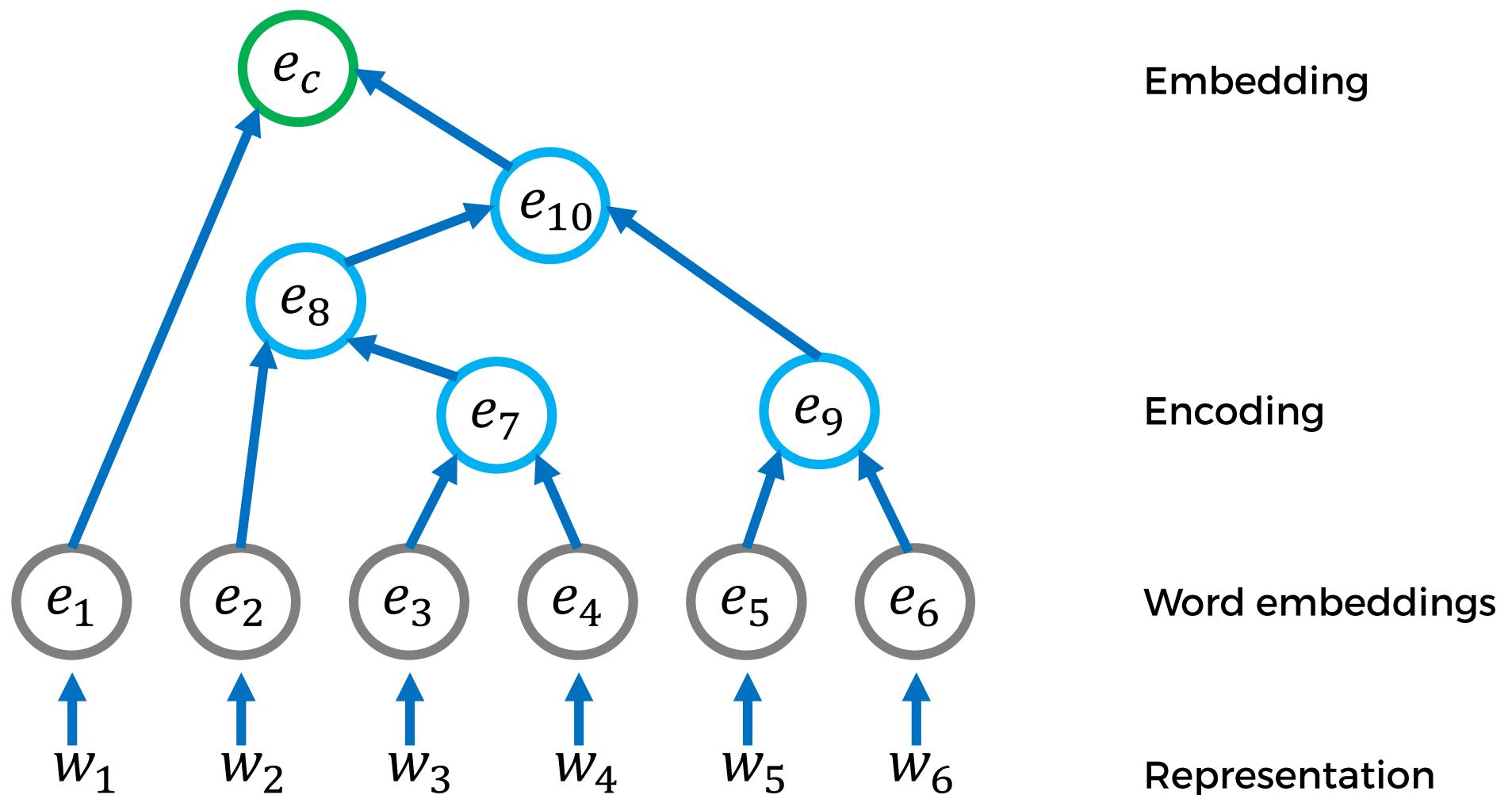
# AutoenCODE NN Language Model + Recursive AutoEncoder (RvNN)



# AutoenCODE NN Language Model + Recursive AutoEncoder (RvNN)



# AutoenCODE NN Language Model + Recursive AutoEncoder (RvNN)

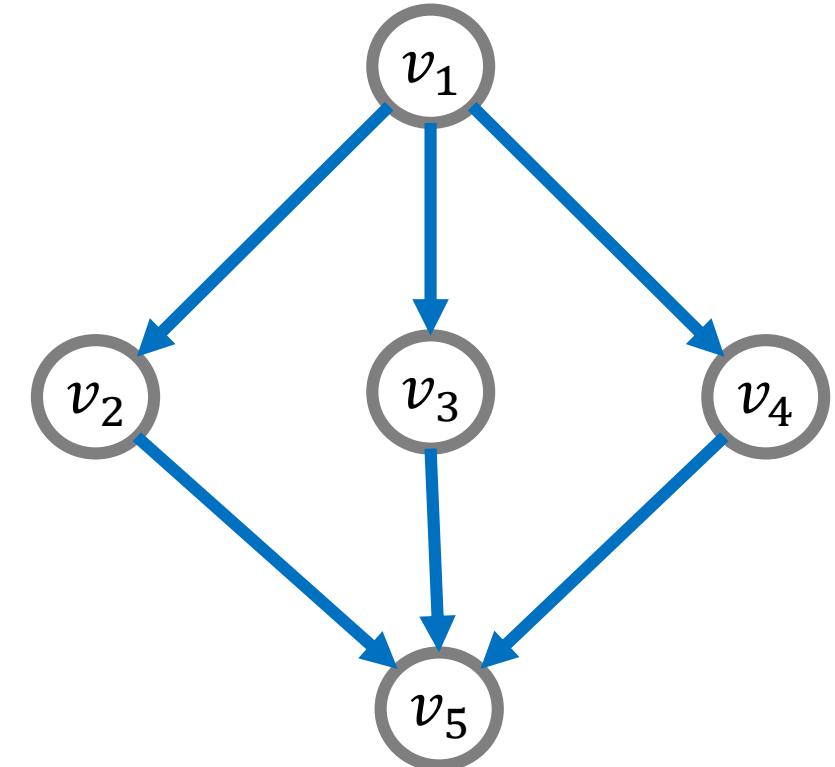


**HOPE**      Graph Embedding  
                  High-Order Proximity preserved Embedding

# HOPE

Graph Embedding  
High-Order Proximity preserved Embedding

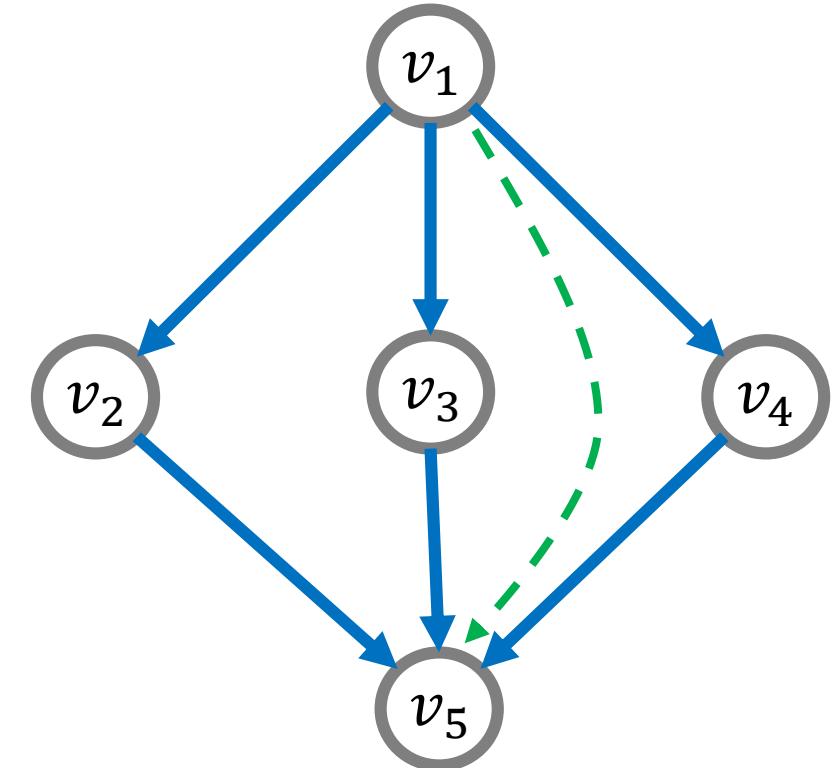
- Embeds directed graphs
- Graph reconstruction from embedding
- Asymmetric transitivity
- Katz proximity metric



# HOPE

Graph Embedding  
High-Order Proximity preserved Embedding

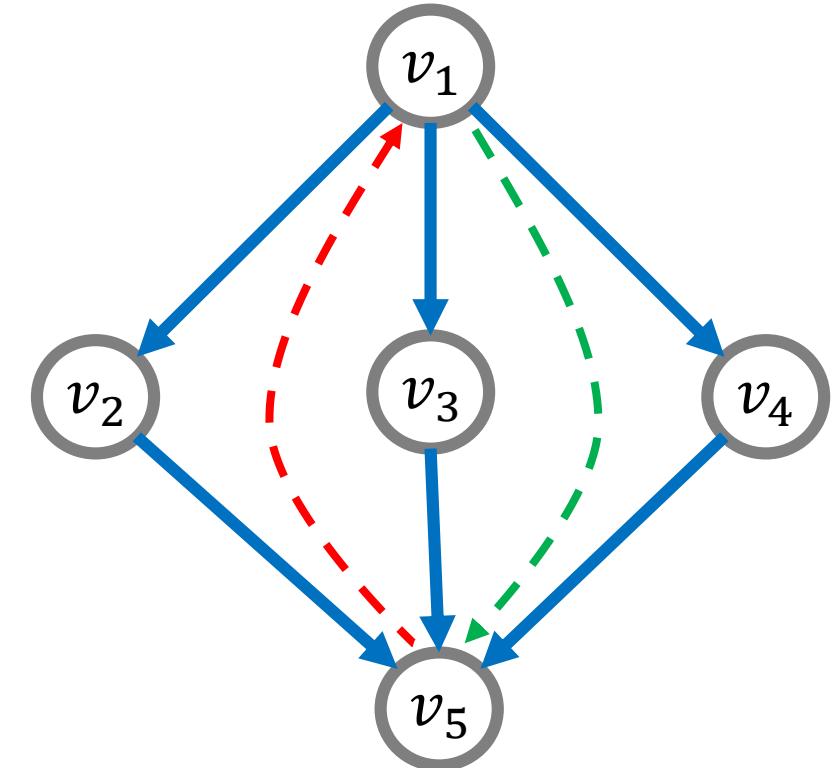
- Embeds directed graphs
- Graph reconstruction from embedding
- Asymmetric transitivity
- Katz proximity metric



# HOPE

Graph Embedding  
High-Order Proximity preserved Embedding

- Embeds directed graphs
- Graph reconstruction from embedding
- Asymmetric transitivity
- Katz proximity metric



# Combined Model



# Combined Model

Ensemble Learning (Random Forest)

Clone Detector

Clone Classifier



## Experimental Design & Results

## RQ1 How effective are different representations in detecting similar code fragments?

Dataset: 10 Java projects from *Qualitas.class Corpus*

Granularity: Methods and Classes

1. Extract code representations
2. Train representation-specific models
3. Generate embeddings
4. Compute similarities
5. Analyze candidates

## RQ1 How effective are different representations in detecting similar code fragments?

IDENT	AST	CFG	BYTECODE
F	F	F	T
F	F	T	F
F	F	T	T
F	T	F	F
F	T	F	T
F	T	T	F
F	T	T	T
T	F	F	F
T	F	F	T
T	F	T	F
T	F	T	T
T	T	F	F
T	T	F	T
T	T	T	F
T	T	T	T

## RQ1 How effective are different representations in detecting similar code fragments?

IDENT	AST	CFG	BYTECODE	Precision %	
				Methods	Classes
F	F	F	T	5	49
F	F	T	F	9	58
F	F	T	T	88	73
F	T	F	F	79	63
F	T	F	T	95	93
F	T	T	F	100	100
F	T	T	T	100	100
T	F	F	F	95	100
T	F	F	T	100	100
T	F	T	F	100	-
T	F	T	T	100	100
T	T	F	F	100	100
T	T	F	T	100	100
T	T	T	F	100	100
T	T	T	T	100	100

## RQ1 How effective are different representations in detecting similar code fragments?

### Methods

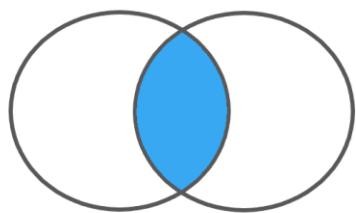
Representation	FP	TP	Type I	Type II	Type III	Type IV	Precision	Recall
IDENT	1	201	151	15	35	0	100	52
AST	11	292	138	132	19	3	96	75
CFG	43	178	69	81	19	9	81	46
BYTE	46	222	89	77	49	7	83	57

### Classes

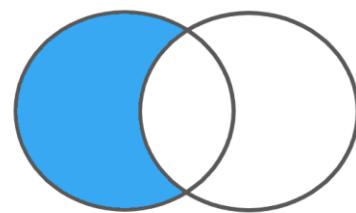
Representation	FP	TP	Type I	Type II	Type III	Type IV	Precision	Recall
IDENT	0	120	23	51	46	0	100	40
AST	18	188	18	121	44	5	91	63
CFG	24	120	7	65	41	7	83	40
BYTE	34	217	23	115	77	2	86	73

## RQ2 What is the **complementarity** of different representations?

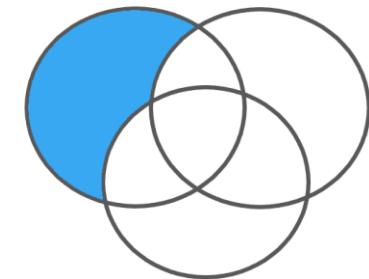
Intersection



Difference



Exclusive



$$R_i \cap R_j = \frac{|TP_{R_i} \cap TP_{R_j}|}{|TP_{R_i} \cup TP_{R_j}|} \%$$

$$R_i \setminus R_j = \frac{|TP_{R_i} \setminus TP_{R_j}|}{|TP_{R_i} \cup TP_{R_j}|} \%$$

$$EXC(R_i) = \frac{|TP_{R_i} \setminus \bigcup_{j \neq i} TP_{R_j}|}{|\bigcup_j TP_{R_j}|} \%$$

## RQ2 What is the **complementarity** of different representations?

Methods											
$R_1 \cap R_2$	Intersection %				Difference %				Exclusive %		
	Iden	AST	CFG	Byte	$R_1 \setminus R_2$	Iden	AST	CFG	Byte	$R_i$	$EXC(R_i)$
Iden		40	21	36	Iden		17	43	29	Iden	5% (21)
AST			42	44	AST		43	46	38	AST	9% (33)
CFG				36	CFG		36	12	24	CFG	1% (4)
Byte					Byte		35	18	39	Byte	1% (2)
Classes											
$R_1 \cap R_2$	Intersection %				Difference %				Exclusive %		
	Iden	AST	CFG	Byte	$R_1 \setminus R_2$	Iden	AST	CFG	Byte	$R_i$	$EXC(R_i)$
Iden		33	14	42	Iden		19	43	8	Iden	3% (8)
AST			31	51	AST		48	49	19	AST	9% (26)
CFG				34	CFG		43	20	14	CFG	7% (21)
Byte					Byte		49	30	52	Byte	7% (21)

## RQ2 What is the **complementarity** of different representations?

Methods											
$R_1 \cap R_2$	Intersection %				Difference %				Exclusive %		
	Iden	AST	CFG	Byte	$R_1 \setminus R_2$	Iden	AST	CFG	Byte	$R_i$	$EXC(R_i)$
Iden		40	21	36	Iden		17	43	29	Iden	5% (21)
AST			42	44	AST		43	46	38	AST	9% (33)
CFG				36	CFG		36	12	24	CFG	1% (4)
Byte					Byte		35	18	39	Byte	1% (2)
Classes											
$R_1 \cap R_2$	Intersection %				Difference %				Exclusive %		
	Iden	AST	CFG	Byte	$R_1 \setminus R_2$	Iden	AST	CFG	Byte	$R_i$	$EXC(R_i)$
Iden		33	14	42	Iden		19	43	8	Iden	3% (8)
AST			31	51	AST		48	49	19	AST	9% (26)
CFG				34	CFG		43	20	14	CFG	7% (21)
Byte					Byte		49	30	52	Byte	7% (21)

RQ3 How effective are **combined** multi-representation models?

Ensemble Learning (Random Forest)

Clone Detector

Clone Classifier



## RQ3 How effective are **combined** multi-representation models?

### Clone Detector

	Methods			Classes		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Clone	98	97	98	90	93	91
Not Clone	90	91	90	61	52	56

### Clone Classifier

	Methods			Classes		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Not Clone	89	94	91	59	61	60
Type I	89	88	88	86	78	82
Type II	82	84	83	81	85	83
Type III	74	75	75	61	59	60
Type IV	67	18	29	0	0	0
Weighted Avg.	84	84	84	67	68	68

## RQ3 How effective are **combined** multi-representation models?

### Clone Detector

	Methods			Classes		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Clone	98	97	98	90	93	91
Not Clone	90	91	90	61	52	56

### Clone Classifier

	Methods			Classes		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Not Clone	89	94	91	59	61	60
Type I	89	88	88	86	78	82
Type II	82	84	83	81	85	83
Type III	74	75	75	61	59	60
Type IV	67	18	29	0	0	0
Weighted Avg.	84	84	84	67	68	68

## RQ3 How effective are **combined** multi-representation models?

### Clone Detector

	Methods			Classes		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Clone	98	97	98	90	93	91
Not Clone	90	91	90	61	52	56

### Clone Classifier

	Methods			Classes		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Not Clone	89	94	91	59	61	60
Type I	89	88	88	86	78	82
Type II	82	84	83	81	85	83
Type III	74	75	75	61	59	60
Type IV	67	18	29	0	0	0
Weighted Avg.	84	84	84	67	68	68

## RQ3 How effective are **combined** multi-representation models?

### Clone Detector

	Methods			Classes		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Clone	98	97	98	90	93	91
Not Clone	90	91	90	61	52	56

### Clone Classifier

	Methods			Classes		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Not Clone	89	94	91	59	61	60
Type I	89	88	88	86	78	82
Type II	82	84	83	81	85	83
Type III	74	75	75	61	59	60
Type IV	67	18	29	0	0	0
Weighted Avg.	84	84	84	67	68	68

## RQ4 Are DL-based models applicable for detecting clones among **different** projects?

### Scenarios:

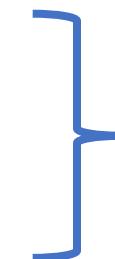
- Software Maintainer has to analyze the amount of duplicated code across projects belonging to their organization
- Developer using a jar file (compiled library) needs to asses provenance and/or licensing issues before releasing the code

### Dataset:

- 46 compiled Apache Commons libraries

## RQ4 Are DL-based models applicable for detecting clones among **different** projects?

- Software Maintainer has to analyze the amount of duplicated code across projects belonging to their organization
  - lang3-3.6 - text-1.1
  - text-1.1 - collections4-4.1
  - math3-3.6.1 - rng-1.0
  - codec-1.9 - net-3.6
- Developer using a jar file (compiled library) needs to asses provenance and/or licensing issues before releasing the code



Share Duplicated Code

weaver-1.3 imported and shaded:

- collections4-4.1 (373 classes)
- lang3-3.6 (79 classes)
- io-2.5 (13 classes)

**RQ5** Can trained DL-based models be **reused** on different, previously unseen projects?

## Model Reusability and Transfer Learning

Limited Vocabulary: AST, Bytecode, CFG

1. Train model on project A
2. Evaluate model on project B
3. Compare the candidates with original model

## RQ5 Can trained DL-based models be reused on different, previously unseen projects?

AST model trained on lucene, evaluated on other 9 projects

Project	Methods %		Classes %	
	$L_R \in L_O$	$L_O \in L_R$	$L_R \in L_O$	$L_O \in L_R$
ant-1.8.2	99	88	73	31
antlr-3.4	100	100	33	100
argouml-0.34	99	96	97	73
hadoop-1.1.2	99	95	95	74
hibernate-4.2.0	89	82	30	84
jhotdraw-7.5.1	99	98	82	77
maven-3.0.5	97	84	50	100
pmd-4.2.5	97	99	99	99
tomcat-7.0.2	98	97	87	69
Overall	97	93	58	90

## RQ5 Can trained DL-based models be reused on different, previously unseen projects?

AST model trained on lucene, evaluated on other 9 projects

Project	Methods %		Classes %	
	$L_R \in L_O$	$L_O \in L_R$	$L_R \in L_O$	$L_O \in L_R$
ant-1.8.2	99	88	73	31
antlr-3.4	100	100	33	100
argouml-0.34	99	96	97	73
hadoop-1.1.2	99	95	95	74
hibernate-4.2.0	89	82	30	84
jhotdraw-7.5.1	99	98	82	77
maven-3.0.5	97	84	50	100
pmd-4.2.5	97	99	99	99
tomcat-7.0.2	98	97	87	69
Overall	97	93	58	90

## RQ5 Can trained DL-based models be reused on different, previously unseen projects?

AST model trained on lucene, evaluated on other 9 projects

Project	Methods %		Classes %	
	$L_R \in L_O$	$L_O \in L_R$	$L_R \in L_O$	$L_O \in L_R$
ant-1.8.2	99	88	73	31
antlr-3.4	100	100	33	100
argouml-0.34	99	96	97	73
hadoop-1.1.2	99	95	95	74
hibernate-4.2.0	89	82	30	84
jhotdraw-7.5.1	99	98	82	77
maven-3.0.5	97	84	50	100
pmd-4.2.5	97	99	99	99
tomcat-7.0.2	98	97	87	69
Overall	97	93	58	90

# Conclusions

Learn from **available** representations



# Conclusions

- Learn from **available** representations

Combine **multiple** representations



# Conclusions

- Learn from **available** representations
- Combine **multiple** representations

**Reuse** models on different projects



**Open Science**

# Open Science

## Data

Deep Learning Code Similarities

Home Source Code Datasets Representations RQ1 RQ2 RQ3 RQ4

### Deep Learning Similarities from Different Representations of Source Code

MSR 2018 - 15th International Conference on Mining Software Repositories , May 28-29, 2018, Gothenburg, Sweden

Authors: Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk

#### Abstract

Assessing the similarity between code components plays a pivotal role in a number of Software Engineering (SE) tasks, such as clone detection, impact analysis, refactoring, etc. Code similarity is generally measured by relying on manually defined or hand-crafted features (e.g., by analyzing the overlap among identifiers or comparing the Abstract Syntax Trees of two code components). These features represent a best guess at what SE researchers can utilize to exploit and reliably assess code similarity for a given task. Recent work has shown, when using a stream of identifiers to represent the code, that Deep Learning (DL) can effectively replace manual feature engineering for the task of clone detection. However, source code can be represented at different levels of abstraction: identifiers, Abstract Syntax Trees, Control Flow Graphs, and bytecode. We



<https://sites.google.com/view/learningcodesimilarities>

# Open Science

## Data

The screenshot shows a Google Sheets document with the title "Deep Learning Code Similarities". The abstract section contains the following text:

MSR 2018 - 15th International Conference on Mining Software Repositories , May 28-29, 2018, Gothenburg, Sweden  
Authors: Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk

**Abstract**  
Assessing the similarity between code components plays a pivotal role in a number of Software Engineering (SE) tasks, such as clone detection, impact analysis, refactoring, etc. Code similarity is generally measured by relying on manually defined or hand-crafted features (e.g., by analyzing the overlap among identifiers or comparing the Abstract Syntax Trees of two code components). These features represent a best guess at what SE researchers can utilize to exploit and reliably assess code similarity for a given task. Recent work has shown, when using a stream of identifiers to represent the code, that Deep Learning (DL) can effectively replace manual feature engineering for the task of clone detection. However, source code can be represented at different levels of abstraction: identifiers, Abstract Syntax Trees, Control Flow Graphs, and bytecode. We



<https://sites.google.com/view/learningcodesimilarities>

## Source Code



**AutoenCODE**



<https://github.com/micheletufano/AutoenCODE>

# Open Science

## Data

The screenshot shows a presentation slide titled "Deep Learning Similarities from Different Representations of Source Code". At the top, there's a navigation bar with links: Home, Source Code, Datasets, Representations, RQ1, RQ2, RQ3, and RQ4. Below the title, it says "MSR 2018 - 15th International Conference on Mining Software Repositories, May 28-29, 2018, Gothenburg, Sweden". It also lists authors: Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. A large blue box contains the abstract text, which discusses the use of deep learning for code clone detection and refactoring.



## Source Code



## AutoenCODE

**Michele Tufano**  
@tufanomichele  
<http://www.cs.wm.edu/~mtufano/>



<https://sites.google.com/view/learningcodesimilarities>

<https://github.com/micheletufano/AutoenCODE>