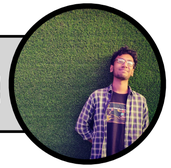
Designing DL-based Models for Software Development Tasks

Dr. Antonio Mastropaolo

Instructor

Mr. Alvi Haque



Teaching Assistant

WILLIAM & MARY

CHARTERED 1693

Spring 2026

Designing DL-based Models for Software Development Tasks

Learning from **Existing** Data

Huge amount of existing code (big code)

Programs are regular and repetitive

DL can learn "implementation patterns" from millions of examples

.. complete partial code
.. use an API
.. fix programming errors

Representing Programs

Designing DL-based Models for Software Development Tasks

Representing Programs

Sequence of characters

Sequence of tokens

Abstract Syntax Tree (AST)

Control Flow Graph (CFG)

etc.

Designing DL-based Models for Software Development Tasks

Representing Programs

Sequence of characters

Sequence of tokens

Abstract Syntax Tree (AST)

Control Flow Graph (CFG)

etc.

Sequence of Tokens

Using a Tokenizer

Splits sequence of characters into subsequences called tokens

E.g., for Java:

Identifiers, e.g., MyClass
Keywords, e.g., if
Separators, e.g., . or {
*Operators, e.g., * or ++*
Literals, e.g., 23 or “hi”
Comments, e.g., / bla */*

Sequence of Tokens

```
If (flag == true){
  name = “antonio”;
}
```

Designing DL-based Models for Software Development Tasks

Sequence of Tokens

```
If (flag == true){
  name = “antonio”;
}
```

```
If ( flag == true ){
  name = “Antonio” ;
}
```

keywords
separators
operators
identifiers
literals

Designing DL-based Models for Software Development Tasks

Sequence of Tokens

```
If (flag == true){
  name = “antonio”;
}
```

```
If ( flag == true ){
  name = “Antonio” ;
}
```

If (flag == true){ name = “Antonio” ; }

keywords
separators
operators
identifiers
literals

Abstract Syntax Tree

Tree representation of source code

Abstract because some details of syntax are omitted (e.g., { in Java)

Nodes: construct in source code

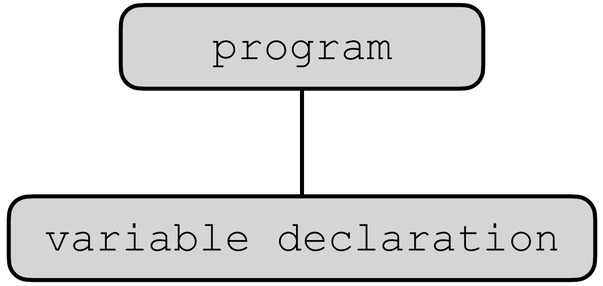
Edges: parent-child relationship

Abstract Syntax Tree

```
var x = 6 * y;
```

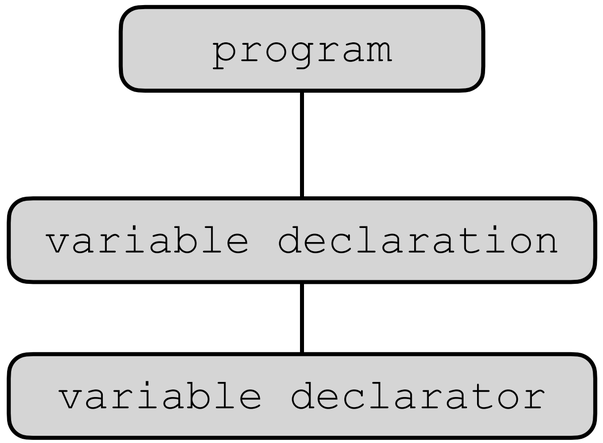
Abstract Syntax Tree

```
var x = 6 * y;
```



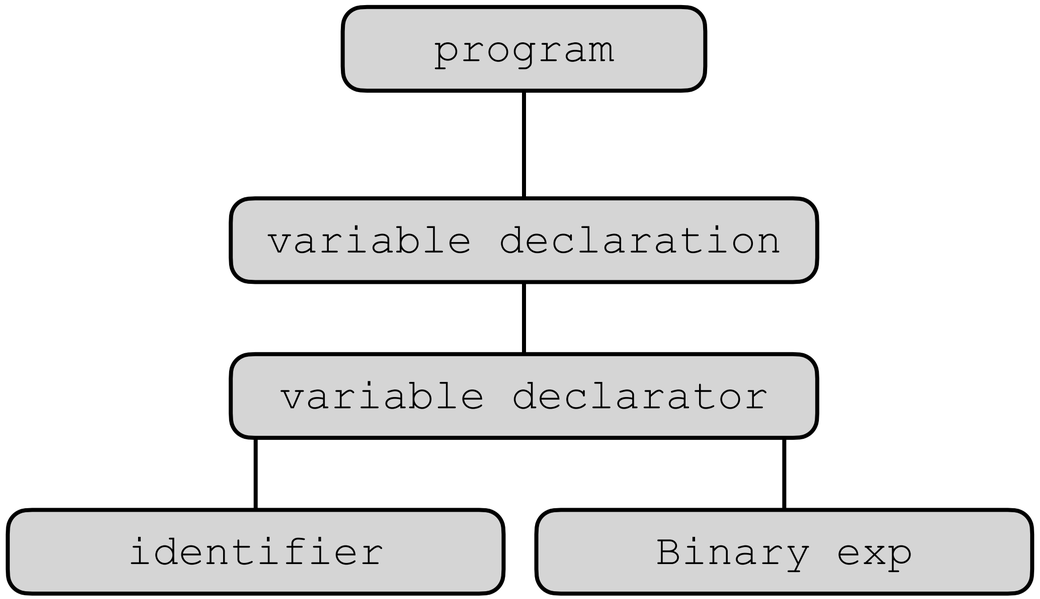
Abstract Syntax Tree

```
var x = 6 * y;
```



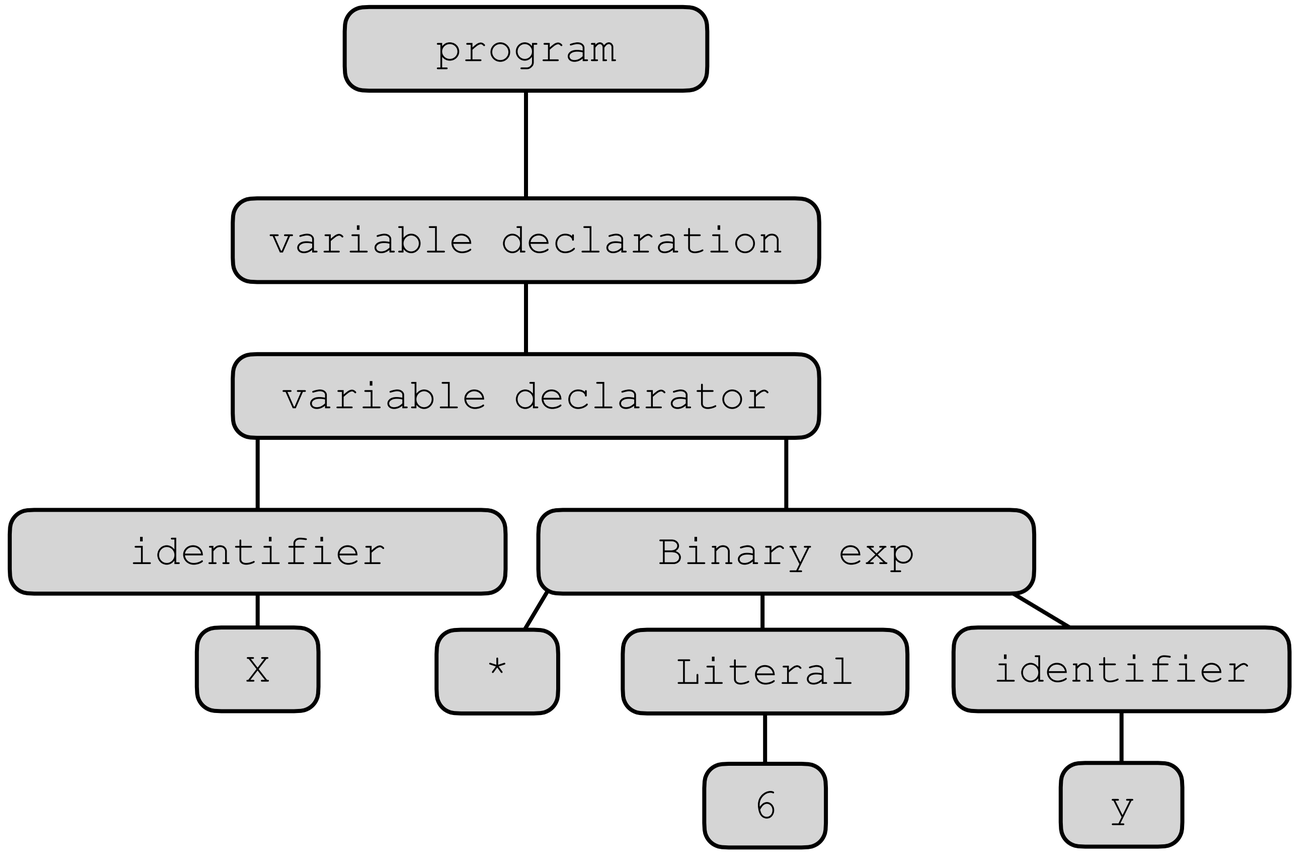
Abstract Syntax Tree

```
var x = 6 * y;
```



Abstract Syntax Tree

```
var x = 6 * y;
```



Control Flow Graph

Models flow of control through a program

Graph (N, E) with:

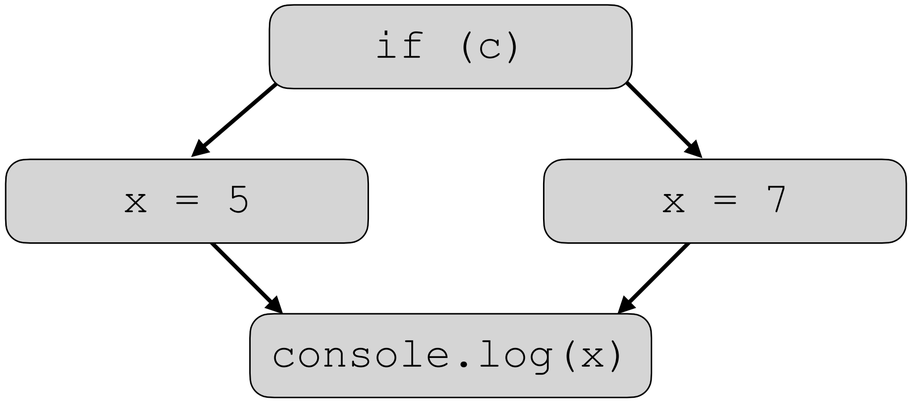
- *N nodes representing basic blocks which are always executed together*
- *E edges representing possible transfer of control*

Control Flow Graph

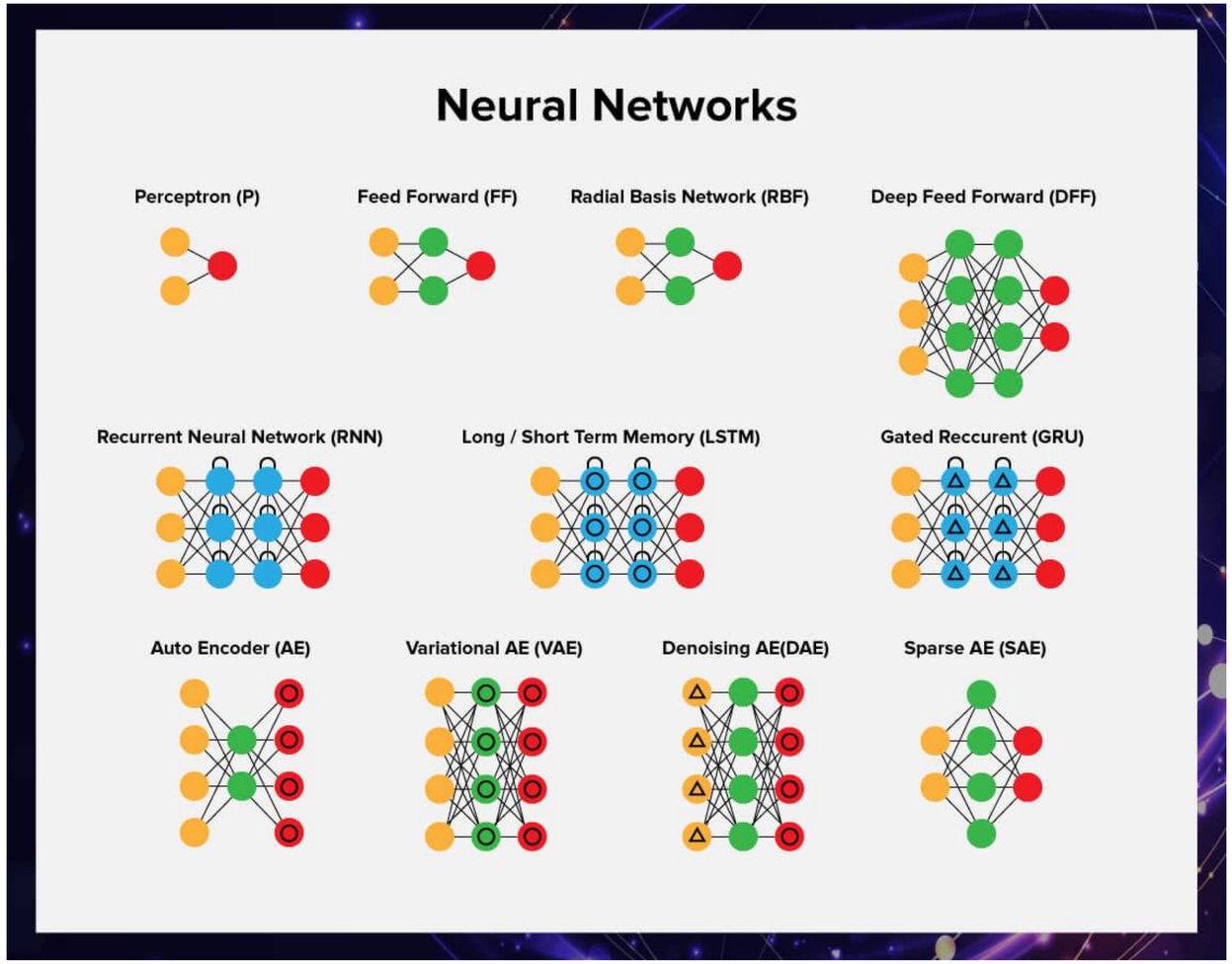
```
if (c){
  x = 5;
} else {
  x = 7;
}
console.log(x);
```

Control Flow Graph

```
if (c){
  x = 5;
} else {
  x = 7;
}
console.log(x);
```

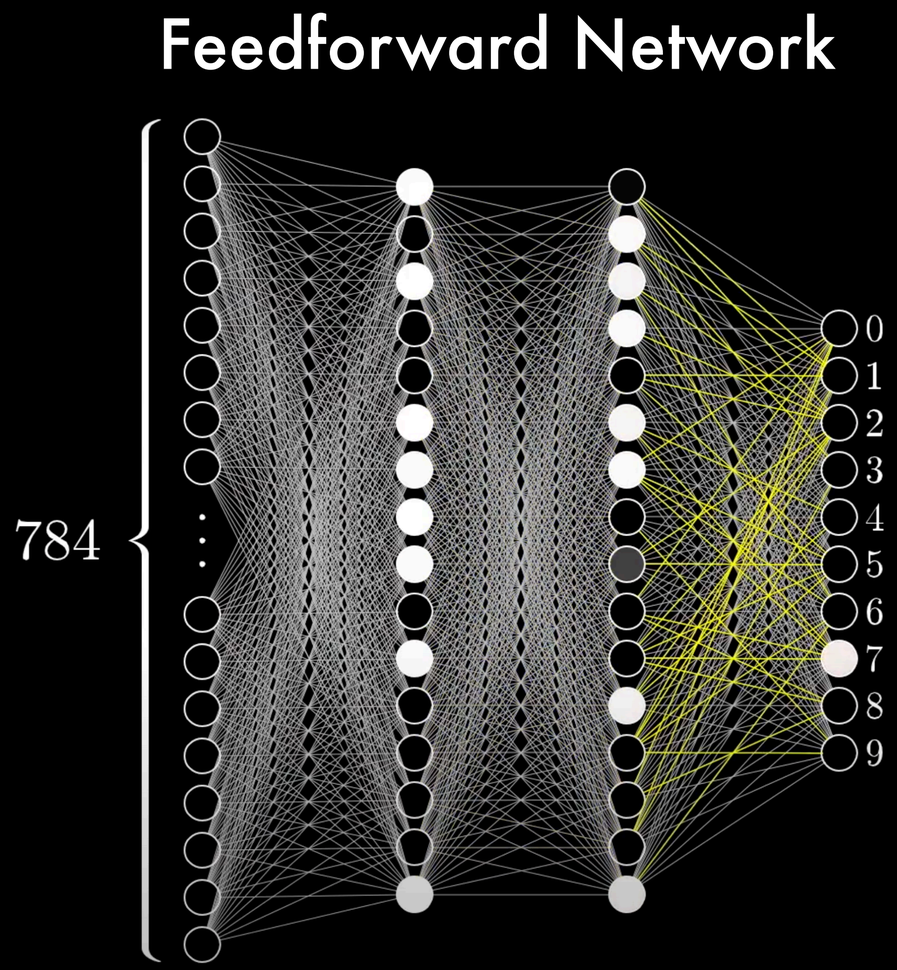


Designing DL-based Models for Software Development Tasks



https://serokell.io/blog/deep-learning-and-neural-network-guide

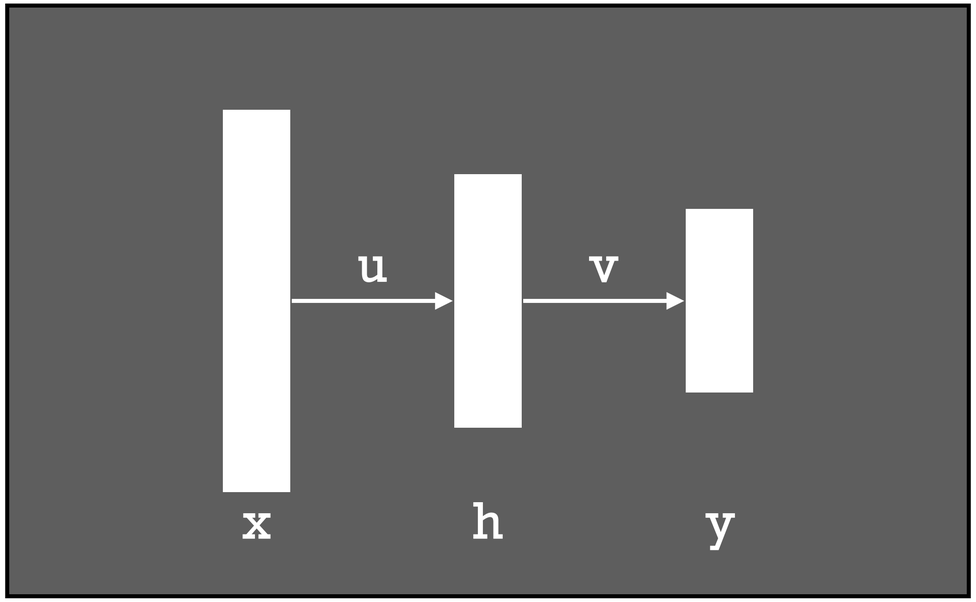
Designing DL-based Models for Software Development Tasks



Designing DL-based Models for Software Development Tasks

Feedforward Network

Feedforward Network



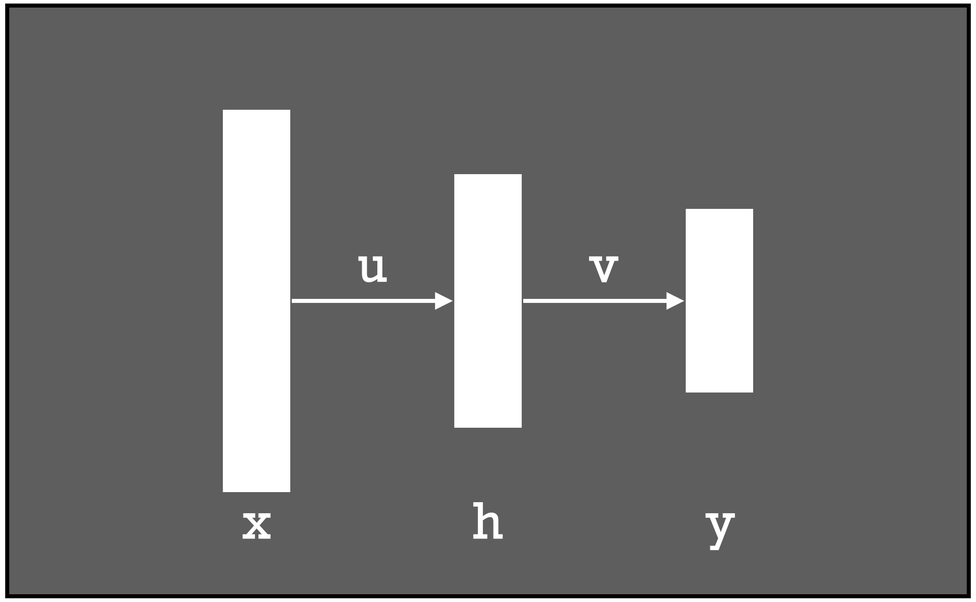
x = input layer
h = hidden layer
y = output layer
u, v = weight matrices

Designing DL-based Models for Software Development Tasks

AI 4 S/ENG is …

(cool)

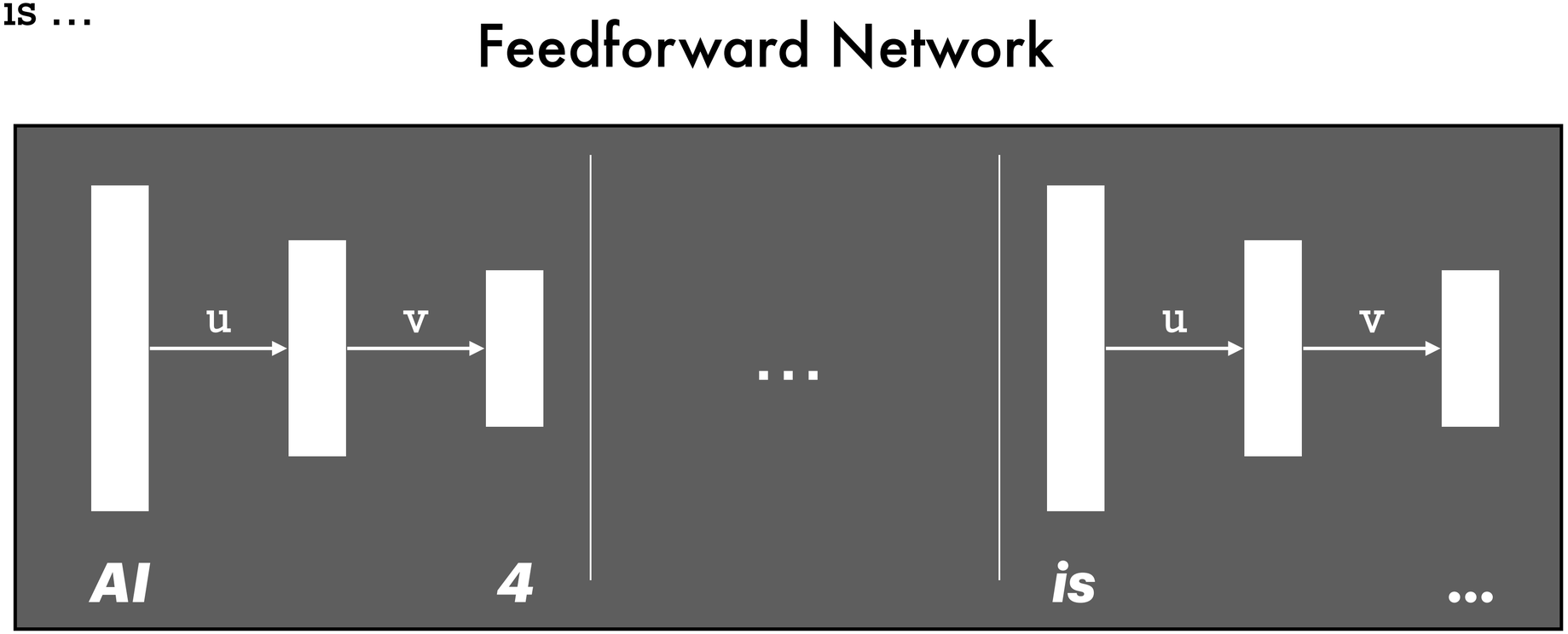
Feedforward Network



Designing DL-based Models for Software Development Tasks

AI 4 S/ENG is …

(cool)



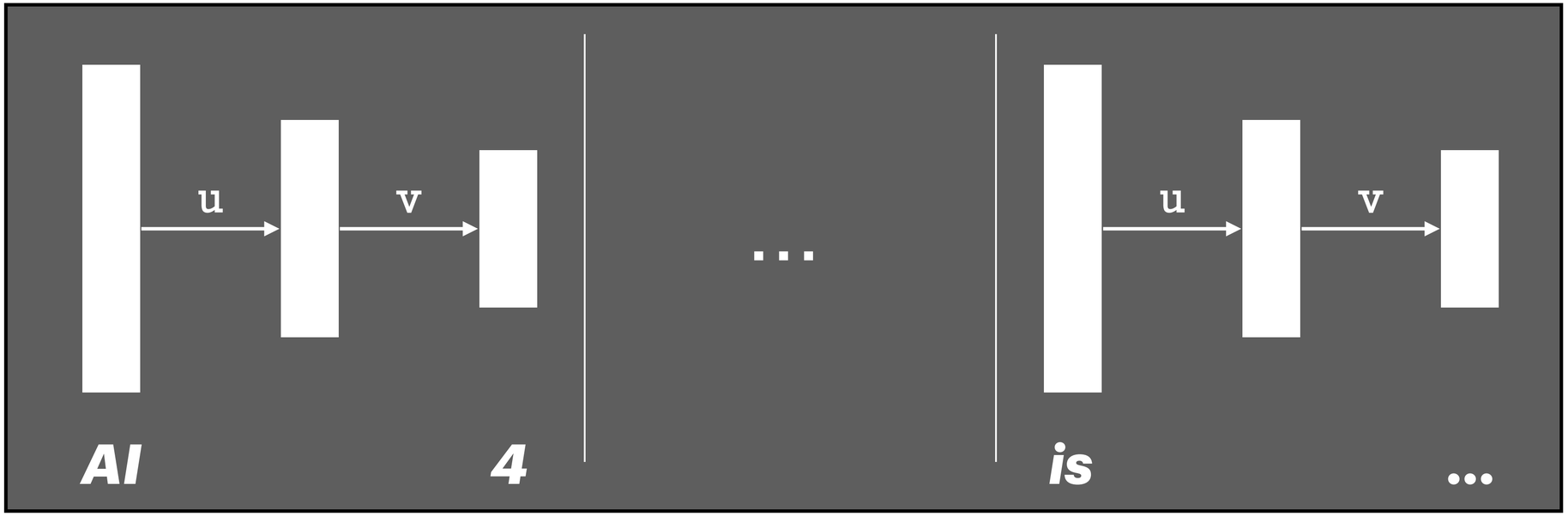
Designing DL-based Models for Software Development Tasks

AI 4 S/ENG is …

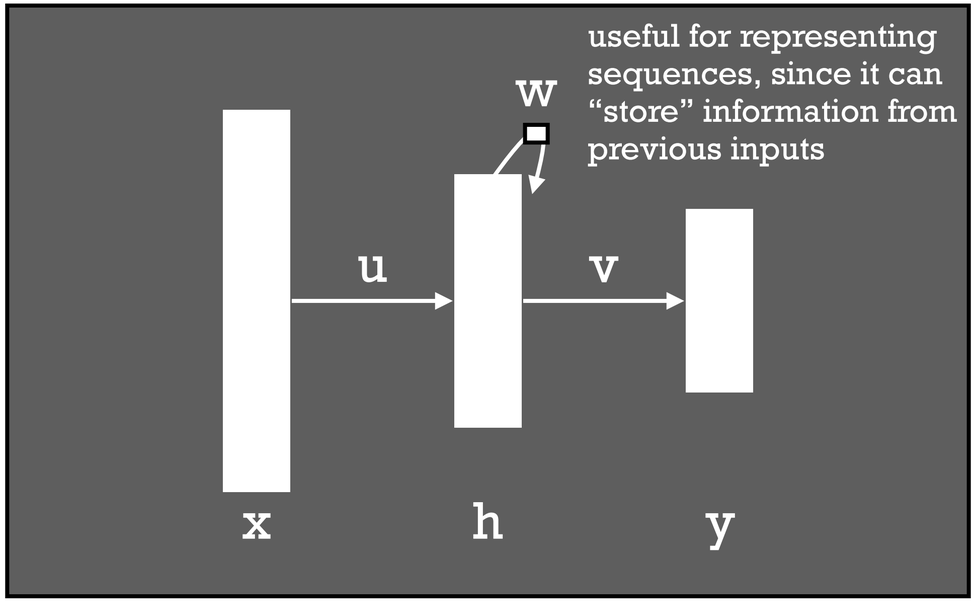
(cool)

UNIGRAM

Feedforward Network



Recurrent Neural Network (RNN)

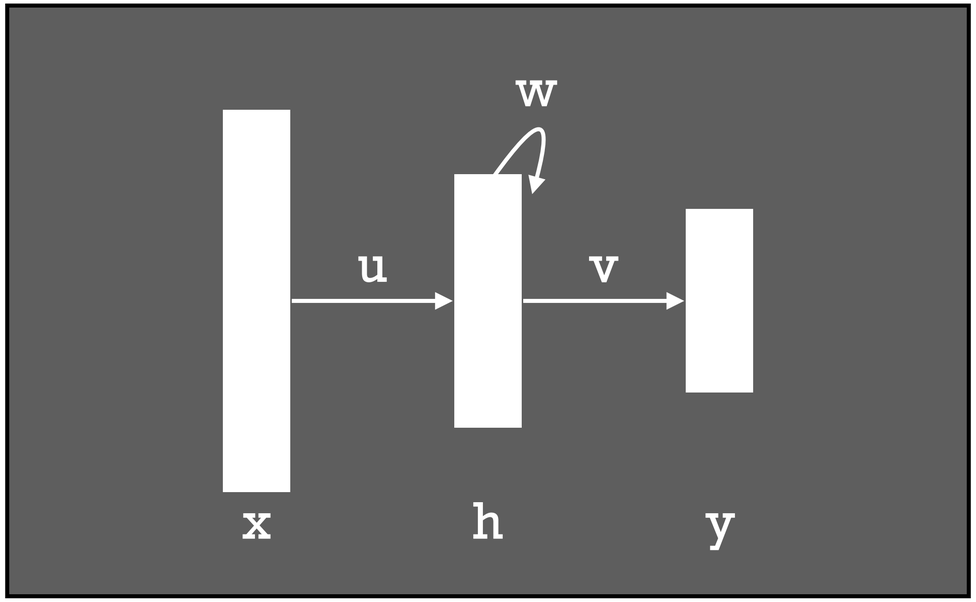


Designing DL-based Models for Software Development Tasks

AI 4 S/ENG is …

(cool)

RNN



Designing DL-based Models for Software Development Tasks

AI 4 S/ENG is …

(cool)

RNN

Designing DL-based Models for Software Development Tasks

AI 4 S/ENG is …

(cool)

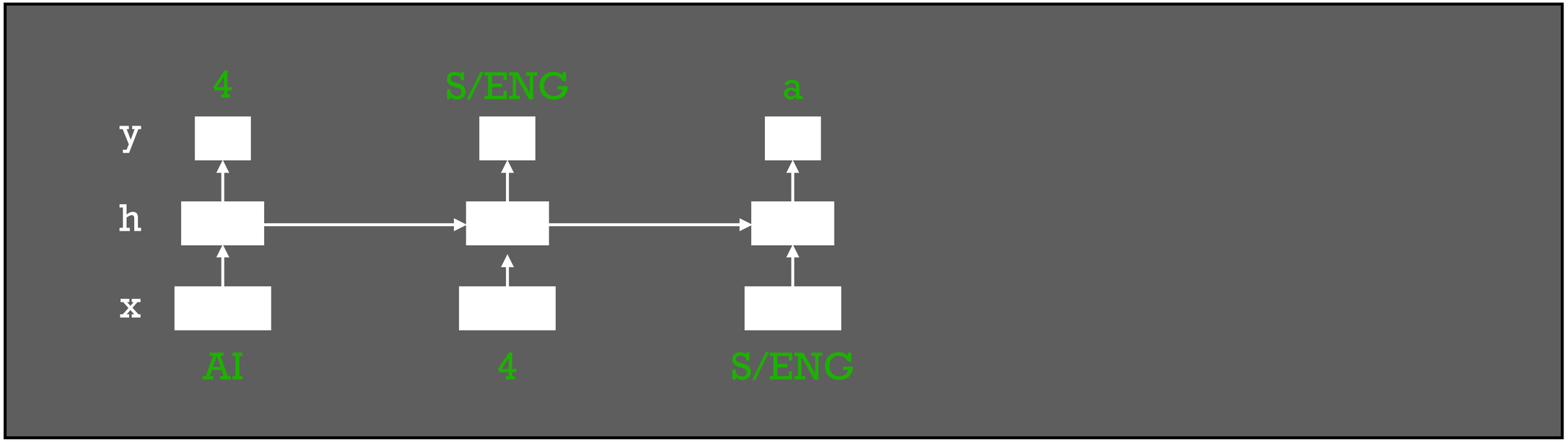
RNN

Designing DL-based Models for Software Development Tasks

AI 4 S/ENG is …

(cool)

RNN

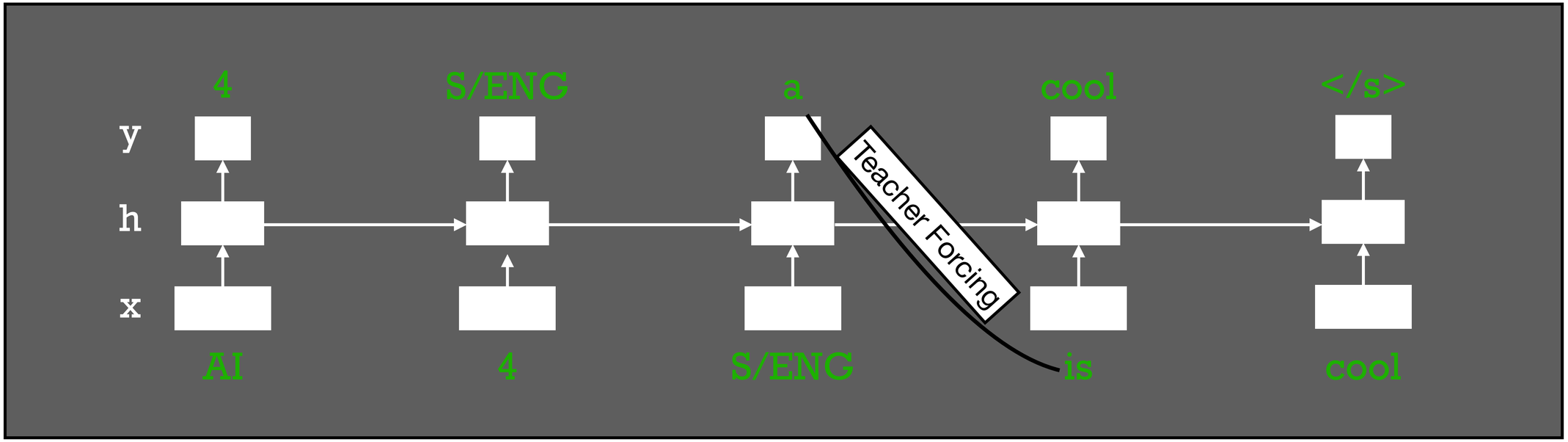


Designing DL-based Models for Software Development Tasks

AI 4 S/ENG is …

(cool)

RNN



RNN-based software analysis

https://serokell.io/blog/deep-learning-and-neural-network-guide

Designing DL-based Models for Software Development Tasks

Given a partial program (i.e., the code has not been fully written), we can model two different code completion scenarios and generate suitable code to fill the holes:

Designing DL-based Models for Software Development Tasks

Given a partial program (i.e., the code has not been fully written), we can model two different code completion scenarios and generate suitable code to fill the holes:

1) Full-context (infilling / fill-in-the-blank) completion: the model is given both the prefix and the suffix of the program (including the end of the method/file) and must synthesize the missing code that fits between them.

1

Designing DL-based Models for

1

Given a partial program … odel two different code completion scenarios an…

1) Full-context (infilling / … the prefix and the suffix of the program (including the end of the method/file) and must synthesize the missing code that fits between them.

1

```
def normalize_and_join(words):
    """
    …
    """
    result = []
    for w in words:
        w = w.strip().lower()
        if not w:
            continue
        result.append(w)
    return
```

Ground Truth

```
def normalize_and_join(words):
    """
    …
    """
    result = []
    # >>> FILL-IN-THE-BLANK START
    # <<< FILL-IN-THE-BLANK END
    return "-".join(result)
```

Designing DL-based Models for Software Development Tasks

Given a partial program (i.e., the code has not been fully written), we can model two different code completion scenarios and generate suitable code to fill the holes:

1) Full-context (infilling / fill-in-the-blank) completion: the model is given both the prefix and the suffix of the program (including the end of the method/file) and must synthesize the missing code that fits between them. 1

2) Autoregressive (left-to-right) completion: the model is given only the prefix , in other words (the code written so far) and must continue generating code forward to complete the program. 2

Designing DL-based Models for

2

Given a partial program … odel two different code completion scenarios an…

1) Full-context (infilling / … the prefix and the suffix of the program (including the end of the method/file) and must synthesize the missing code that fits between them.

1

2) Autoregressive (left-to… , in other words (the code written so far) and … the program.

2

```
def normalize_and_join(words):
    """
    …
    """
    result = []
    for w in words:
        w = w.strip().lower()
        if not w:
            continue
        result.append(w)
    return
```

Ground Truth

```
def normalize_and_join(words):
    """
    …
    """
    result = []
    for w in words:
        w = w.strip().lower()
        # >>> GENERATE TOKENS FOR COMPLETIONS
```

Designing DL-based Models for Software Development Tasks

Pythia: AI-assisted Code Completion System

Alexey Svyatkovskiy
Microsoft
Redmond, WA
alsvyatk@microsoft.com

Ying Zhao
Microsoft
Redmond, WA
zhying@microsoft.com

Shengyu Fu
Microsoft
Redmond, WA
shengyfu@microsoft.com

Neel Sundaresan
Microsoft
Redmond, WA
neels@microsoft.com

ABSTRACT

In this paper, we propose a novel end-to-end approach for AI-assisted code completion called Pythia. It generates ranked lists of method and API recommendations which can be used by software developers at edit time. The system is currently deployed as part of Intellicode extension in Visual Studio Code IDE. Pythia exploits state-of-the-art large-scale deep learning models trained on code contexts extracted from abstract syntax trees. It is designed to work at a high throughput predicting the best matching code completions on the order of 100 *ms*.

We describe the architecture of the system, perform comparisons to frequency-based approach and invocation-based Markov Chain language model, and discuss challenges serving Pythia models on lightweight client devices.

The offline evaluation results obtained on 2700 Python open source software GitHub repositories show a top-5 accuracy of 92%, surpassing the baseline models by 20% averaged over classes, for both intra and cross-project settings.

CCS CONCEPTS

• **Software and its engineering** → **Integrated and visual development environments**; • **Computing methodologies** → *Neural networks.*

KEYWORDS

Code completion, neural networks, naturalness of software

1 INTRODUCTION

In software development through Integrated Development Environments (IDEs) [13], code completion is one of the most widely used features. Intelligent code completion [14] assists developers by reducing typographic and other common errors, effectively improving developer productivity. These features may include pop-ups when typing, calling methods and APIs on typed classes and objects, querying parameters of functions, variable or function name disambiguation, and program structure completion that use code context to reliably predict following code.

Traditional code completion tools in IDEs typically list out all possible attributes or methods that can be invoked when a user types a "." or an "=" trigger character. However, not ranking the suggested results requires users to scroll through a long alphabetically ordered list, which is often even slower than typing the full name of a method directly. Studies show that developers tend to rely heavily on prefix filtering to reduce the number of choices [10].

In this paper, we introduce Pythia – a neural code completion system, trained on code snippets extracted from a large-scale open source code dataset. The fundamental task of the system is to find the most likely method given a code snippet. In other words, given original code snippet C, the vocabulary V, and the set of all possible methods $M \subset V$, we would like to determine:

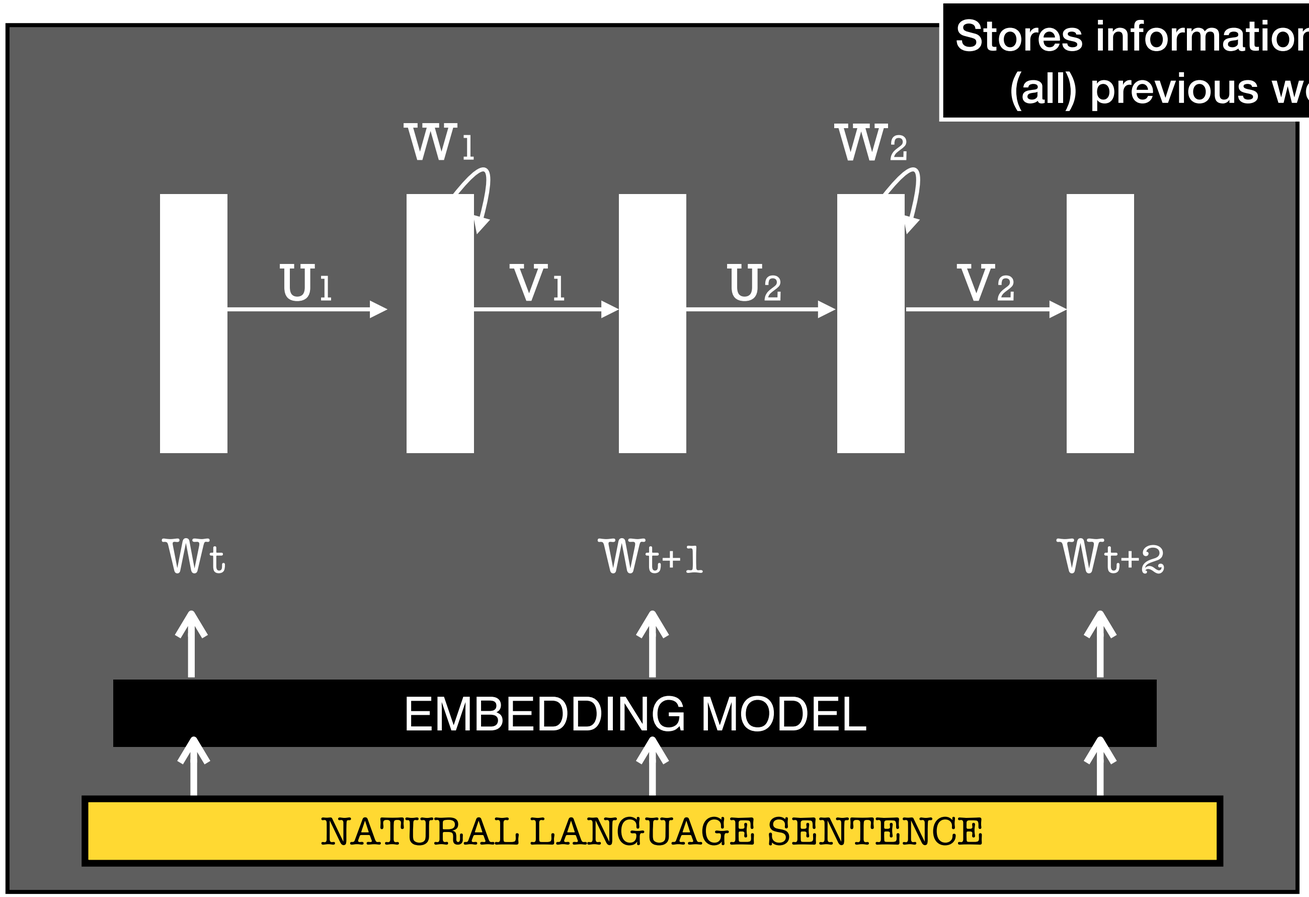
$$m^* = argmax(P(m|C)), \forall m \in M. \quad (1)$$

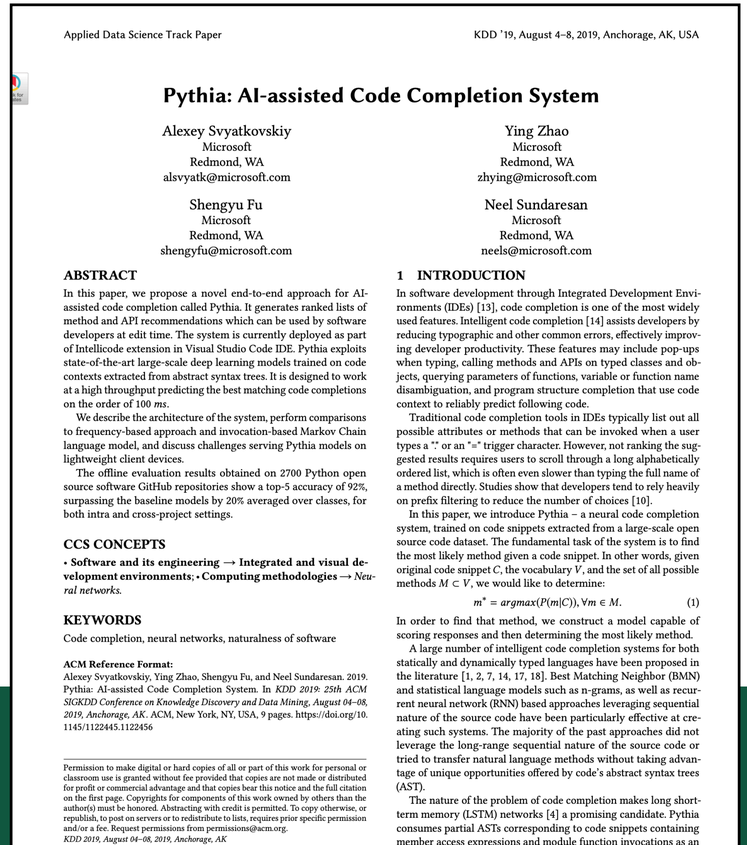
In order to find that method, we construct a model capable of scoring responses and then determining the most likely method.

A large number of intelligent code completion systems for both

Designing DL-based Models for Software Development Tasks

RNN-based Model





Applied Data Science Track Paper KDD '19, August 4–8, 2019, Anchorage, AK, USA

Pythia: AI-assisted Code Completion System

Alexey Svyatkovskiy
Microsoft
Redmond, WA
alsvyatk@microsoft.com

Ying Zhao
Microsoft
Redmond, WA
zhying@microsoft.com

Shengyu Fu
Microsoft
Redmond, WA
shengyfu@microsoft.com

Neel Sundaresan
Microsoft
Redmond, WA
neels@microsoft.com

ABSTRACT

In this paper, we propose a novel end-to-end approach for AI-assisted code completion called Pythia. It generates ranked lists of method and API recommendations which can be used by software developers at edit time. The system is currently deployed as part of Intellicode extension in Visual Studio Code IDE. Pythia exploits state-of-the-art large-scale deep learning models trained on code contexts extracted from abstract syntax trees. It is designed to work at a high throughput predicting the best matching code completions on the order of 100 *ms*.

We describe the architecture of the system, perform comparisons to frequency-based approach and invocation-based Markov Chain language model, and discuss challenges serving Pythia models on lightweight client devices.

The offline evaluation results obtained on 2700 Python open source software GitHub repositories show a top-5 accuracy of 92%, surpassing the baseline models by 20% averaged over classes, for both intra and cross-project settings.

CCS CONCEPTS

• **Software and its engineering** → **Integrated and visual development environments;** • **Computing methodologies** → *Neural networks.*

KEYWORDS

Code completion, neural networks, naturalness of software

ACM Reference Format:
Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, and Neel Sundaresan. 2019. Pythia: AI-assisted Code Completion System. In *KDD 2019: 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, August 04–08, 2019, Anchorage, AK.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

KDD 2019, August 04–08, 2019, Anchorage, AK

1 INTRODUCTION

In software development through Integrated Development Environments (IDEs) [13], code completion is one of the most widely used features. Intelligent code completion [14] assists developers by reducing typographic and other common errors, effectively improving developer productivity. These features may include pop-ups when typing, calling methods and APIs on typed classes and objects, querying parameters of functions, variable or function name disambiguation, and program structure completion that use code context to reliably predict following code.

Traditional code completion tools in IDEs typically list out all possible attributes or methods that can be invoked when a user types a "." or an "=" trigger character. However, not ranking the suggested results requires users to scroll through a long alphabetically ordered list, which is often even slower than typing the full name of a method directly. Studies show that developers tend to rely heavily on prefix filtering to reduce the number of choices [10].

In this paper, we introduce Pythia – a neural code completion system, trained on code snippets extracted from a large-scale open source code dataset. The fundamental task of the system is to find the most likely method given a code snippet. In other words, given original code snippet C, the vocabulary V, and the set of all possible methods $M \subset V$, we would like to determine:

$$m^* = argmax(P(m|C)), \forall m \in M. \quad (1)$$

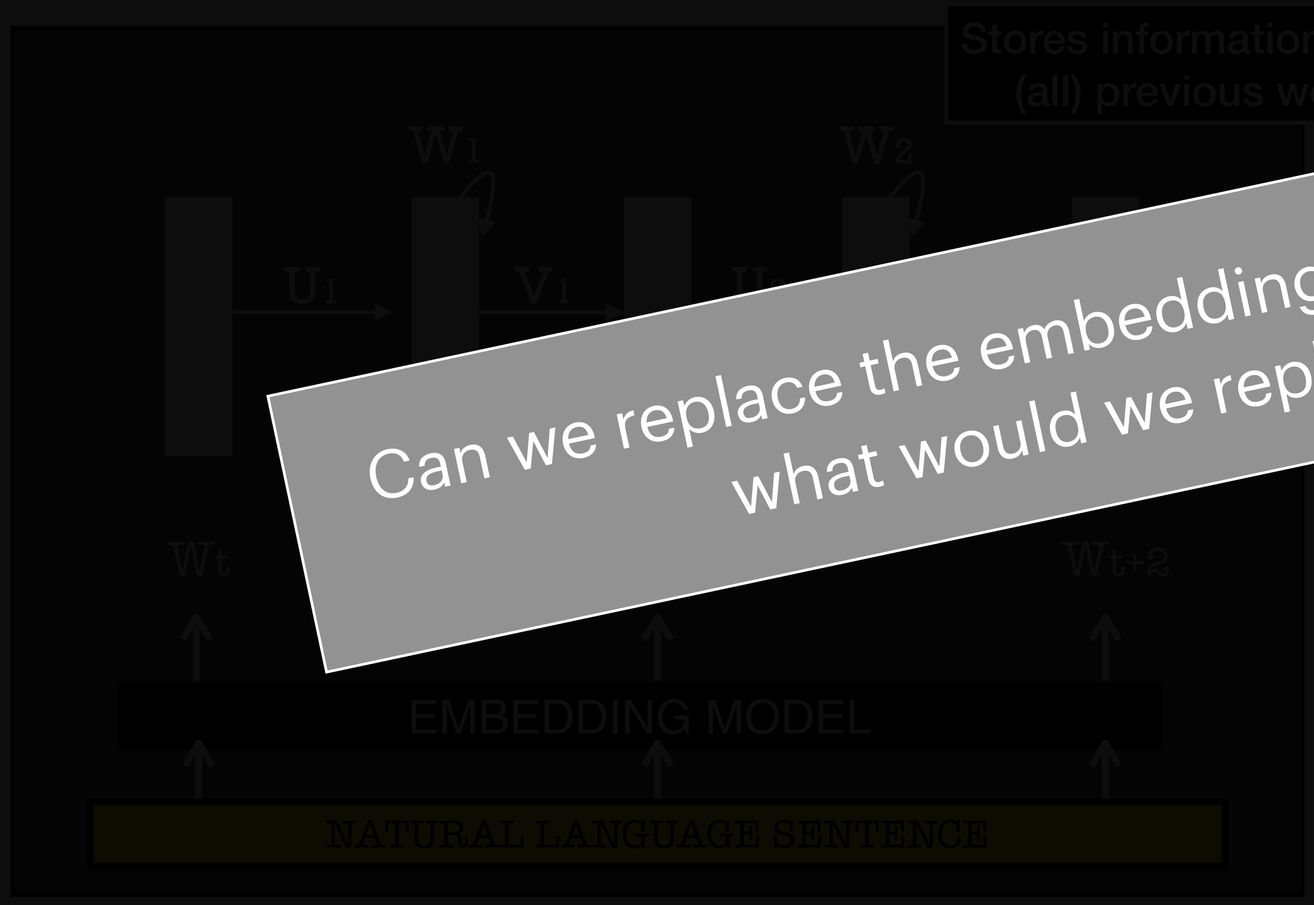
In order to find that method, we construct a model capable of scoring responses and then determining the most likely method.

A large number of intelligent code completion systems for both statically and dynamically typed languages have been proposed in the literature [1, 2, 7, 14, 17, 18]. Best Matching Neighbor (BMN) and statistical language models such as n-grams, as well as recurrent neural network (RNN) based approaches leveraging sequential nature of the source code have been particularly effective at creating such systems. The majority of the past approaches did not leverage the long-range sequential nature of the source code or tried to transfer natural language methods without taking advantage of unique opportunities offered by code's abstract syntax trees (AST).

The nature of the problem of code completion makes long short-term memory (LSTM) networks [4] a promising candidate. Pythia consumes partial ASTs corresponding to code snippets containing member access expressions and module function invocations as an

Designing DL-based Models for Software Development Tasks

RNN-based Model

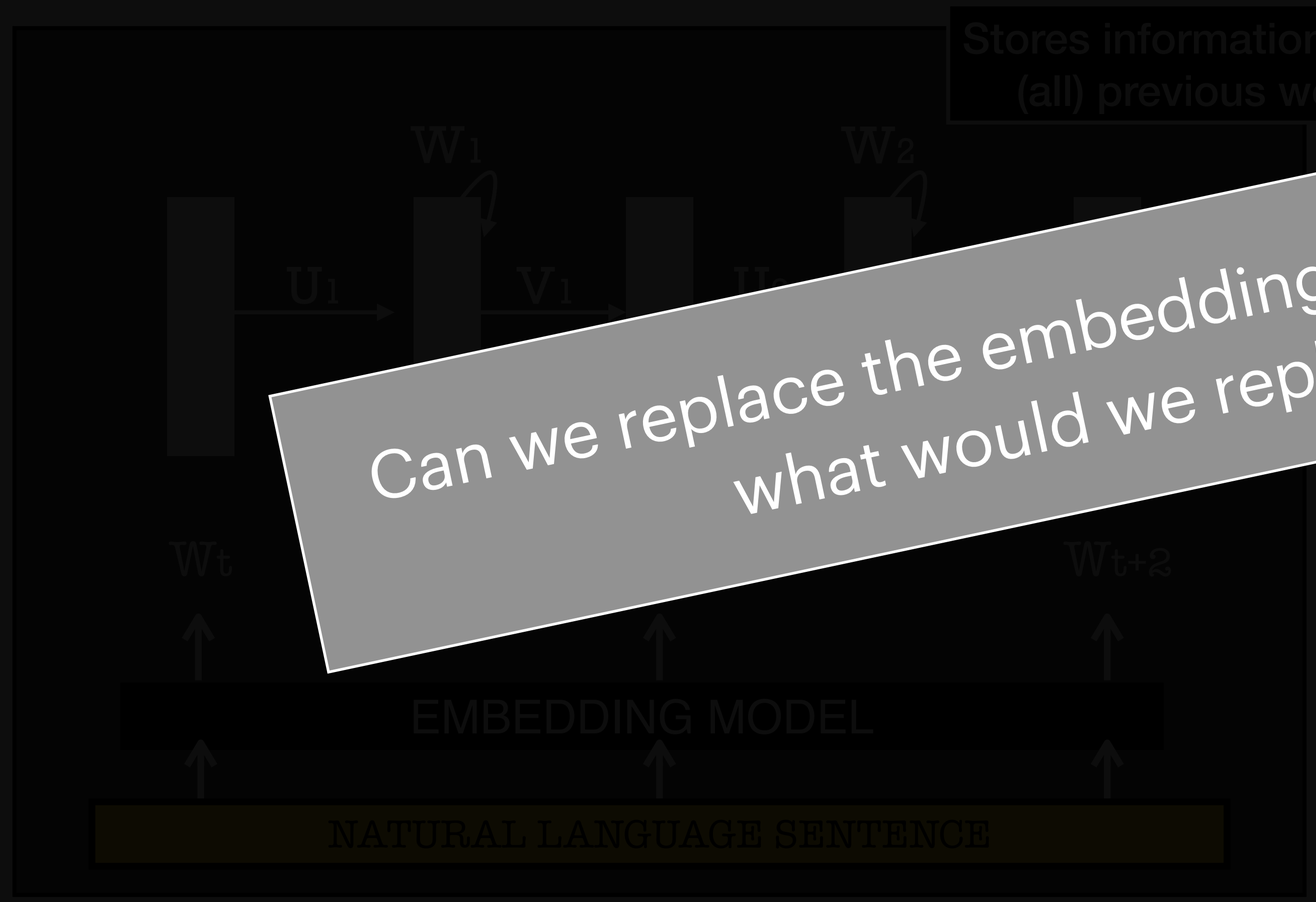


Can we replace the embedding model? And if we can, what would we replace it with?"

Designing DL-based Models for Software Development Tasks

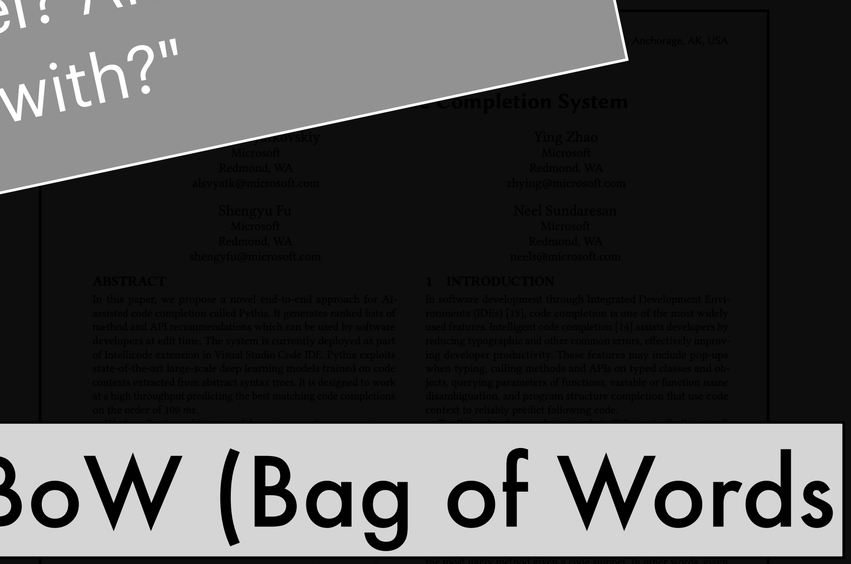
RN

One hot



Can we replace the embedding model? And if we can, what would we replace it with?"

BoW (Bag of Words


Anchorage, AK, USA

Completion System

Ying Zhao
Microsoft
Redmond, WA
zhying@microsoft.com

Neel Sundaresan
Microsoft
Redmond, WA
neels@microsoft.com

ABSTRACT

1. INTRODUCTION

KEYWORDS

ACM Reference Format:

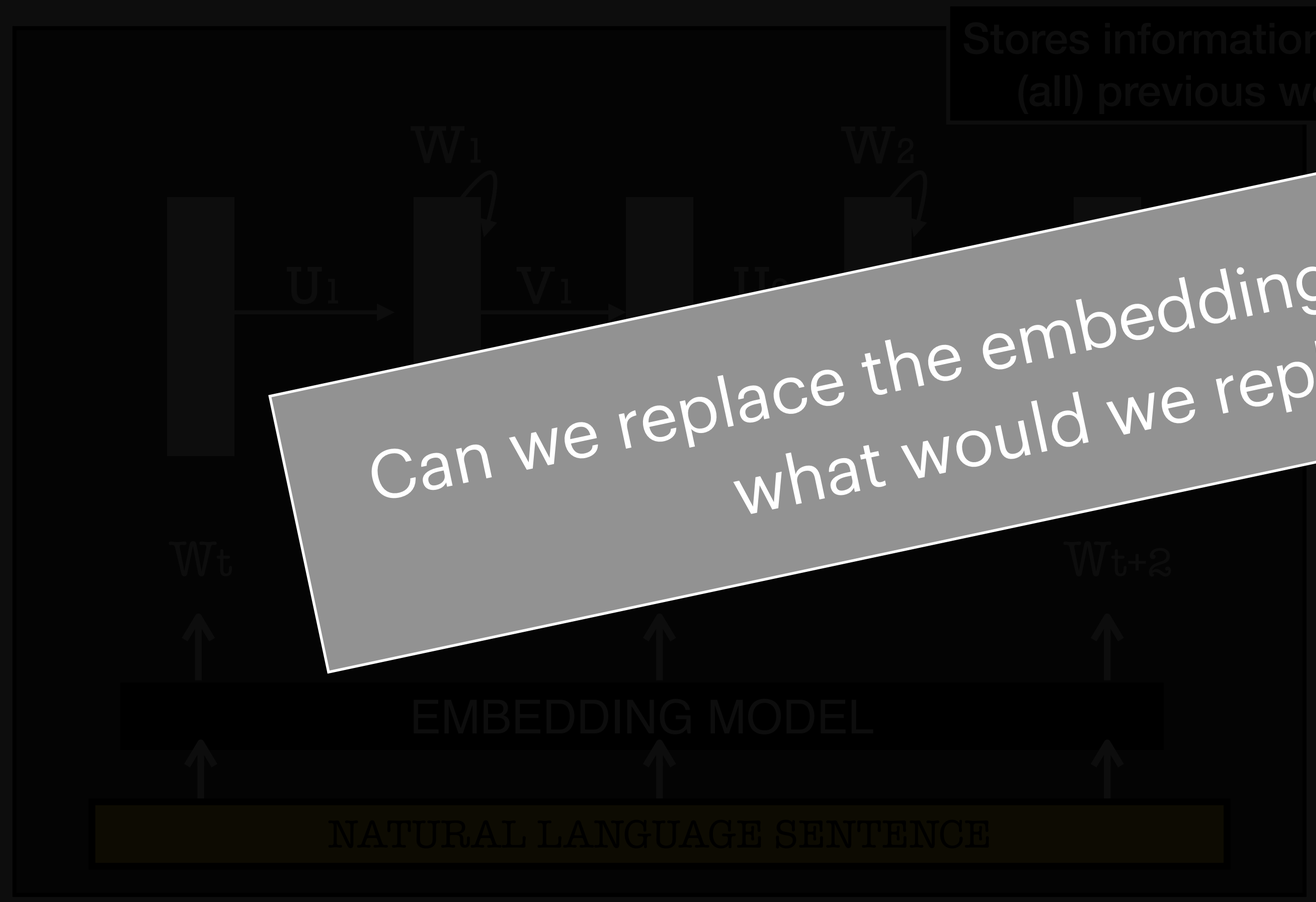
antoniomastropaolo.com

aura-se-lab.github.io

Designing DL-based Models for Software Development Tasks

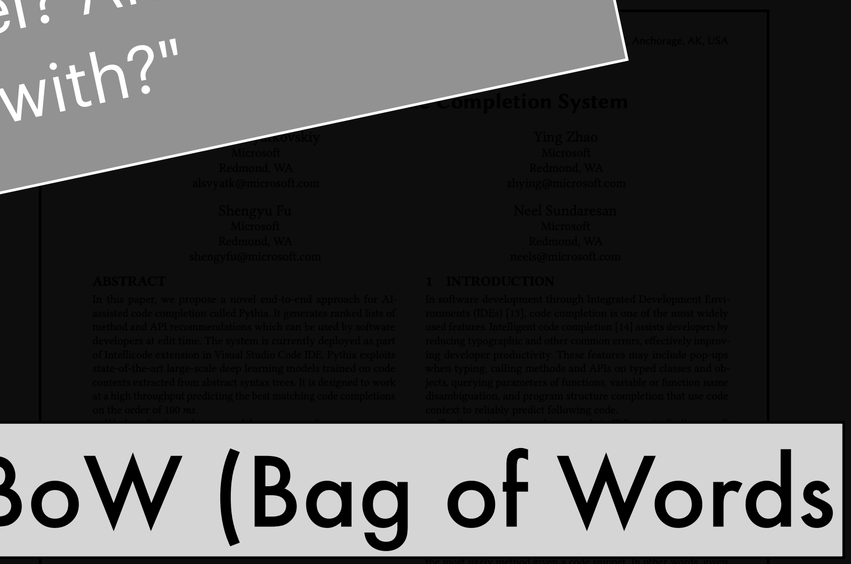
RN

One hot



Can we replace the embedding model? And if we can, what would we replace it with?"

BoW (Bag of Words


Anchorage, AK, USA

Completion System

Ying Zhao
Microsoft
Redmond, WA
zhying@microsoft.com

Neel Sundaresan
Microsoft
Redmond, WA
neels@microsoft.com

ABSTRACT

1. INTRODUCTION

KEYWORDS

ACM Reference Format:

antoniomastropaolo.com

aura-se-lab.github.io

Text Encoding Approaches for NLP Tasks

- I LIKE DOGS

- I LIKE CATS

Designing DL-based Models for Software Development Tasks

Text Encoding Approaches for NLP Tasks

- I LIKE DOGS

- I LIKE CATS

Designing DL-based Models for Software Development Tasks

Text Encoding Approaches for NLP Tasks

- I LIKE DOGS
- I LIKE CATS

One hot

1	0	0	0	0

0. <S>

0	1	0	0	0

1. I

0	0	0	1	0

3. LIKE

.	.	.	.	.

VOCAB

0. <S>

1. I

2. CATS

3. LIKE

4. DOGS

5. </S>

Designing DL-based Models for Software Development Tasks

Text Encoding Approaches for NLP Tasks

One hot

- I LIKE DOGS
- I LIKE CATS

1	0	0	0	0

0. <S>

0	1	0	0	0

1. I

0	0	0	1	0

3. LIKE

.	.	.	.	.

VOCAB
0. <S>
1. I
2. CATS
3. LIKE
4. DOGS
5. </S>

Designing DL-based Models for Software Development Tasks

Sentences/documents have different lengths

Encodings help turn them into fixed-size inputs

Text Encoding Approaches for NLP Tasks

- I LIKE DOGS

- I LIKE CATS

1	0	0	0	0

0. <S>

0	1	0	0	0

1. I

0	0	0	1	0

3. LIKE

.	.	.	.	.

VOCAB

0. <S>

1. I

2. CATS

3. LIKE

4. DOGS

5. </S>

Sentences/documents have different lengths

Encodings help turn them into fixed-size inputs

Text Encoding Approaches for NLP Tasks

- I LIKE DOGS
- I LIKE CATS

No Notion of Similarity

High dimensionality

Sparsity

Designing DL-based Models for Software Development Tasks

Text Encoding Approaches for NLP Tasks

- I LIKE DOGS
- I LIKE CATS

BoW
Bag of Words

VOCAB
0. <S>
1. I
2. CATS
3. LIKE
4. DOGS
5. </S>

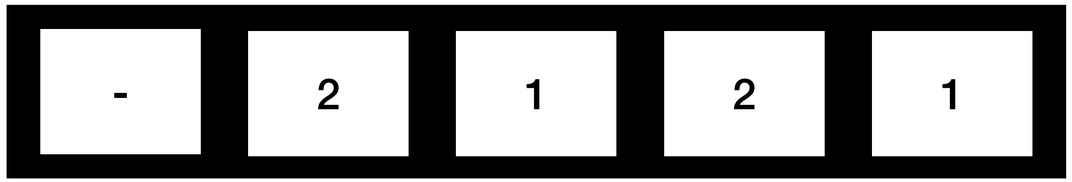
Designing DL-based Models for Software Development Tasks

Text Encoding Approaches for NLP Tasks

- I LIKE DOGS

- I LIKE CATS

BoW
Bag of Words



VOCAB
0. <S>
1. I
2. CATS
3. LIKE
4. DOGS
5. </S>

Designing DL-based Models for Soft nt Tasks

Simple and Fast for Traditional ML (See TF-IDF)

Text Encoding Appro

BoW
Bag of Words

- I LIKE DOGS
- I LIKE CATS

VOCAB

0. <S>

1. I

2. CATS

3. LIKE

4. DOGS

5. </S>

Designing DL-based Models for Sof nt Tasks

Text Encoding Appro

Simple and Fast for Traditional ML (See TF-IDF)

BoW
Bag of Words

- I LIKE DOGS
- I LIKE CATS

High dimensionality

Sparsity

VOCAB
0. <S>
1. I
2. CATS
3. LIKE
4. DOGS
5. </S>

Designing DL-based Models for Software Development Tasks

Text Encoding Approaches for NLP Tasks

BoW
Bag of Words

- I LIKE DOGS
- I LIKE CATS

VOCAB

0. <S>

1. I

2. CATS

3. LIKE

4. DOGS

5. </S>

Designing DL-based Models for Software Development Tasks

Text Encoding Approaches for NLP Tasks

BoW
Bag of Words

- I LIKE DOGS
- I LIKE CATS

-0.2	1.4	2.4

1. I

-0.6	1.1	0.1

3. LIKE

...	...	...

.....

...	...	...

.....

VOCAB
0. <S>
1. I
2. CATS
3. LIKE
4. DOGS
5. </S>

Dense vectors where similar words are close in space

Captures semantic similarity; low-dimensional

Transfers across tasks when pretrained

Text Encoding Approaches for NLP Tasks

BoW

Bag of Words

- I LIKE DOGS

-0.2	1.4	2.4

1. I

-0.6	1.1	0.1

3. LIKE

...	...	...

.....

...	...	...

.....

VOCAB

0. <S>

1. I

2. CATS

3. LIKE

4. DOGS

5. </S>

Dense vectors where similar words are close in space

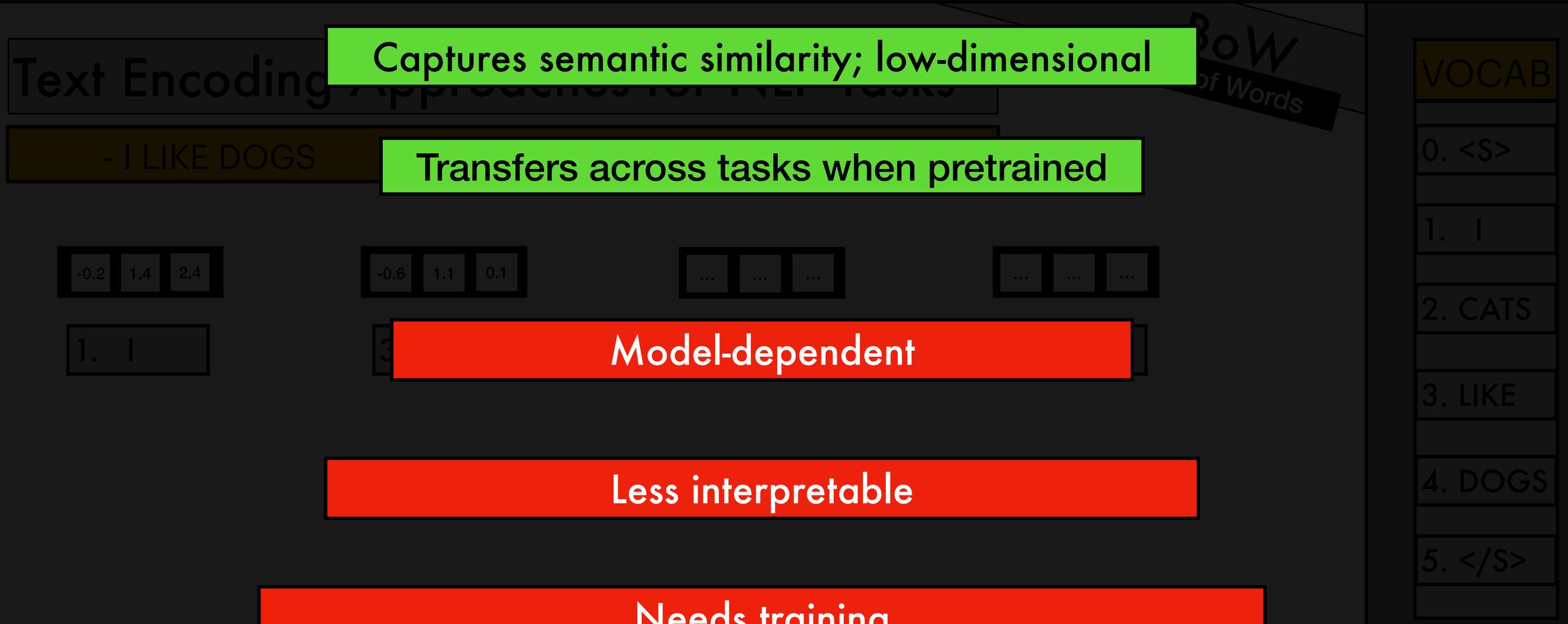
Captures semantic similarity; low-dimensional

Transfers across tasks when pretrained

Model-dependent

Less interpretable

Needs training



Sequence-to-Sequence Networks

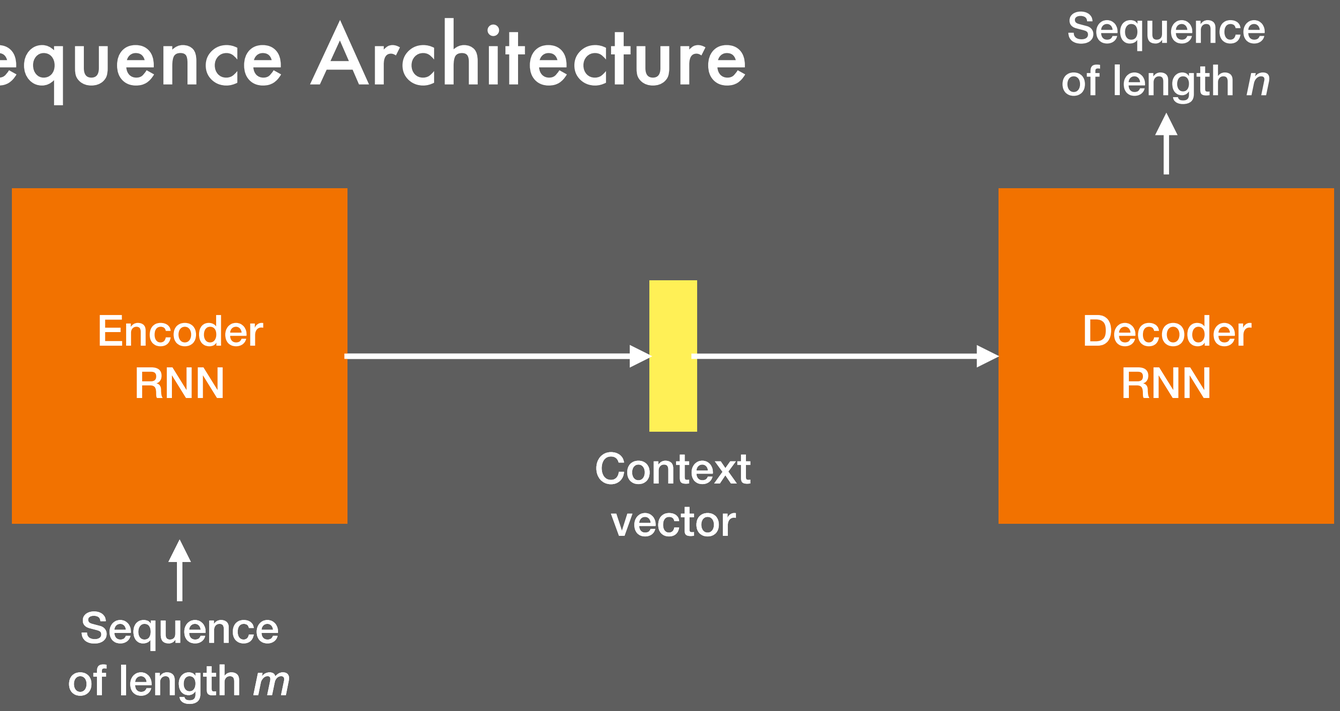
Goal: Translate sequence of items into another sequence of items

Examples of application:

- *Translation between natural languages*
- *Generate image captions*
- *Answer natural language questions*

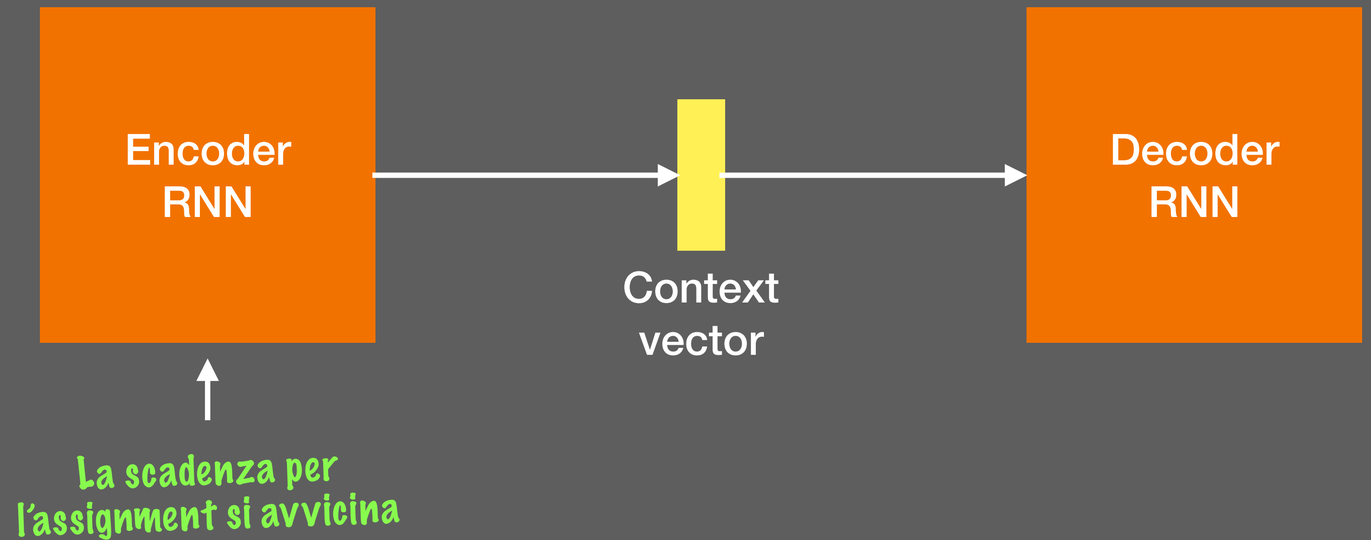
Designing DL-based Models for Software Development Tasks

Sequence-to-Sequence Architecture



Designing DL-based Models for Software Development Tasks

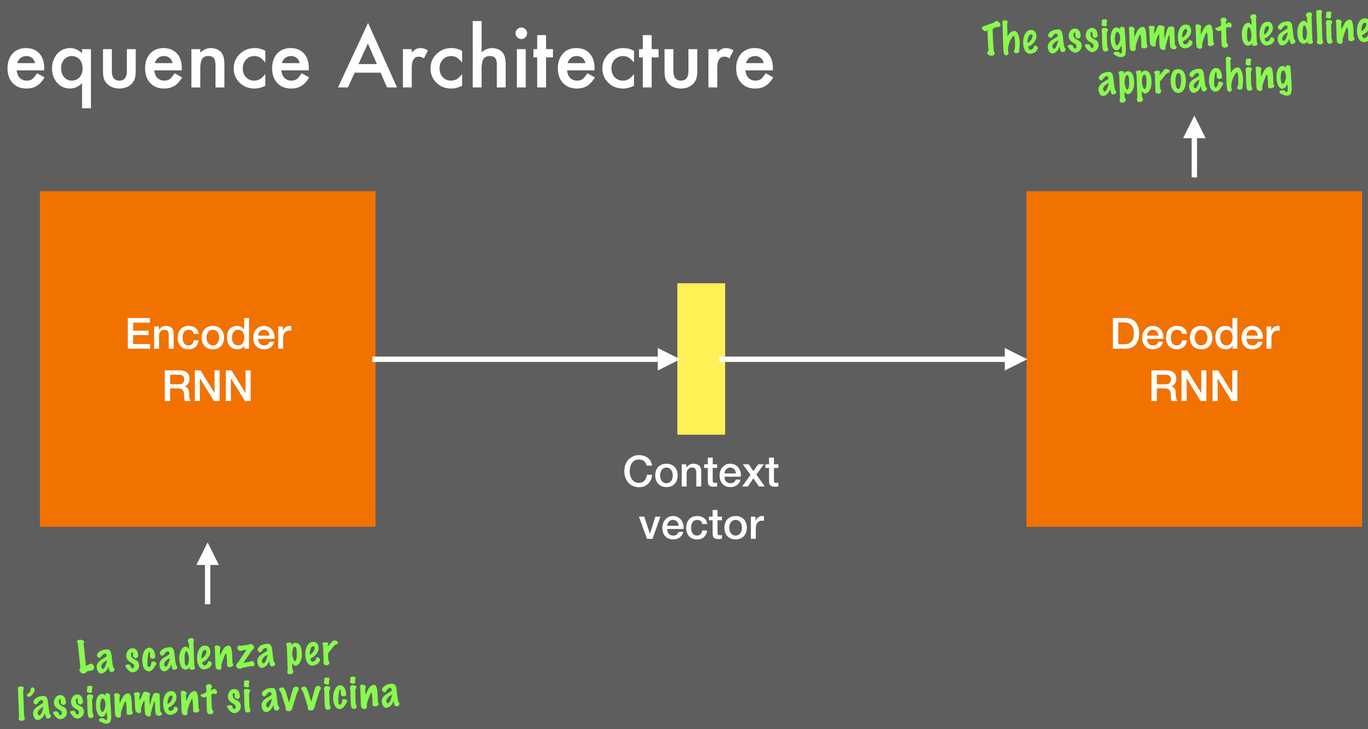
Sequence-to-Sequence Architecture



*m may be different from n

Designing DL-based Models for Software Development Tasks

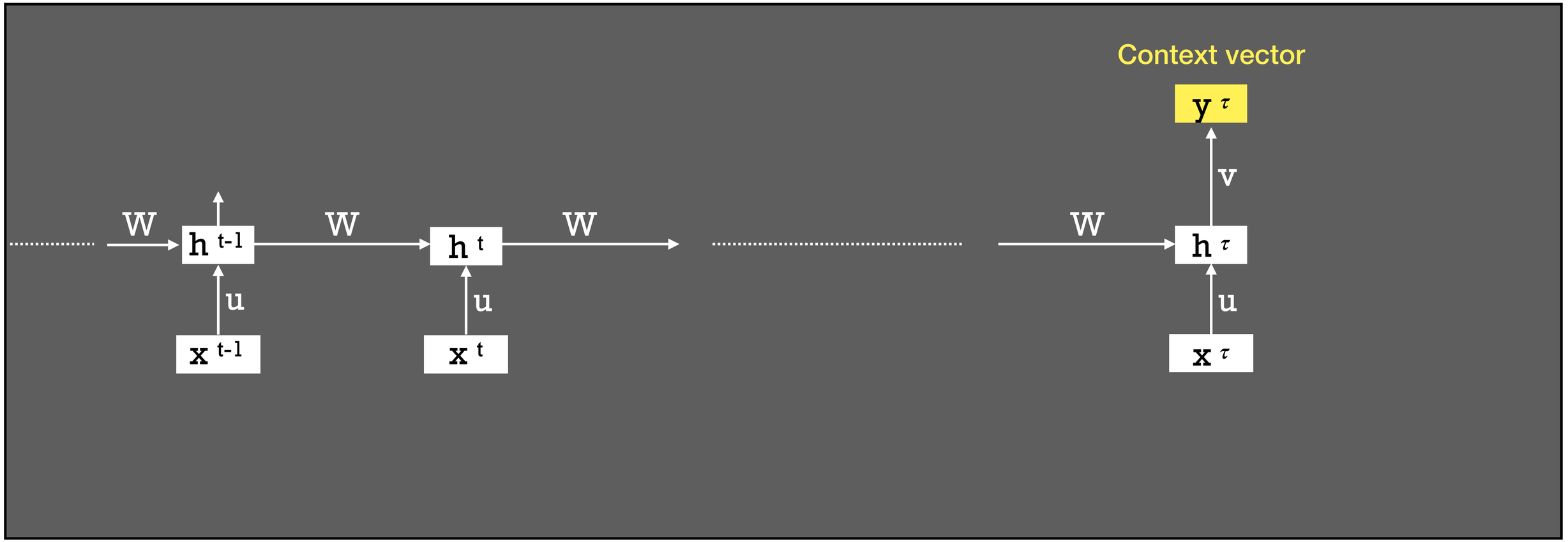
Sequence-to-Sequence Architecture



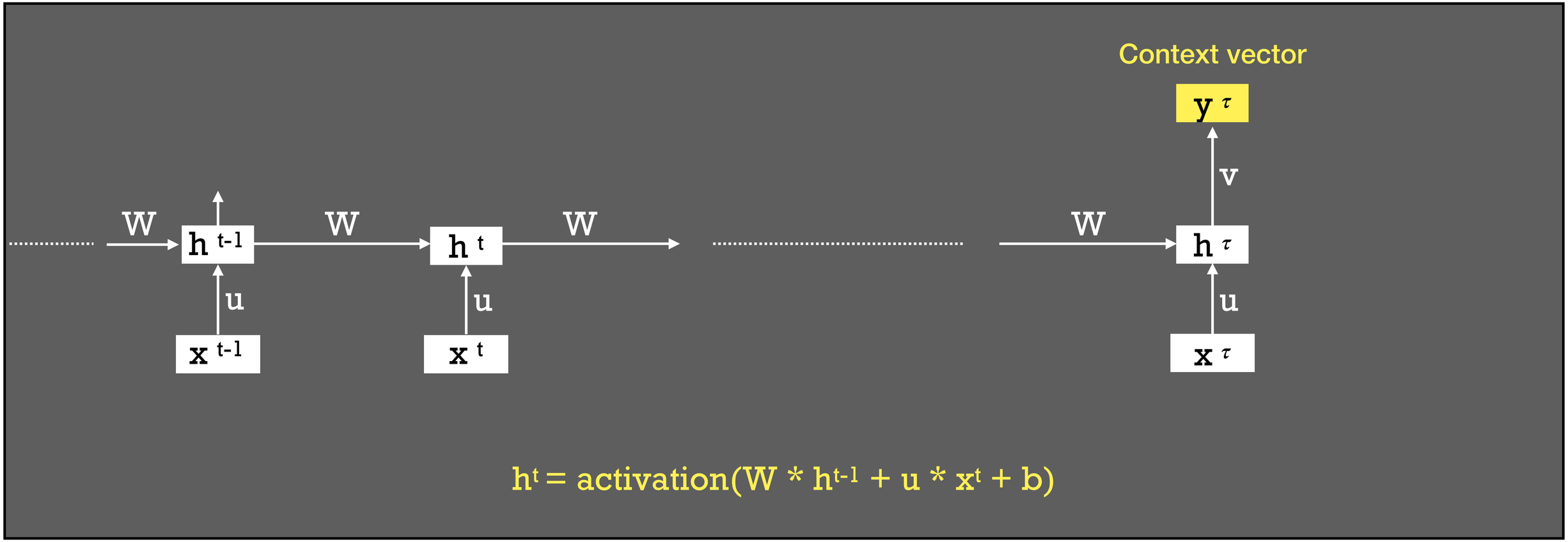
*m may be different from n

Both networks are trained jointly: The context vector summarizes the input in such a way that it maximizes correct output generation (e.g., useless details from the input may be ignored)

Encoder RNN



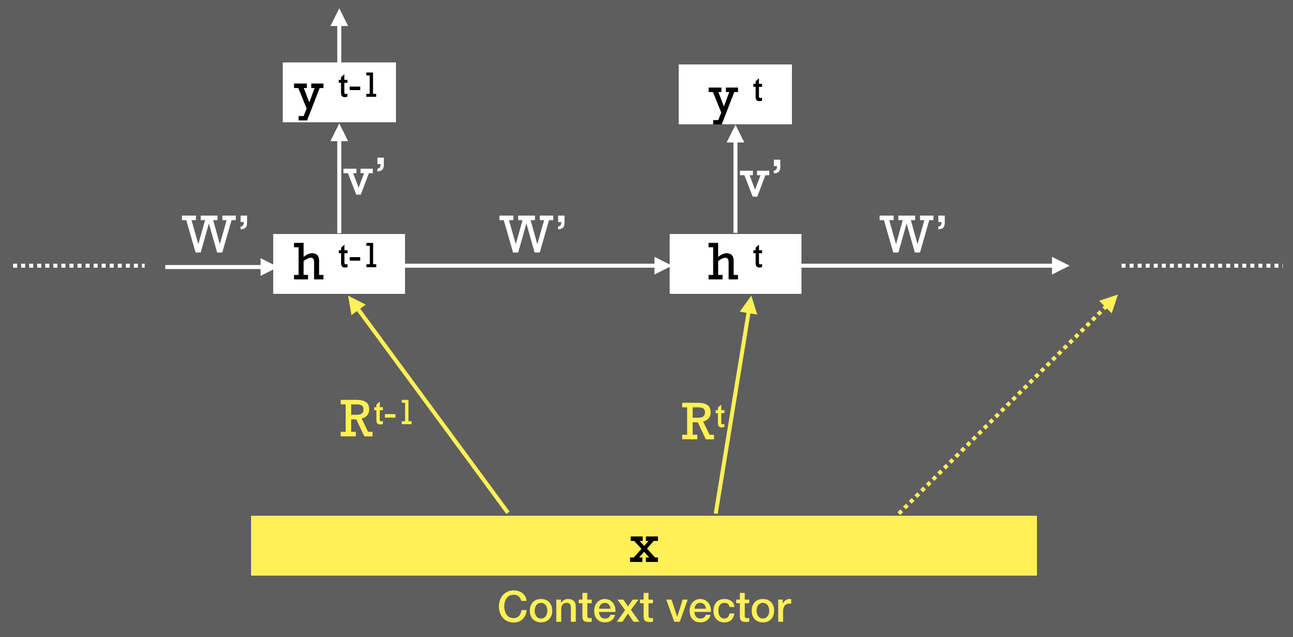
Encoder RNN



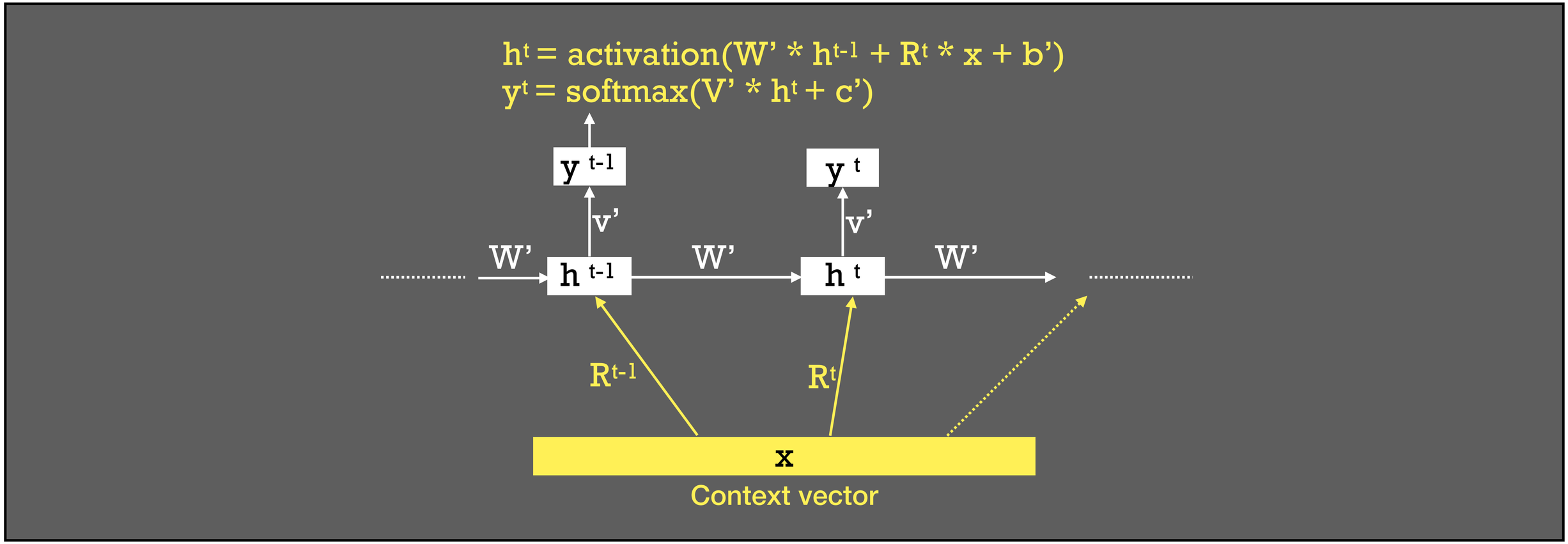
$h^t = \text{activation}(W * h^{t-1} + u * x^t + b)$

Decoder RNN

$$h^t = \text{activation}(W' * h^{t-1} + R^t * x + b')$$



Decoder RNN


$h^t = activation(W' * h^{t-1} + R^t * x + b')$
$y^t = softmax(V' * h^t + c')$
y t-1
y t
v'
v'
W'
W'
W'
h t-1
h t
R^{t-1}
R^t
x
Context vector

Seq2Seq Training

Training data: *N pairs of sequences* (x_i, y_i)

Function to minimize: *Translation mistakes from the training set.*

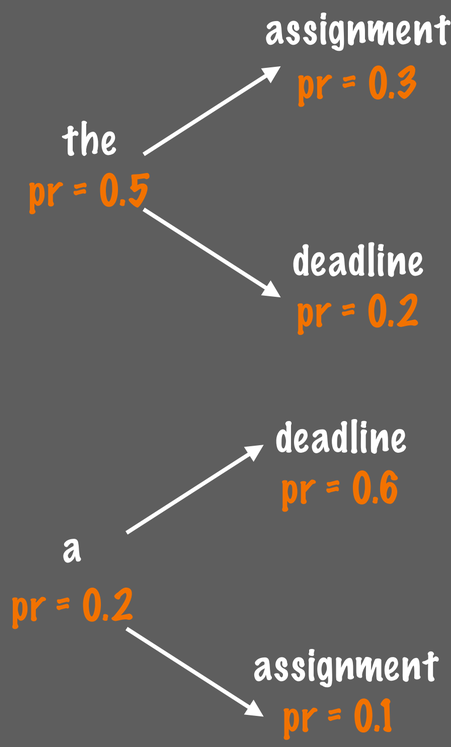
Beam search: *For many applications, we want the k most likely outputs (e.g., translations).*

With beam size = k, at every step we keep the k most likely output tokens.

Designing DL-based Models for Software Development Tasks

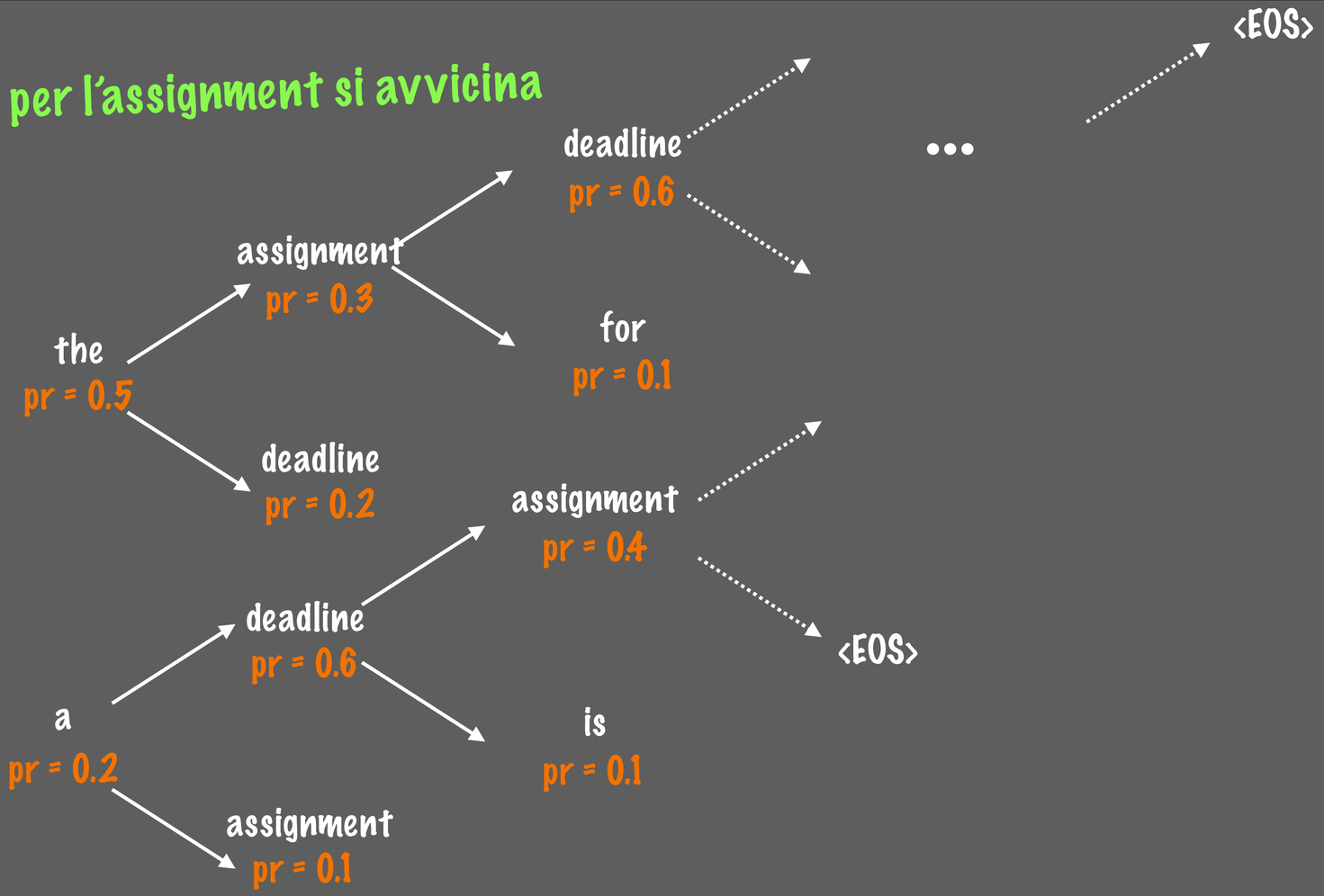
La scadenza per l'assignment si avvicina

K=2



Designing DL-based Models for Software Development Tasks

La scadenza per l'assignment si avvicina



K=2

Using Seq2Seq for Fixing Bugs in Java Systems

https://serokell.io/blog/deep-learning-and-neural-network-guide

Motivation

Fixing bugs in Java systems might be a demanding and time-consuming tasks

On which line the bug is being exercised?

Is a multi-line o single-line bug?

Goal:

Automatically generate fix bug-fixes for buggy Java programs.

Formulate the problem as a translation problem:

Input: Each input represent a sequence of Java tokens resembling the original buggy method

Output: Each input represent a sequence of Java tokens resembling the original fixed method

https://serokell.io/blog/deep-learning-and-neural-network-guide

Designing DL-based Models for Software Development Tasks

An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation

MICHELE TUFANO, College of William and Mary, USA
CODY WATSON, College of William and Mary, USA
GABRIELE BAVOTA, Università della Svizzera italiana (USI), Switzerland
MASSIMILIANO DI PENTA, University of Sannio, Italy
MARTIN WHITE, College of William and Mary, USA
DENYS POSHYVANYK, College of William and Mary, USA

Millions of open-source projects with numerous bug fixes are available in code repositories. This proliferation of software development histories can be leveraged to learn how to fix common programming bugs. To explore such a potential, we perform an empirical study to assess the feasibility of using Neural Machine Translation techniques for learning bug-fixing patches for real defects. First, we mine millions of bug-fixes from the change histories of projects hosted on GitHub, in order to extract meaningful examples of such bug-fixes. Next, we abstract the buggy and corresponding fixed code, and use them to train an Encoder-Decoder model able to translate buggy code into its fixed version. In our empirical investigation we found that such a model is able to fix thousands of unique buggy methods in the wild. Overall, this model is capable of predicting fixed patches generated by developers in 9-50% of the cases, depending on the number of candidate patches we allow it to generate. Also, the model is able to emulate a variety of different Abstract Syntax Tree operations and generate candidate patches in a split second.

CCS Concepts: • **Software and its engineering** → *Maintaining software.*

Additional Key Words and Phrases: neural machine translation, bug-fixes

ACM Reference Format:
Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2019. An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. *ACM Trans. Softw. Eng. Methodol.* -, -, Article - (February 2019), 28 pages. https://doi.org/-.-

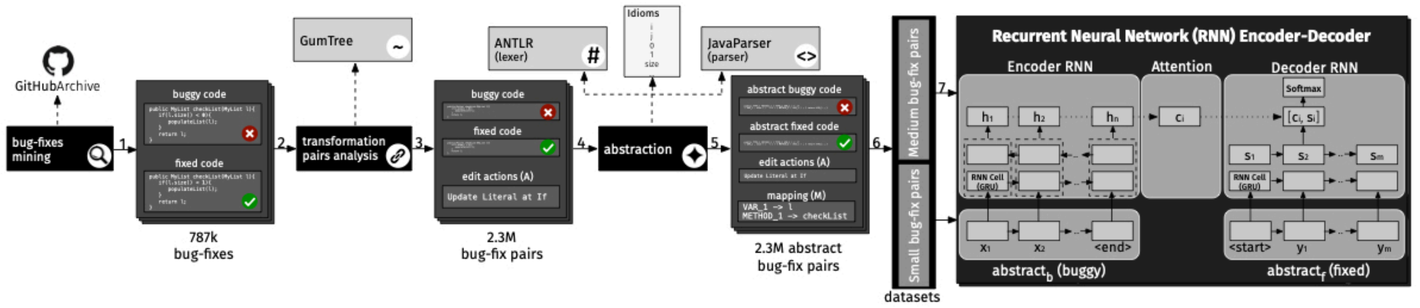
1 INTRODUCTION

Localizing and fixing bugs is known to be an effort-prone and time-consuming task for software developers [33, 72, 89]. To support programmers in this common activity, researchers have proposed a number of approaches aimed at automatically repairing programs [5, 8, 20, 21, 36, 37, 41, 42, 45, 48, 52, 53, 61, 64, 66, 76, 78, 86, 88, 93]. The proposed techniques either use a generate-and-validate approach, which consists of generating many repairs (*e.g.,* through Genetic Programming like GenProg [45, 87]), or an approach that produces a single fix [32, 61]. While automated program

Authors' addresses: Michele Tufano, College of William and Mary, Williamsburg, Virginia, USA, mtufano@cs.wm.edu; Cody Watson, College of William and Mary, Williamsburg, Virginia, USA, cawatson@cs.wm.edu; Gabriele Bavota, Università della Svizzera italiana (USI), Lugano, Switzerland, gabriele.bavota@usi.ch; Massimiliano Di Penta, University of Sannio, Benevento, Italy, dipenta@unisannio.it; Martin White, College of William and Mary, Williamsburg, Virginia, USA, mgwhite@cs.wm.edu; Denys Poshyvanyk, College of William and Mary, Williamsburg, Virginia, USA, denys@cs.wm.edu.

1049-331X/2019/2-ART- $15.00
https://doi.org/-.-

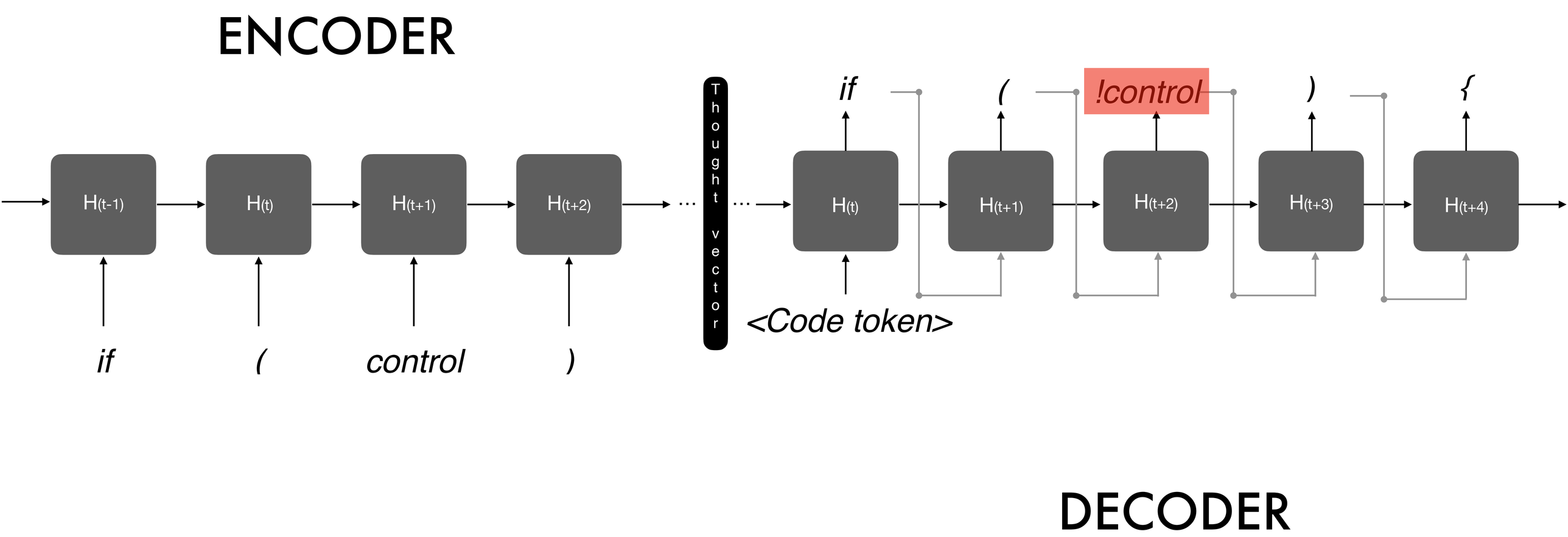
Neural-Machine-Translation



Designing DL-based Models for Software Development Tasks

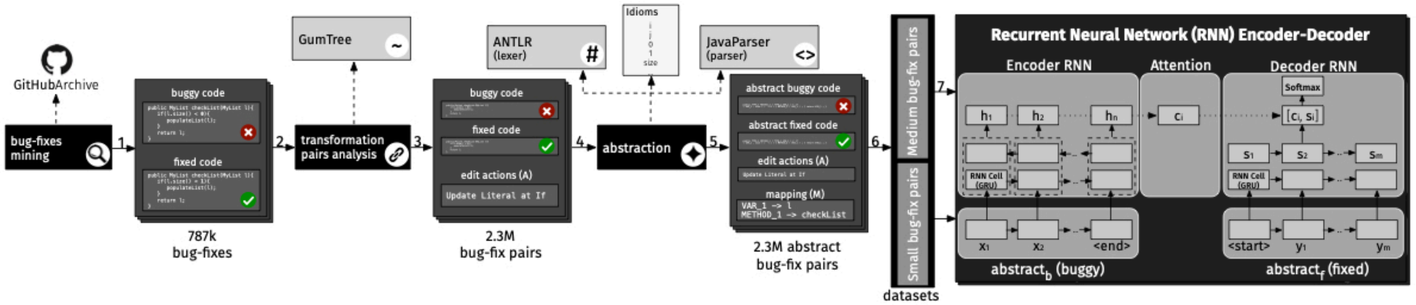
Input		Target	
1	Buggy method	2	Fixed method
1	Buggy method	2	Fixed method
1	Buggy method	2	Fixed method
1	Buggy method	2	Fixed method
...	~60K	...	~60K

Deep Learning for SE Automation



Designing DL-based Models for Software Development Tasks

Neural-Machine-Translation



EXACT MATCHES

BLEU SCORE

CODE-BLEU

CRYSTAL-BLEU

Designing DL-based Models for Software Development Tasks

BLEU SCORE

BLEU SCORE

- R1: The cat is on the mat.
- R2: There is a cat on the mat.
- C1: The cat and the dog.

Designing DL-based Models for Software Development Tasks

BLEU SCORE

- R1: The cat is on the mat.
- R2: There is a cat on the mat.
- C1: The cat and the dog.

- There are 3 unigrams appearing in the reference translations: “the”, “cat”, “the”
- C1 has length 5, hence the precision (let’s call it BLEU*) is
- BLEU*=3/5=0.6

BLEU SCORE

- What if C1 were:
  - C1 = The The The
- Then, BLEU would be:
  - BLEU*=3/3=1
  - This doesn’t make sense— at all!

Designing DL-based Models for Software Development Tasks

BLEU SCORE

- What if C1 were:
  - C1 = The The The
- Then, BLEU would be:
  - BLEU*=3/3=1
  - This doesn't make sense— at all!

- We should use as numerator the **maximum** number of times the **word(s)** appear in the reference translations
  - R1: The cat is on the mat.
  - R2: There is a cat on the mat.
  - C1: The cat and the dog.
- In this case, the word "the" appears at most twice. Hence
- BLEU**=2/3=0.67

Designing DL-based Models for Software Development Tasks

BLEU SCORE

- What if C1 were:
  - C1 = The The The
- Then, BLEU would be:
  - BLEU*=3/3=1
  - This doesn't make sense— at all!

- We should use as numerator the **maximum** number of times the **word(s)** appear in the reference translations
  - R1: The cat is on the mat.
  - R2: There is a cat on the mat.
  - C1: The cat and the dog.
- In this case, the word "the" appears at most twice. Hence
- BLEU**=2/3=0.67

FIRST FIX

BLEU SCORE

- Let's consider two references and two candidate generations:
  - R1: The cat is on the mat.
  - R2: There is a cat on the mat.
  - C3: Mat the cat is on a there.
- Let us now compute the bigrams too defining $BLEU_1$** and $BLEU_2$** for C4
  - $BLEU_1$**(C3) = 7/7 = 1.0
  - $BLEU_2$**(C3) = 0/6 = 0
- The bigrams for C3 are "Mat the", "the cat", "cat is", "is on", "on a", "a there"
- ***They never appear in the references***

Designing DL-based Models for Software Development Tasks

BLEU SCORE

We can **penalize** short sequences by adding a discount factor;
Brevity Penalty

$$BLEU = BP \cdot \overline{BLEU}$$

- Let's consider:
  - R1: The cat is on the mat.
  - R2: There is a cat on the mat.
  - C5: There is a cat.
- The $BLEU_1$** and $BLEU_2$** for C5 are computed as follows:
  - $BLEU_1$**(C5) = 4/4 = 1.0
  - $BLEU_2$**(C5) = 3/3= 1.0

BI-GRAM

- It seems both indicate a **perfect prediction**, yet the candidates are different from the reference!

Designing DL-based Models for Software Development Tasks

CodeBLEU

CodeBLEU: a Method for Automatic Evaluation of Code Synthesis

Shuo Ren[1], Daya Guo[2], Shuai Lu[3], Long Zhou[4], Shujie Liu[4], Duyu Tang[4], Neel Sundaresan[4], Ming Zhou[4], Ambrosio Blanco[4], Shuai Ma[1]

[1]SKLSDE Lab, Beihang University; Beijing Advanced Innovation Center for Big Data and Brain Computing
[2]Sun Yat-sen University [3]Peking University [4]Microsoft
[1]{shuoren, mashuai}@buaa.edu.cn [2]guody5@mail2.sysu.edu.cn [3]lushuai96@pku.edu.cn
[4]{Long.Zhou, shujliu, dutang, neels, mingzhou, ambrob}@microsoft.com

Abstract

Evaluation metrics play a vital role in the growth of an area as it defines the standard of distinguishing between good and bad models. In the area of code synthesis, the commonly used evaluation metric is BLEU or perfect accuracy, but they are not suitable enough to evaluate codes, because BLEU is originally designed to evaluate natural language, neglecting important syntactic and semantic features of codes, and perfect accuracy is too strict thus it underestimates different outputs with the same semantic logic. To remedy this, we introduce a new automatic evaluation metric, dubbed CodeBLEU. It absorbs the strength of BLEU in the n-gram match, and further injects code syntax via abstract syntax trees (AST) and code semantics via data-flow. We conduct experiments by evaluating the correlation coefficient between CodeBLEU and quality scores assigned by the programmers on three code synthesis tasks, i.e., text-to-code, code translation, and code refinement. Experimental results show that, our proposed CodeBLEU can achieve a better correlation with programmer assigned scores compared with BLEU and accuracy.

1 Introduction

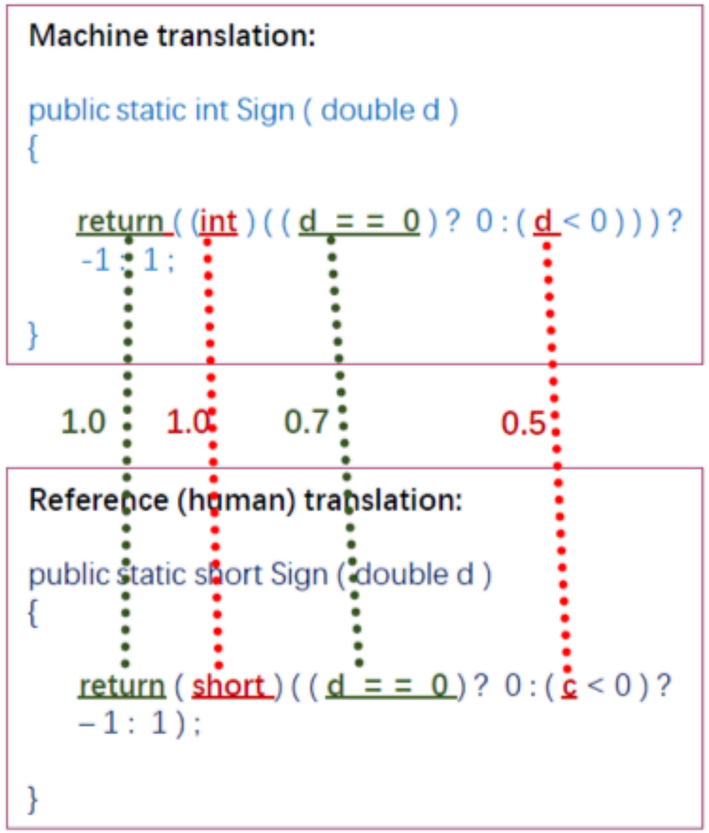
A suitable evaluation metric is important to push forward

recently proposed computational accuracy (Lachaux et al. 2020), evaluates whether the hypothesis function generates the same outputs as the reference given the same inputs.

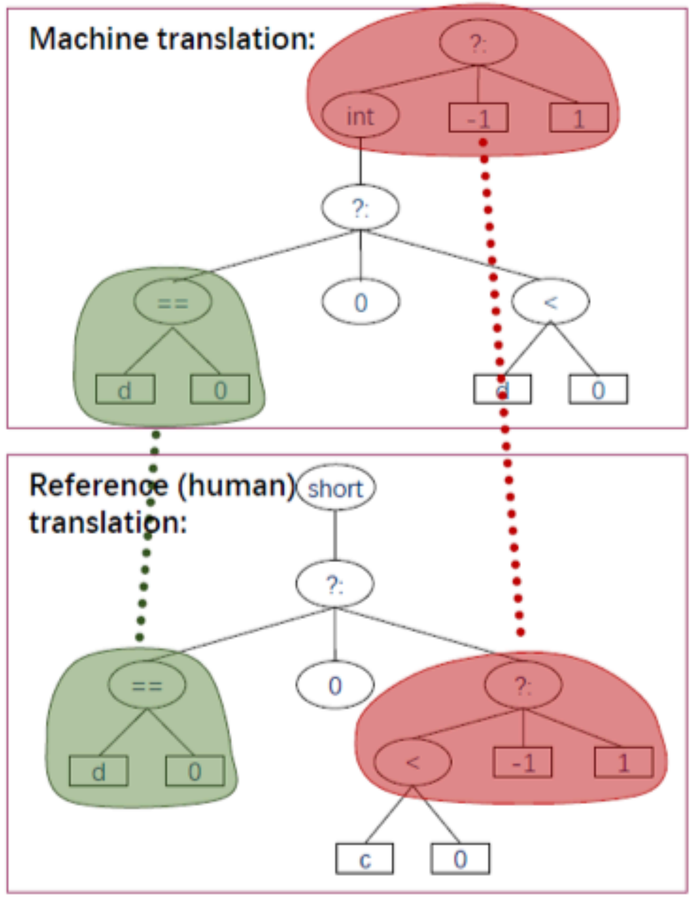
However, the above evaluation approaches still face many drawbacks. First, the n-gram accuracy does not take into account the grammatical and logical correctness, resulting in favoring candidates with high n-gram accuracy and serious logical errors. Second, the perfect accuracy is too strict, and underestimates different outputs with the same semantic logic. Third, the computational accuracy is weak in universality and practicability, since it should be designed for different programming languages, as well as specific compilers and the desired computing resource.

In order to deal with that, in this paper, we propose a new evaluation metric CodeBLEU, considering information from not only the shallow (n-gram) match, but also the syntactic match and the semantic match. More specifically, the n-gram match assigns different weights for different n-grams, the syntactic match considers the abstract syntax tree (AST) information in the evaluation score by matching the sub-trees, and the semantic match uses data-flow structure to measure the semantic similarity. CodeBLEU is a weighted combination of the original BLEU, the weighted n-gram match, the

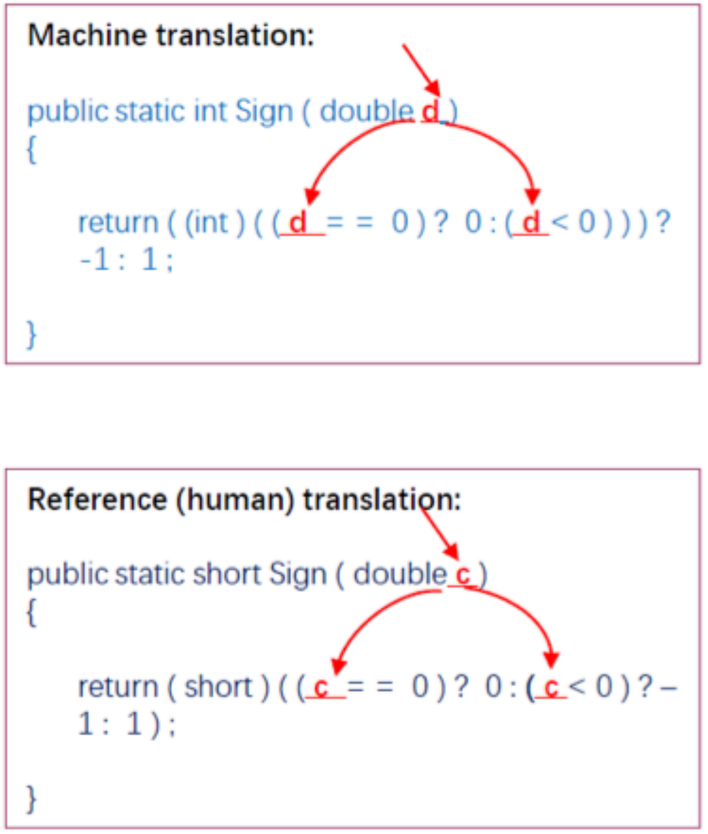
Designing DL-based Models for Software Development Tasks



Weighted N-Gram Match



Syntactic AST Match



Semantic Data-flow Match

Hierarchical Neural Networks

Hierarchical Neural Network

Useful when we want to reason about entities which are represented by multiple parts that:

Are too many to simply concatenate (e.g., in Feed Forward Network)

Are not a sequence, thus making them not suitable for an RNN

May each have a different structure

Hierarchical Neural Network

Useful when we want to reason about entities which are represented by multiple parts that:

Are too many to simply concatenate (e.g., in Feed Forward Network)

Are not a sequence, thus making them not suitable for an RNN

May each have a different structure

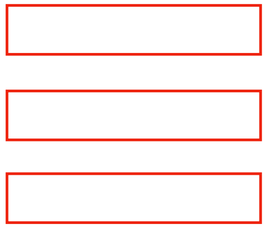
Program crash: *stack trace, error message, information about the user*

Code function: *code tokens, function name, comments describing the function*

Commits in a repository: *diff (possibly spanning multiple locations), commit message*

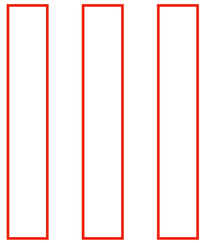
Designing DL-based Models for Software Development Tasks

Text



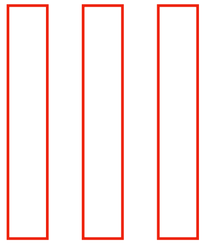
Vectors of words

Images



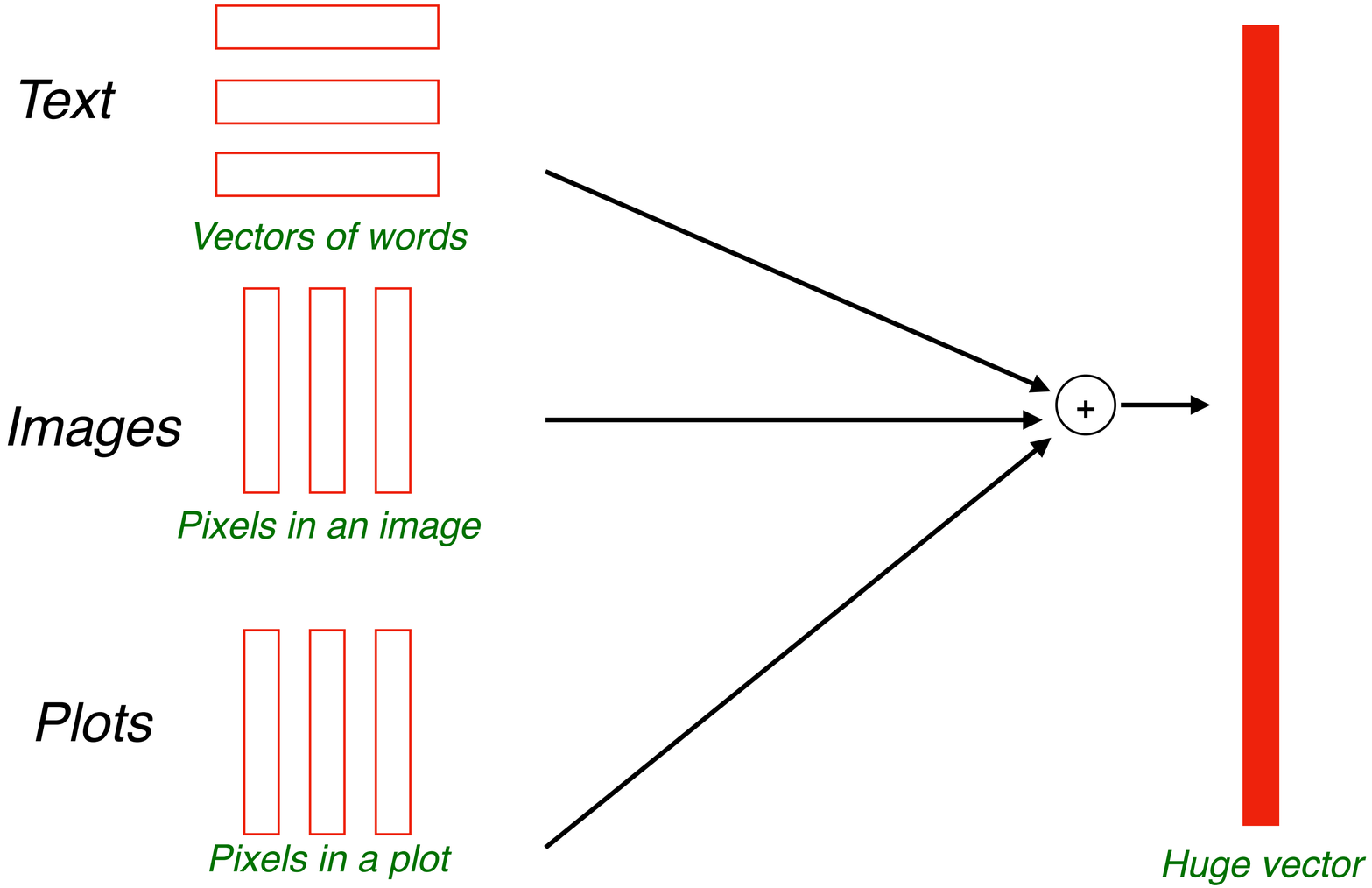
Pixels in an image

Plots

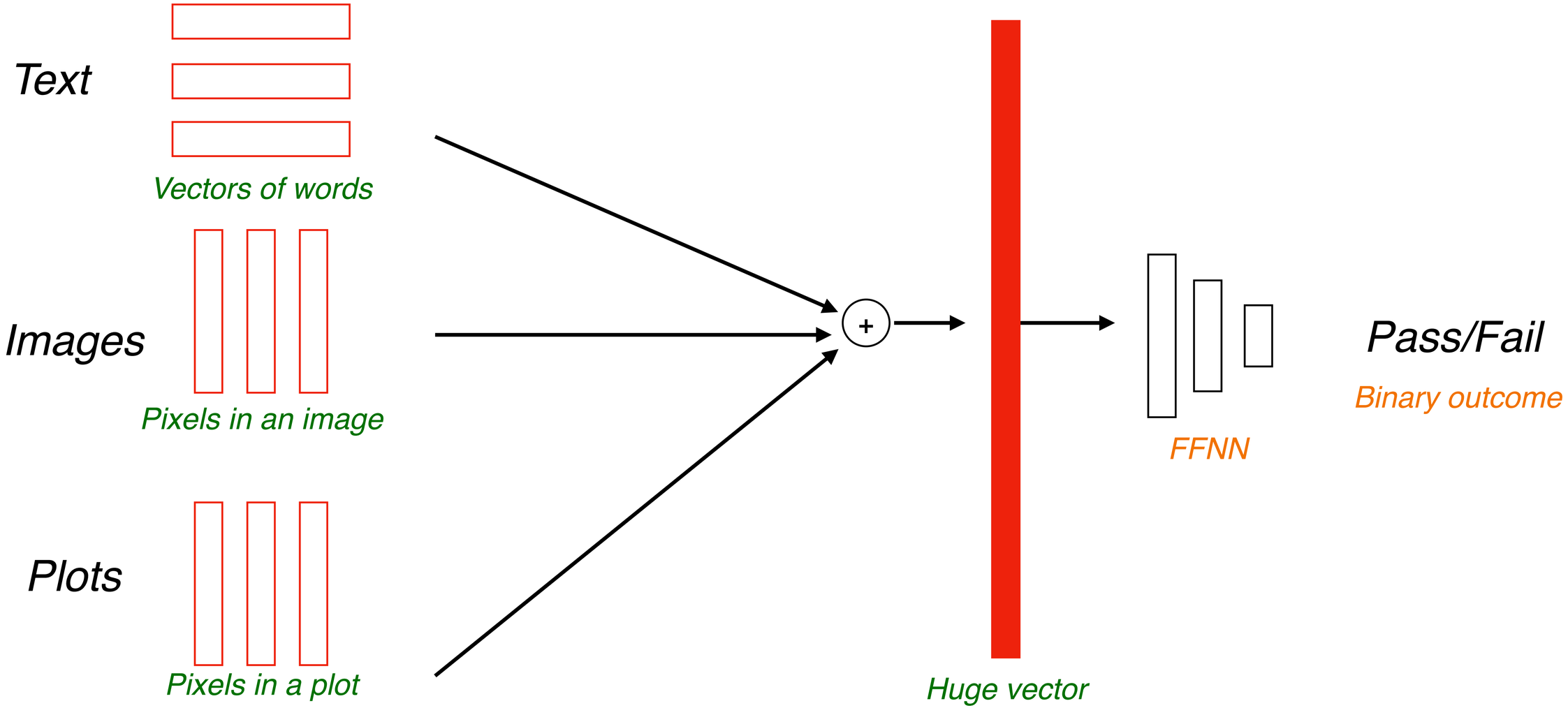


Pixels in a plot

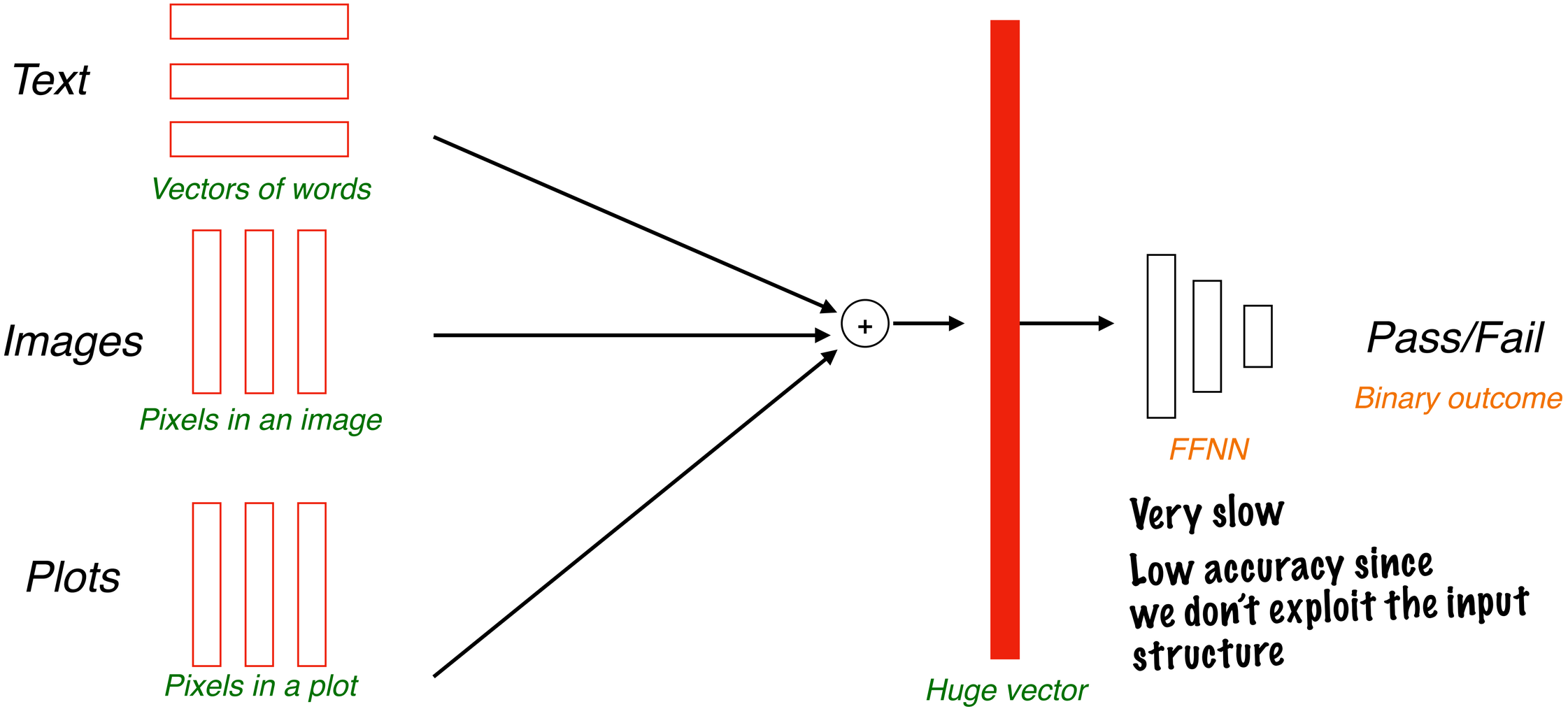
Designing DL-based Models for Software Development Tasks



Deep Learning for SE Automation



Designing DL-based Models for Software Development Tasks



Hierarchical Neural Network

Neural model composed of submodels aligned into a hierarchy

Think of it as a tree where the inputs arrive at leaves

We will reason about different parts of the input individually, and then start aggregating this information towards the root of the tree

Prediction based on summarized information at the root

Hierarchical Neural Network

Neural model composed of submodels aligned into a hierarchy

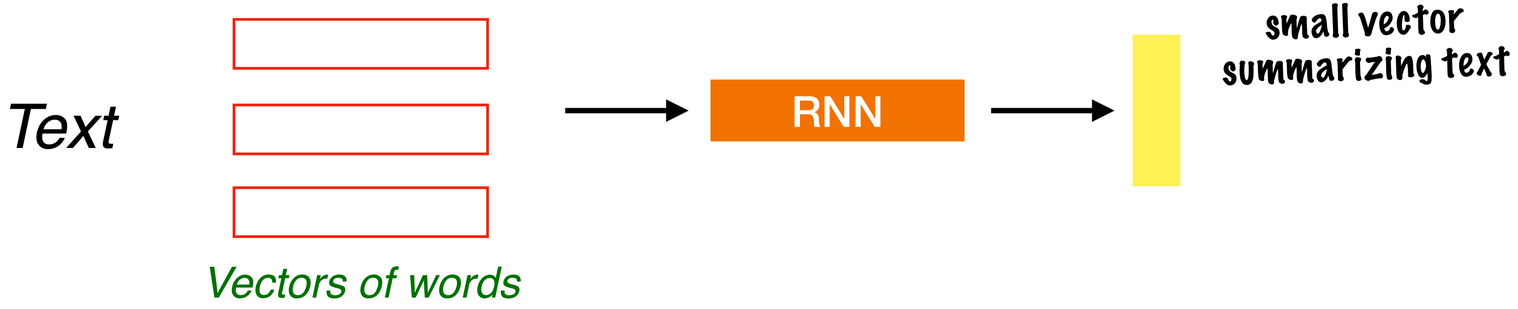
Think of it as a tree where the inputs arrive at leaves

We will reason about different parts of the input individually, and then start aggregating this information towards the root of the tree

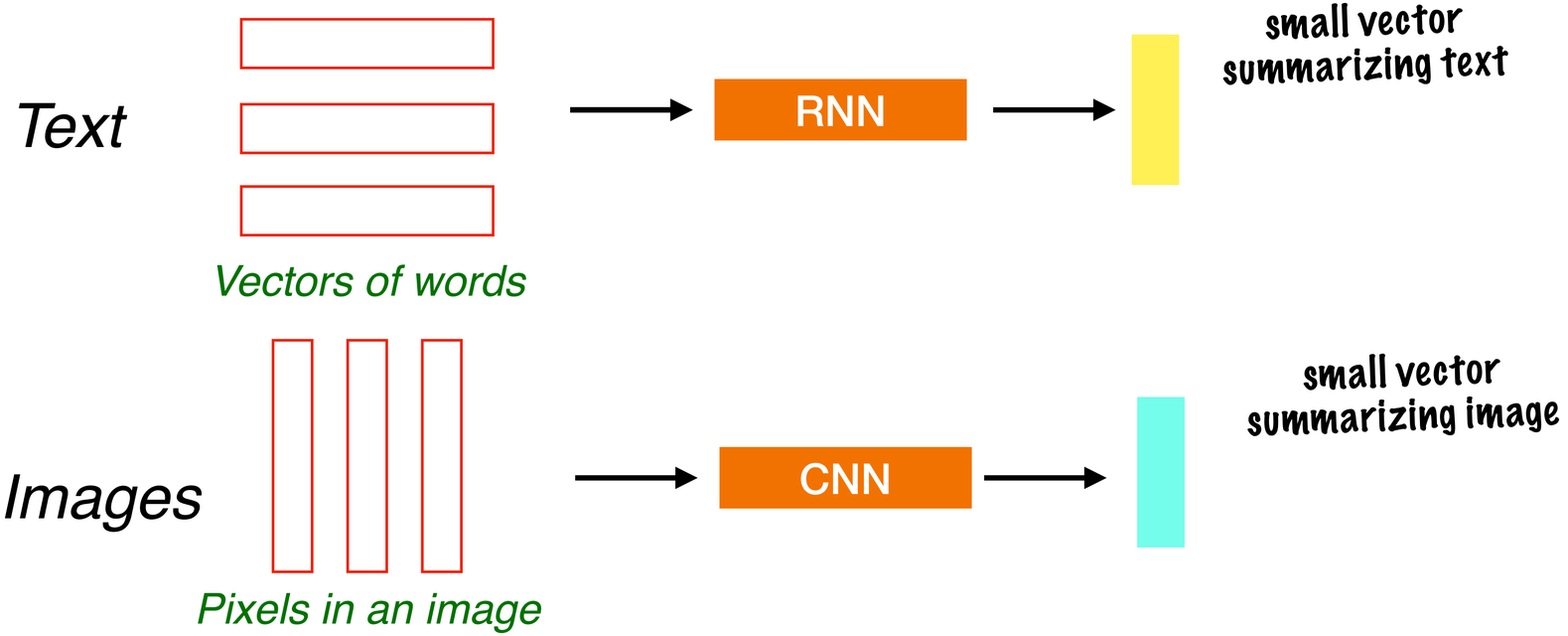
Prediction based on summarized information at the root

Each submodule encodes specific part of the input and different submodules might be different kinds of neural networks

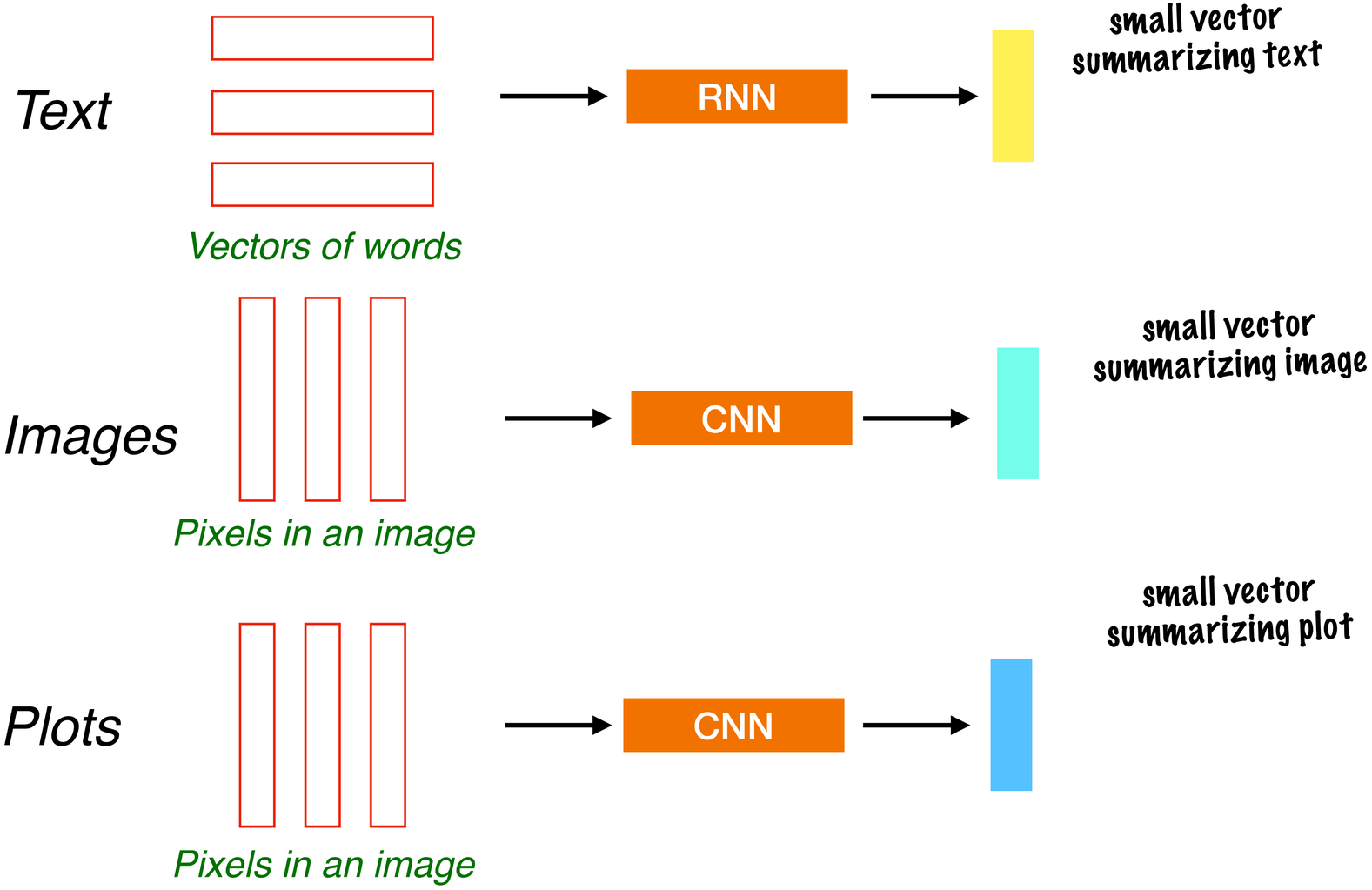
Designing DL-based Models for Software Development Tasks



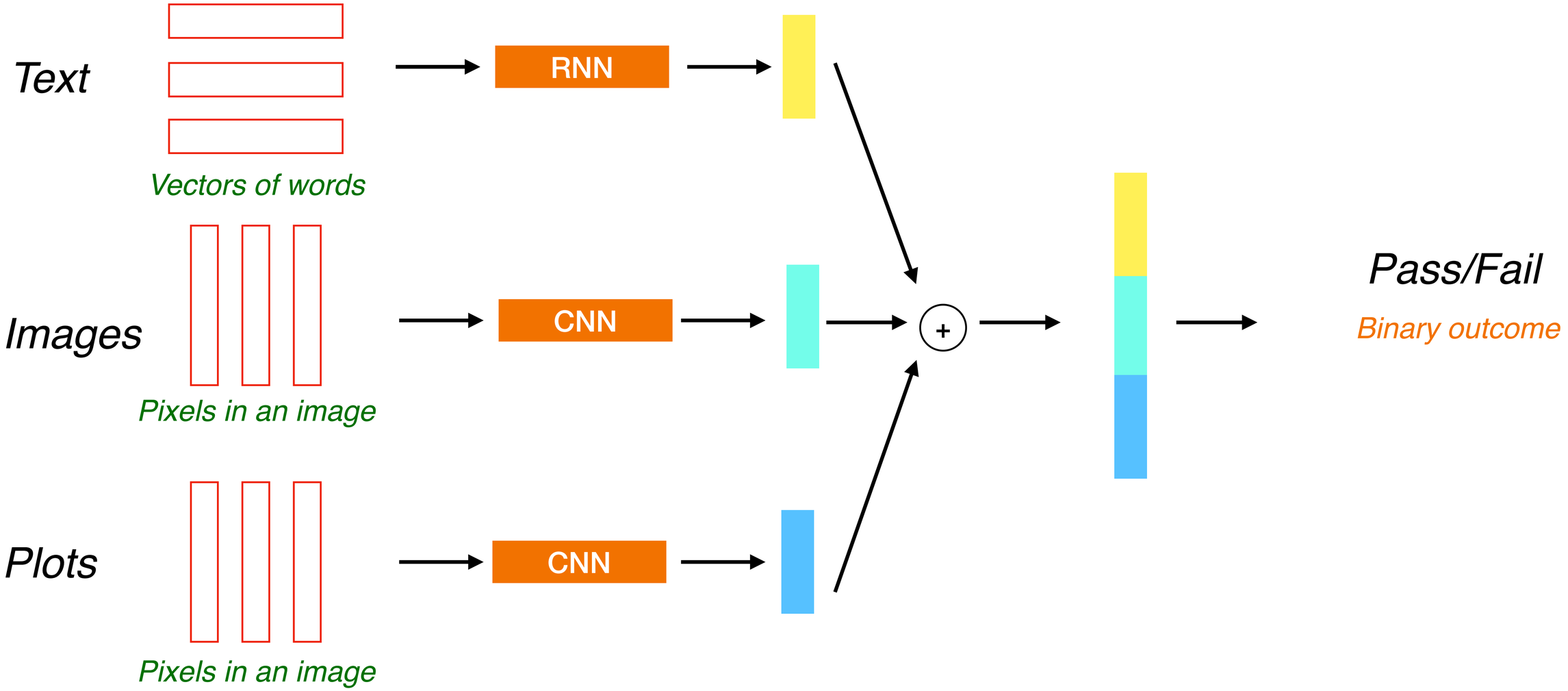
Designing DL-based Models for Software Development Tasks



Designing DL-based Models for Software Development Tasks



Designing DL-based Models for Software Development Tasks



Training a Hierarchical Neural Network

Option 1: ***Train each submodule separately***

Training focuses on a specific model and its input

Need training data for each submodel

Submodule isn't aware of the overall task

Training a Hierarchical Neural Network

Option 1: *Train each submodule separately*

Training focuses on a specific model and its input

Need training data for each submodel

Submodule isn't aware of the overall task

Option 2: *Train entire model jointly*

Need training data only for the overall task

Submodels get optimized for the overall task

For large models, feedback from final prediction may get lost

Training a Hierarchical Neural Network

Option 1: *Train each submodule separately*

Training focuses on a specific model and its input

Need training data for each submodel

Submodule isn't aware of the overall task

Option 2: *Train entire model jointly*

Need training data only for the overall task

Submodels get optimized for the overall task

For large models, feedback from final prediction may get lost

Option 3: *First training bout separate — Second training bout, we calibrate the whole network*

HNN-based Software Analysis

Designing DL-based Models for Software Development Tasks

arXiv:2003.05620v1 [cs.SE] 12 Mar 2020

CC2Vec: Distributed Representations of Code Changes

Thong Hoang, Hong Jin Kang, David Lo
Singapore Management University, Singapore
{vdthoang.2016,hjkang.2018,davidlo}@smu.edu.sg

Julia Lawall
Sorbonne University/Inria/LIP6, France
Julia.Lawall@inria.fr

ABSTRACT

Existing work on software patches often use features specific to a single task. These works often rely on manually identified features, and human effort is required to identify these features for each task. In this work, we propose CC2Vec, a neural network model that learns a representation of code changes guided by their accompanying log messages, which represent the semantic intent of the code changes. CC2Vec models the hierarchical structure of a code change with the help of the attention mechanism and uses multiple comparison functions to identify the differences between the removed and added code.

To evaluate if CC2Vec can produce a distributed representation of code changes that is general and useful for multiple tasks on software patches, we use the vectors produced by CC2Vec for three tasks: log message generation, bug fixing patch identification, and just-in-time defect prediction. In all tasks, the models using CC2Vec outperform the state-of-the-art techniques.

1 INTRODUCTION

Patches, used to edit source code, are often created by developers to describe new features, fix bugs, or maintain existing functionality (e.g., API updates, refactoring, etc.). Patches contain two main pieces of information, a log message and a code change. The log message, used to describe the semantics of the code changes, is written in natural language by the developers. The code change indicates the lines of code to remove or add across one or multiple files. Research has shown that the study of historical patches can be employed to solve software engineering problems, such as just-in-time defect prediction [21, 28], identification of bug fixing patches [22, 57], tangled change prediction [34], recommendation of a code reviewer for a patch [50], and many more.

Exploring patches to solve software engineering problems requires choosing a representation of the patch data. Most prior work involves manually crafting a set of features to represent a patch and using these features for further processing [28–30, 44, 57, 60]. These features have mostly been extracted from properties of patches, such as the modifications to source code (e.g., number of removed and added lines, the number of files modified), the history of changes (e.g., the number of prior or recent changes to the updated files), the record of patch authors and reviewers (e.g., the number of developers or reviewers who contributed to the patch), etc. These features can be used as an input to a machine learning classifier (e.g., Support Vector Machine, Logistic Regression, Random Forest, etc.) to address various software engineering tasks [28, 34, 50, 57]. Extracting a suitable vector representation to represent the "meaning" of a patch is certainly crucial. Intuitively, the quality of a patch representation plays a major role in determining the eventual learning outcome.

In this paper, to boost the effectiveness of existing solutions that employ the properties of patches, we wish to learn vector representations of the code changes in patches that can be used for a number of tasks. We propose a new deep learning architecture named CC2Vec that can effectively embed a code change into a vector space where similar changes are close to each other. As log messages, written by developers, are used to describe the semantics of the code change, we use them to supervise the learning of code changes' representations from patches. Specifically, CC2Vec optimizes the vector representation of a code change in a patch to predict appropriate words, extracted from the first line of the log message. We consider only the first line, as it is the focus of many prior works [39, 49], and is considered to carry the most semantic meaning with the least noise.[1]

CC2Vec analyzes the code change, i.e., scattered fragments of removed and added code across multiple files. Code removed or added from a file follows a hierarchical structure (words form line, lines form hunks). Recent work has suggested that the attention mechanism can help in modelling structural dependencies [3, 32], thus, we hypothesize that the attention mechanism may be effective for modelling the structure of a code change. We propose a specialized hierarchical attention network (HAN) to construct a vector representation of the removed code (and another for the added code) of each affected file in a given patch. Our HAN first builds vector representations of lines; these vectors are then used to construct vector representations of hunks; and we then aggregate these vectors to construct the embedding vector of the removed or added code. Next, we employ multiple comparison functions to capture the difference between two embedding vectors representing removed and added code. This produces features representing the relationship between the removed and added code. Each comparison function produces a vector and these vectors are then concatenated to form an embedding vector for the affected file. Finally, the embedding vectors of all the affected files are concatenated to build a vector representation of the code change in a patch. After training is completed, CC2Vec can be used to extract representations of code changes even from patches with empty or meaningless log messages (which are common in practice [26, 38, 39]). CC2Vec is also programming-language agnostic; one can use it to learn vector representations of code changes for any language.

[1] https://chris.beams.io/posts/git-commit/

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380361

The goal of this paper is to provide a proper representation of code changes that would allow us to automate tasks related to code changes. Predict what the commit message should be. In other words, it predicts whether a code change introduces a bug, etc.

Designing DL-based Models for Software Development Tasks

```
platforms.

Cc: José Roberto de Souza <jose.souza@intel.com>
Fixes: d506a65 ("drm/i915: Catch GTT fault errors for gen11+ planes")
Signed-off-by: Matt Roper <matthew.d.roper@intel.com>
Link: https://patchwork.freedesktop.org/patch/msgid/20200424231423.4065231-1-matthew.d.roper@intel.com
Reviewed-by: Ville Syrjälä <ville.syrjala@linux.intel.com>
(cherry picked from commit 869129e)
Signed-off-by: Rodrigo Vivi <rodrigo.vivi@intel.com>
```

master
v6.3-rc7 ... v5.7-rc4

mattrope authored and **rodrigovivi** committed on Apr 29, 2020 — 1 parent 2abaad4 commit 8598eb7

Showing **1 changed file** with **2 additions** and **4 deletions**. Split | Unified

6 drivers/gpu/drm/i915/i915_irq.c

```
@@ -3358,7 +3358,8 @@ static void gen8_de_irq_postinstall(struct drm_i915_private *dev_priv)
 {
         struct intel_uncore *uncore = &dev_priv->uncore;

-        u32 de_pipe_masked = GEN8_PIPE_CDCLK_CRC_DONE;
+        u32 de_pipe_masked = gen8_de_pipe_fault_mask(dev_priv) |
+                GEN8_PIPE_CDCLK_CRC_DONE;
         u32 de_pipe_enables;
         u32 de_port_masked = GEN8_AUX_CHANNEL_A;
         u32 de_port_enables;
@@ -3369,13 +3370,10 @@ static void gen8_de_irq_postinstall(struct drm_i915_private *dev_priv)
                 de_misc_masked |= GEN8_DE_MISC_GSE;

         if (INTEL_GEN(dev_priv) >= 9) {
-                de_pipe_masked |= GEN9_DE_PIPE_IRQ_FAULT_ERRORS;
                 de_port_masked |= GEN9_AUX_CHANNEL_B | GEN9_AUX_CHANNEL_C |
                                   GEN9_AUX_CHANNEL_D;
                 if (IS_GEN9_LP(dev_priv))
                         de_port_masked |= BXT_DE_PORT_GMBUS;
-        } else {
-                de_pipe_masked |= GEN8_DE_PIPE_IRQ_FAULT_ERRORS;
         }

         if (INTEL_GEN(dev_priv) >= 11)
```

Designing DL-based Models for Software Development Tasks

platforms.

Cc: José Roberto de Souza <jose.souza@intel.com>
Fixes: d506a65 ("drm/i915: Catch GTT fault errors for gen11+ planes")
Signed-off-by: Matt Roper <matthew.d.roper@intel.com>
Link: https://patchwork.freedesktop.org/patch/msgid/20200424231423.4065231-1-matthew.d.roper@intel.com
Reviewed-by: Ville Syrjälä <ville.syrjala@linux.intel.com>
(cherry picked from commit 869129e)
Signed-off-by: Rodrigo Vivi <rodrigo.vivi@intel.com>

master
v6.3-rc7 ... v5.7-rc4

mattrope authored and **rodrigovivi** committed on Apr 29, 2020 — 1 parent 2abaad4 commit 8598eb7

Showing **1 changed file** with **2 additions** and **4 deletions**. Split Unified

6 drivers/gpu/drm/i915/i915_

Given a new code change with an unknown log message, we find the code changes with a known log message that have the closest CC2Vec vector

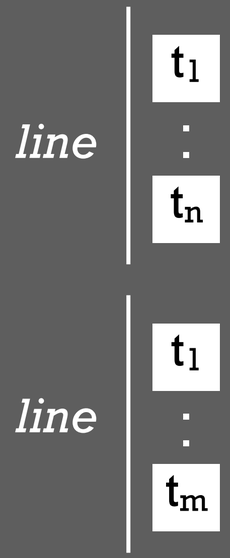
```
@@ -3358,7 +3358,8 @@ stati
3358 3358  {
3359 3359          struct intel_uncore
3360 3360
3361      -         u32 de_pipe_masked
     3361 +         u32 de_pipe_masked
     3362 +                 GEN8_PIPE_C
3362 3363          u32 de_pipe_enables;
3363 3364          u32 de_port_masked = GEN8_AUX_CHANNEL_A;
3364 3365          u32 de_port_enables;
@@ -3369,13 +3370,10 @@ static void gen8_de_irq_postinstall(struct drm_i915_private *dev_priv)
3369 3370                  de_misc_masked |= GEN8_DE_MISC_GSE;
3370 3371
3371 3372          if (INTEL_GEN(dev_priv) >= 9) {
3372      -                 de_pipe_masked |= GEN9_DE_PIPE_IRQ_FAULT_ERRORS;
3373 3373                  de_port_masked |= GEN9_AUX_CHANNEL_B | GEN9_AUX_CHANNEL_C |
3374 3374                                    GEN9_AUX_CHANNEL_D;
3375 3375                  if (IS_GEN9_LP(dev_priv))
3376 3376                          de_port_masked |= BXT_DE_PORT_GMBUS;
3377      -         } else {
3378      -                 de_pipe_masked |= GEN8_DE_PIPE_IRQ_FAULT_ERRORS;
3379 3377          }
3380 3378
3381 3379          if (INTEL_GEN(dev_priv) >= 11)
```

Designing DL-based Models for Software Development Tasks

CC2Vec

Designing DL-based Models for Software Development Tasks

CC2Vec



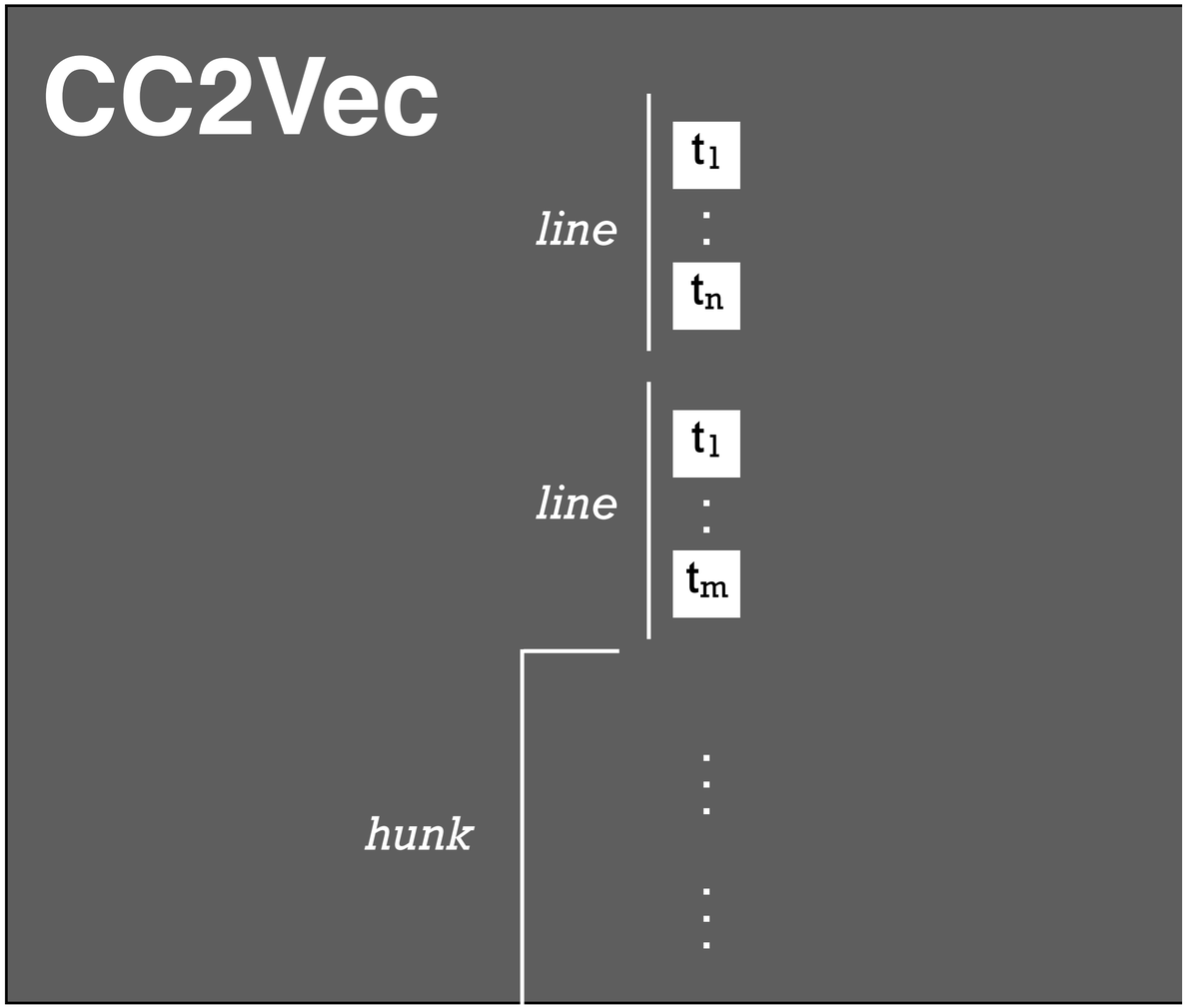
Designing DL-based Models for Software Development Tasks

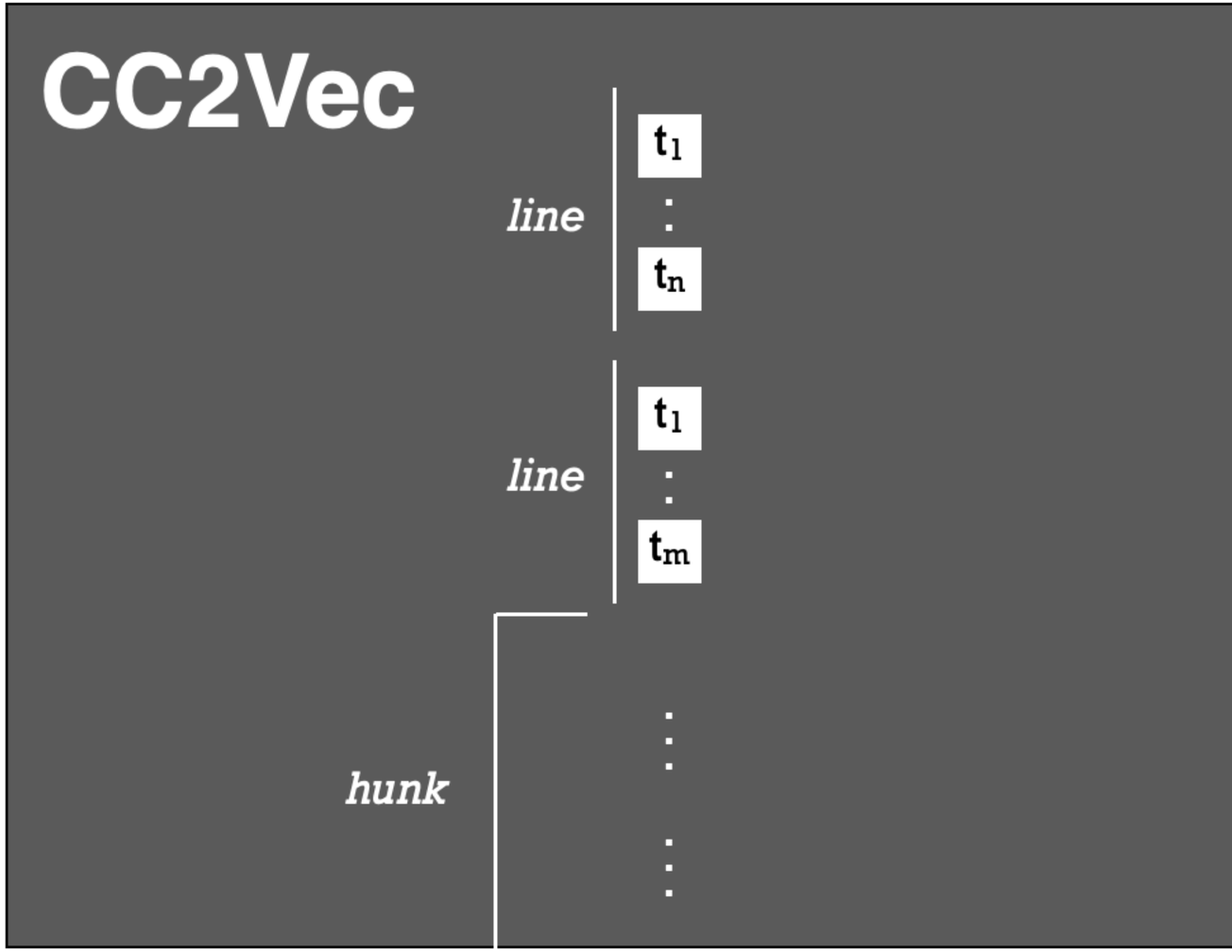
Designing DL-based Models for Software Development Tasks

CC2Vec

In a **diff, a hunk** is basically a chunk of consecutive changes, lines that were added, removed, or modified, shown together with a few unchanged lines around them for context

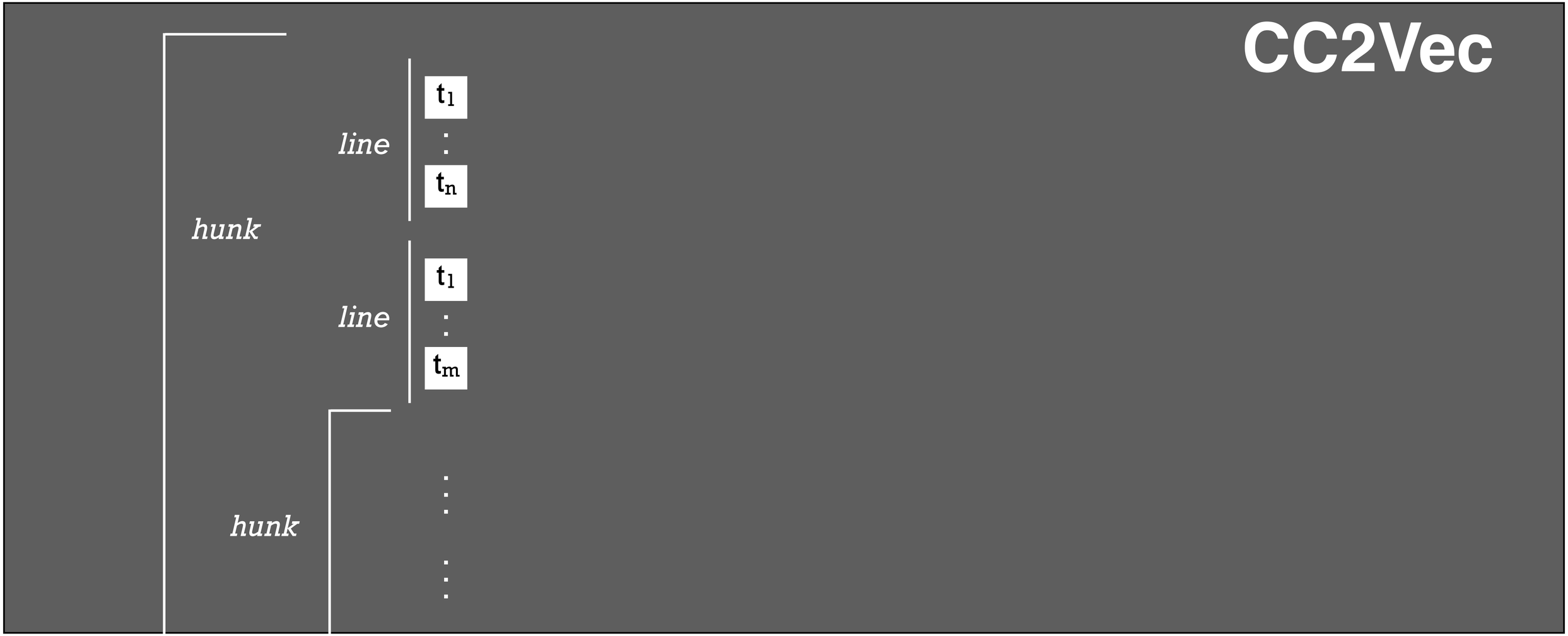
Designing DL-based Models for Software Development Tasks





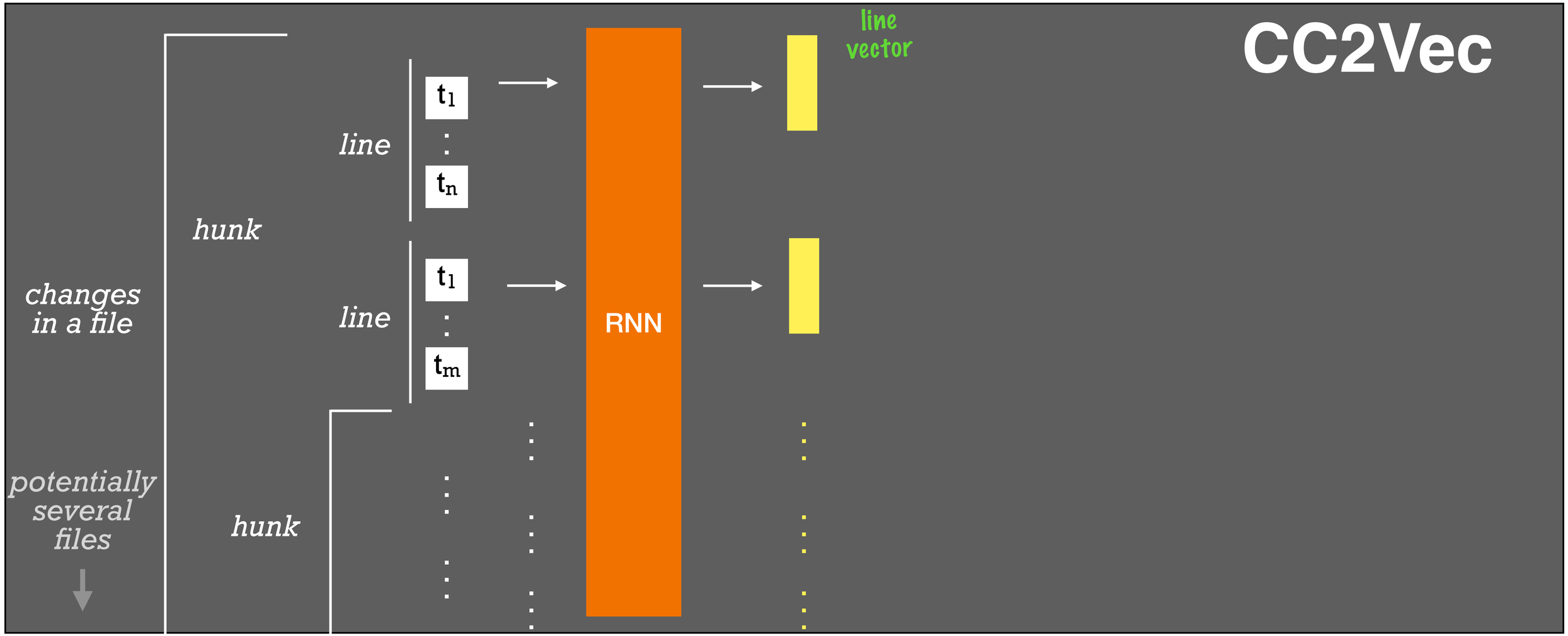
antoniomastropaolo.com
aura-se-lab.github.io

Designing DL-based Models for Software Development Tasks

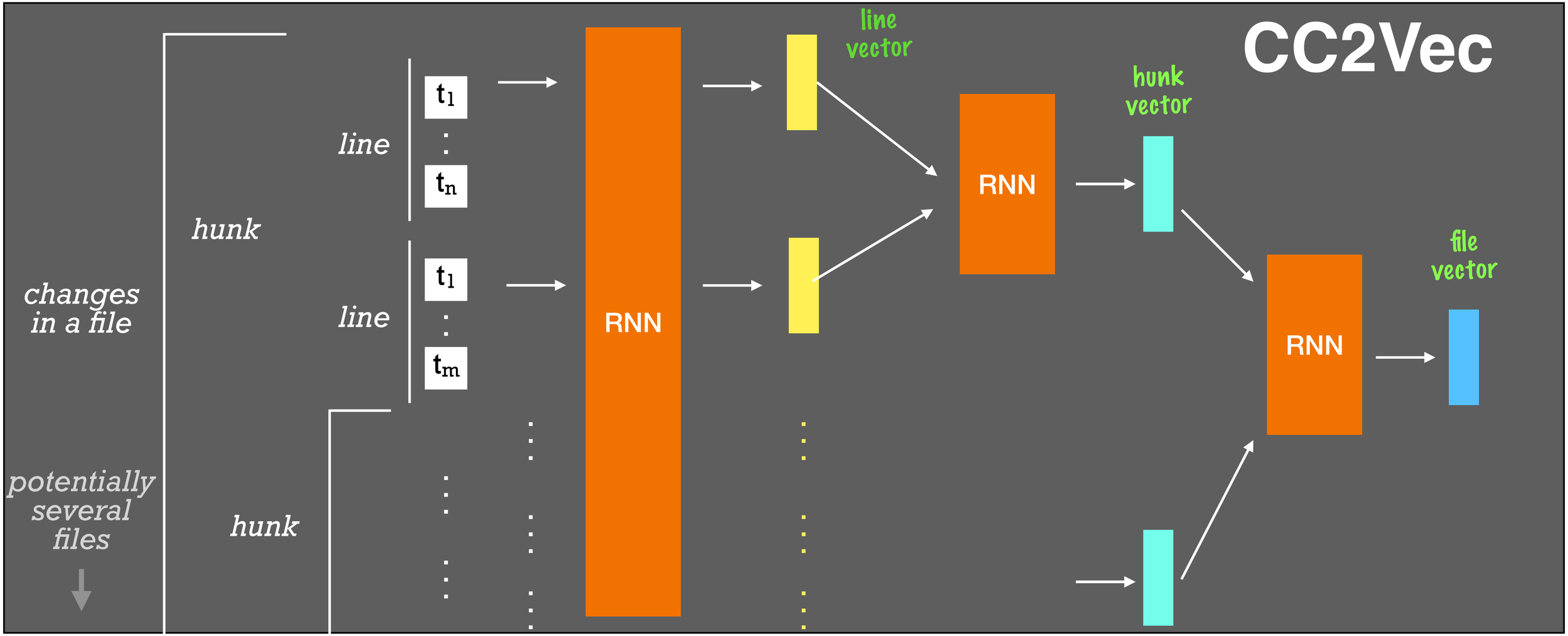


Designing DL-based Models for Software Development Tasks

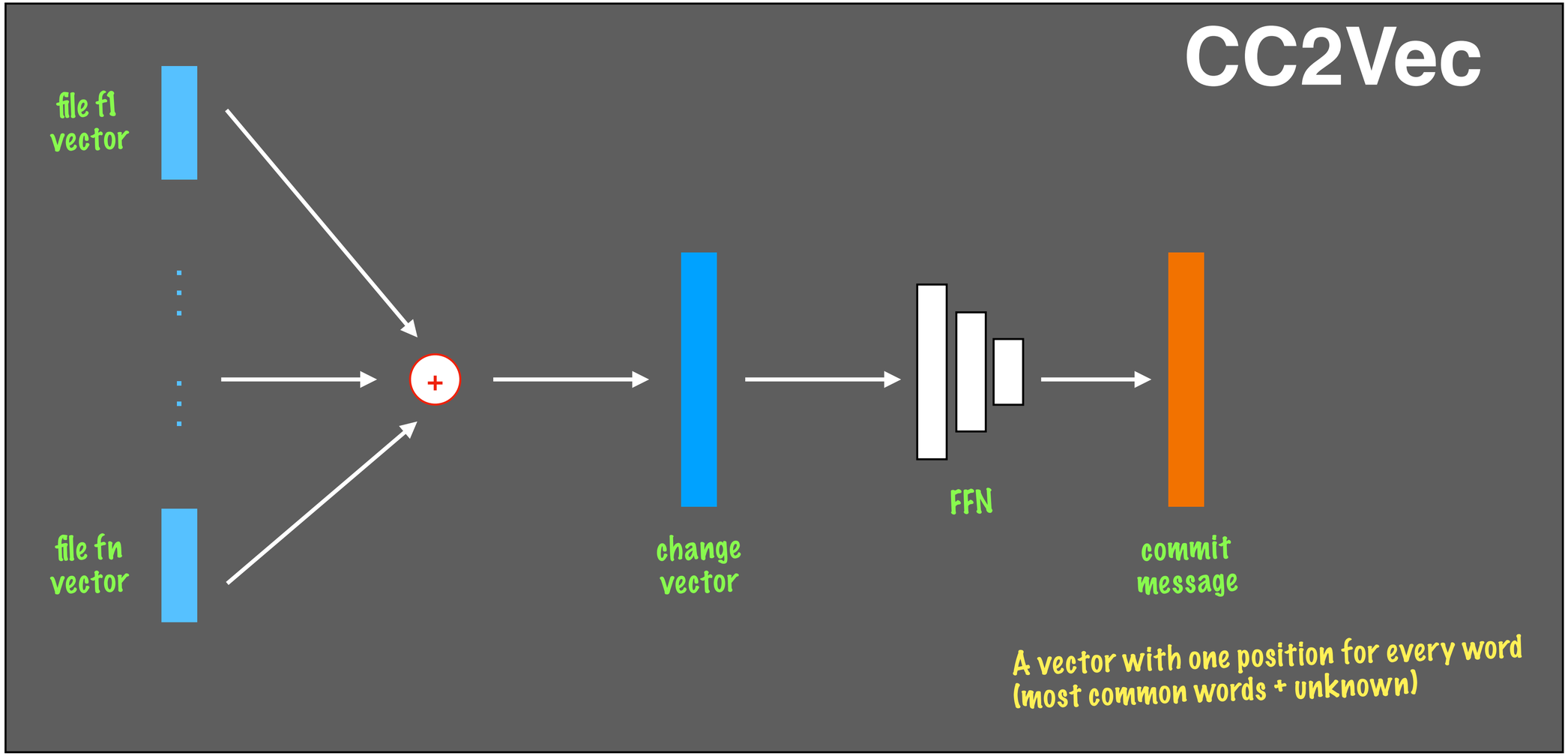
Designing DL-based Models for Software Development Tasks



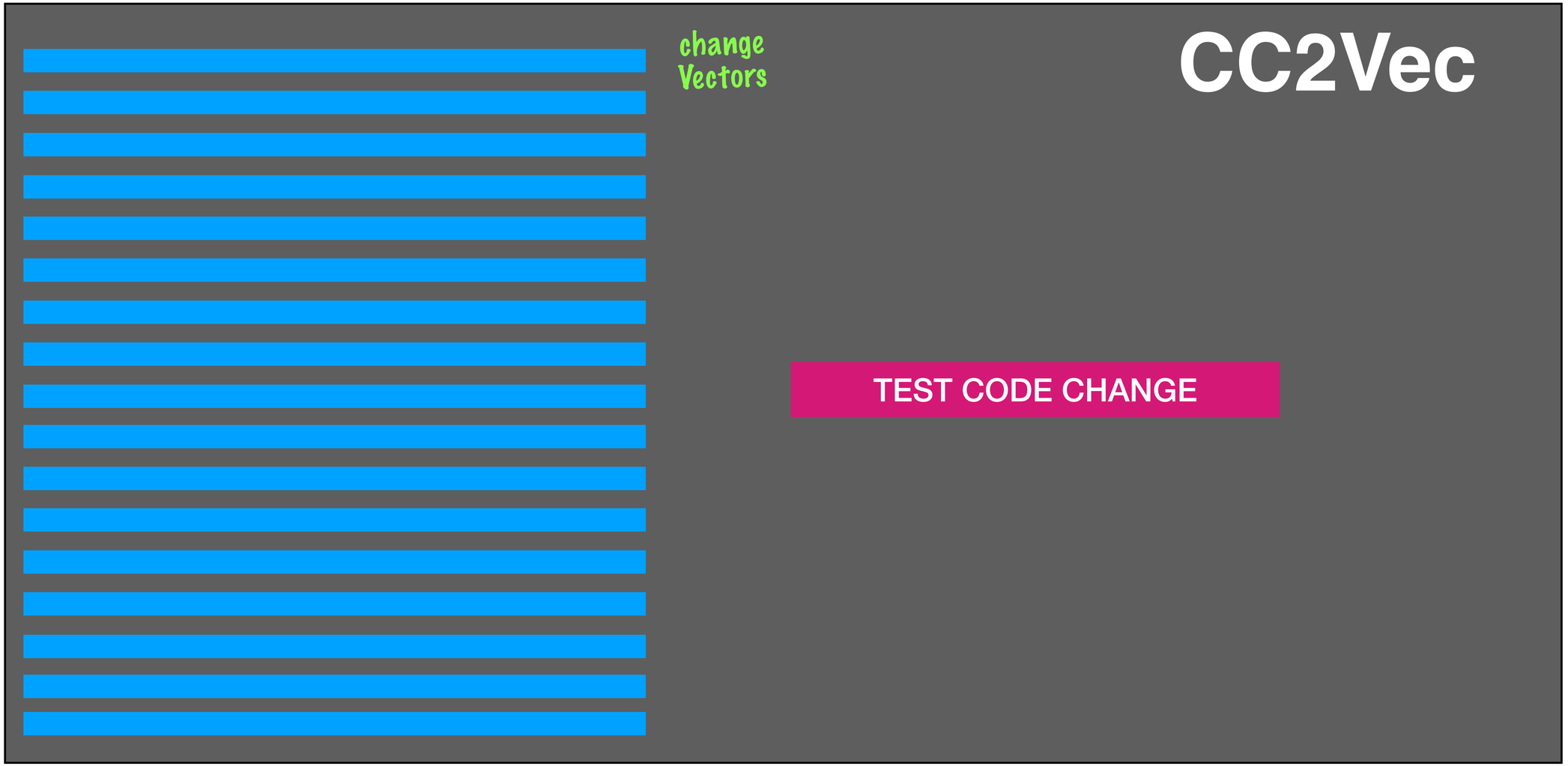
Designing DL-based Models for Software Development Tasks



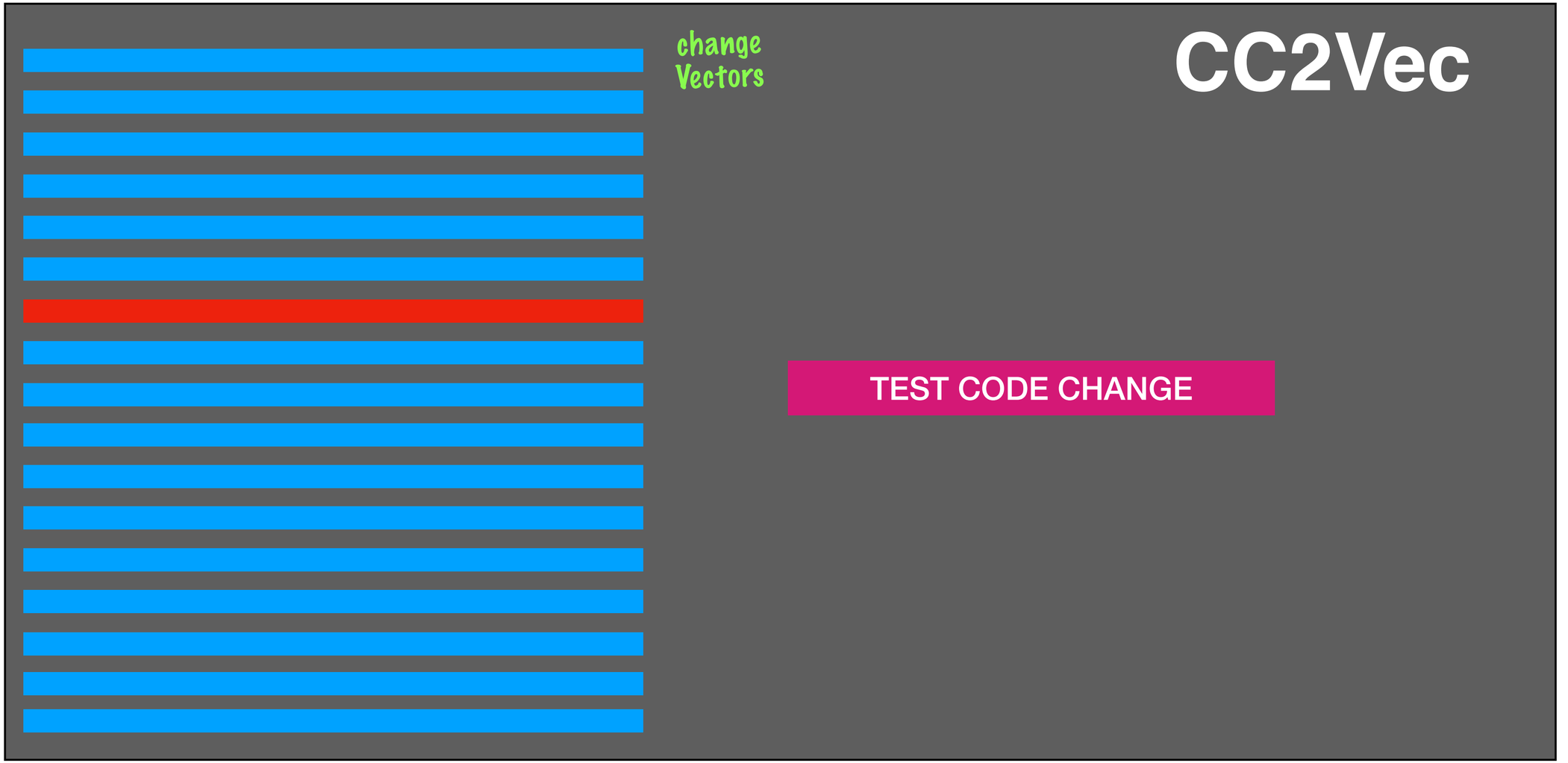
Designing DL-based Models for Software Development Tasks



Designing DL-based Models for Software Development Tasks



Designing DL-based Models for Software Development Tasks



Designing DL-based Models for Software Development Tasks