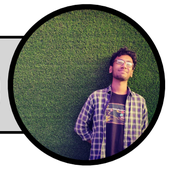
Pre-trained Models for SD Activities

Dr. Antonio Mastropaolo

Instructor

Mr. Alvi Haque



Teaching Assistant

WILLIAM & MARY

CHARTERED 1693

Spring 2026

Pre-trained Models for SD Activities

Transformer Model

Vaswani et al.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

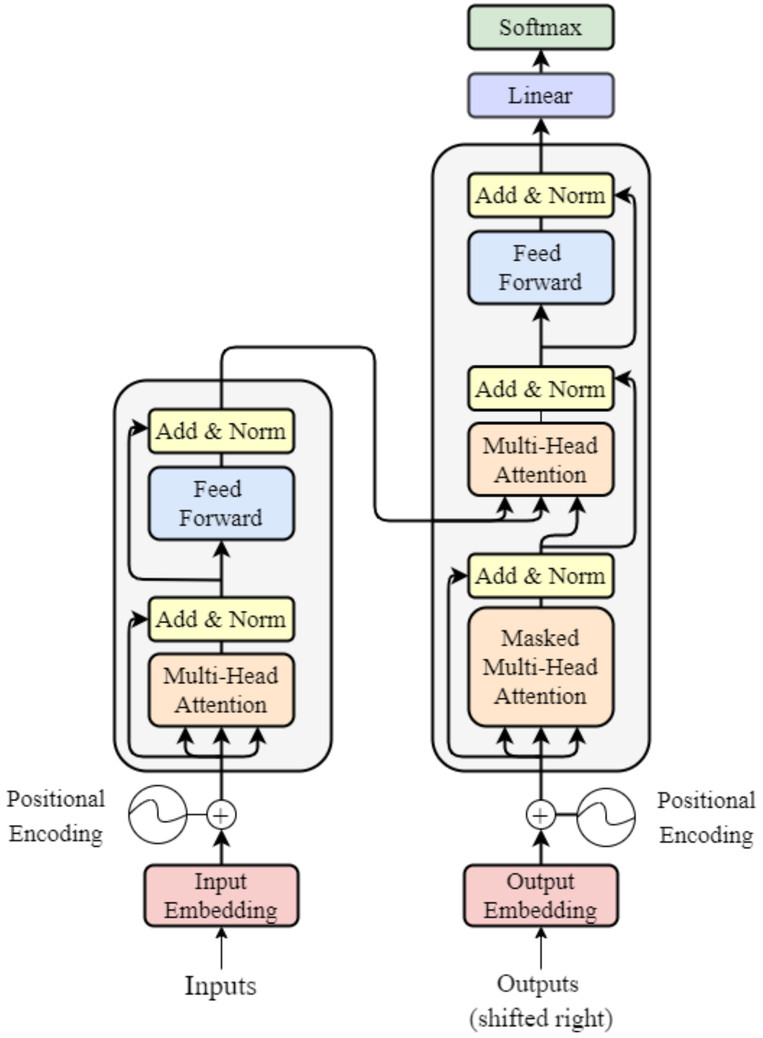
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention

Pre-trained Models for SD Activities



Pre-trained Models for SD Activities

Transformer Model

Vaswani et al.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion J Google **n** llion@g**e.com

idan N. Gomez* † **ersity of Toronto** ai**cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

llia Polosukhin* ‡ ill**olosukhin@gmail.com

Abstract

The dominant sequence tra**ction models are based on complex recurrent or convolutional neural net**d a decoder. The best performing models also **through an attention mechanism. We propo**ure, the Transformer

Pre-trained Models for SD Activities

Long-Range Dependencies

```
public static Inet4Address getEmbeddedIPv4ClientAddress
                (Inet6Address ip) {
     ip1 = New IP(“127.0.0.1”)
     if (isCompatIPv4Address(ip)) {
         return getCompatIPv4Address(ip);
     }if (is6to4Address(ip)) {
         return get6to4IPv4Address(ip);
     }
     if (isTeredoAddress(ip)) {
         return getTeredoInfo(ip).getClient();
     }
     return ip1.getDefault();
}
```

Pre-trained Models for SD Activities

Long-Range Dependencies

```
public static Inet4Address getEmbeddedIPv4ClientAddress
                (Inet6Address ip) {
     ip1 = New IP(“127.0.0.1”)
     if (isCompatIPv4Address(ip)) {
         return getCompatIPv4Address(ip);
     }if (is6to4Address(ip)) {
         return get6to4IPv4Address(ip);
     }
     if (isTeredoAddress(ip)) {
         return getTeredoInfo(ip).getClient();
     }
     return ip1.getDefault();
}
```

Pre-trained Models for SD Activities

Long-Range Dependencies

```
public static Inet4Address getEmbeddedIPv4ClientAddress
                (Inet6Address ip) {
     ip1 = New IP(“127.0.0.1”)
     if (isCompatIPv4Address(ip)) {
         return getCompatIPv4Address(ip);
     }if (is6to4Address(ip)) {
         return get6to4IPv4Address(ip);
     }
     if (isTeredoAddress(ip)) {
         return getTeredoInfo(ip).getClient();
     }
     return ip1.getDefault();
}
```

Pre-trained Models for SD Activities

Pre-train—then—Fine-tune

Pre-training: Instill as much general knowledge as possible into the model, such as the structure and patterns of a language either natural or formal

Pre-trained Models for SD Activities

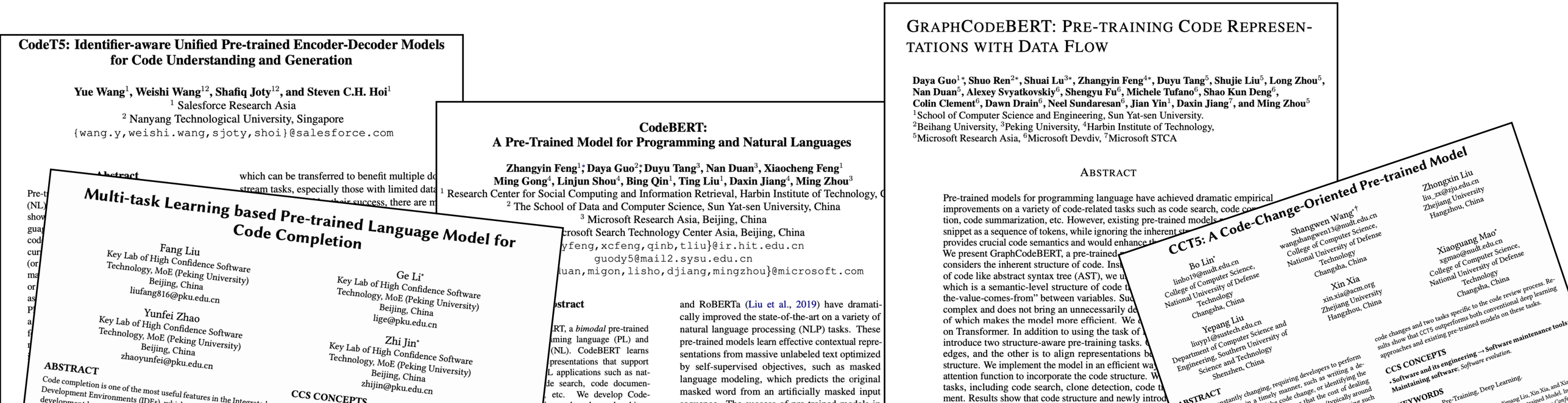
Pre-train—then—Fine-tune

Fine-tuning: The model is specialized for automating a specific tasks at hand i.e., sentiment analysis

Pre-trained Models for SD Activities

A pre-trained model is a deep learning model that has been trained on a vast dataset, allowing the transfer of the knowledge it has gained to be re-used during the fine-tuning stage.

Pre-trained Models for SD Activities



CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation

Yue Wang[1], **Weishi Wang**[12], **Shafiq Joty**[12], **and Steven C.H. Hoi**[1]
[1] Salesforce Research Asia
[2] Nanyang Technological University, Singapore
{wang.y,weishi.wang,sjoty,shoi}@salesforce.com

Multi-task Learning based Pre-trained Language Model for Code Completion

Fang Liu
Key Lab of High Confidence Software Technology, MoE (Peking University)
Beijing, China
liufang816@pku.edu.cn

Ge Li*
Key Lab of High Confidence Software Technology, MoE (Peking University)
Beijing, China
lige@pku.edu.cn

Yunfei Zhao
Key Lab of High Confidence Software Technology, MoE (Peking University)
Beijing, China
zhaoyunfei@pku.edu.cn

Zhi Jin*
Key Lab of High Confidence Software Technology, MoE (Peking University)
Beijing, China
zhijin@pku.edu.cn

CodeBERT:
A Pre-Trained Model for Programming and Natural Languages

Zhangyin Feng[1]*, **Daya Guo**[2]*, **Duyu Tang**[3], **Nan Duan**[3], **Xiaocheng Feng**[1]
Ming Gong[4], **Linjun Shou**[4], **Bing Qin**[1], **Ting Liu**[1], **Daxin Jiang**[4], **Ming Zhou**[3]
[1] Research Center for Social Computing and Information Retrieval, Harbin Institute of Technology,
[2] The School of Data and Computer Science, Sun Yat-sen University, China
[3] Microsoft Research Asia, Beijing, China
crosoft Search Technology Center Asia, Beijing, China
yfeng,xcfeng,qinb,tliu}@ir.hit.edu.cn
guody5@mail2.sysu.edu.cn
tuan,migon,lisho,djiang,mingzhou}@microsoft.com

GRAPHCODEBERT: PRE-TRAINING CODE REPRESENTATIONS WITH DATA FLOW

Daya Guo[1]*, **Shuo Ren**[2]*, **Shuai Lu**[3]*, **Zhangyin Feng**[4]*, **Duyu Tang**[5], **Shujie Liu**[5], **Long Zhou**[5], **Nan Duan**[5], **Alexey Svyatkovskiy**[6], **Shengyu Fu**[6], **Michele Tufano**[6], **Shao Kun Deng**[6], **Colin Clement**[6], **Dawn Drain**[6], **Neel Sundaresan**[6], **Jian Yin**[1], **Daxin Jiang**[7], **and Ming Zhou**[5]
[1]School of Computer Science and Engineering, Sun Yat-sen University.
[2]Beihang University, [3]Peking University, [4]Harbin Institute of Technology,
[5]Microsoft Research Asia, [6]Microsoft Devdiv, [7]Microsoft STCA

CCT5: A Code-Change-Oriented Pre-trained Model

Bo Lin*
linbo19@nudt.edu.cn
College of Computer Science, National University of Defense Technology
Changsha, China

Shangwen Wang*†
wangshangwen13@nudt.edu.cn
College of Computer Science, National University of Defense Technology
Changsha, China

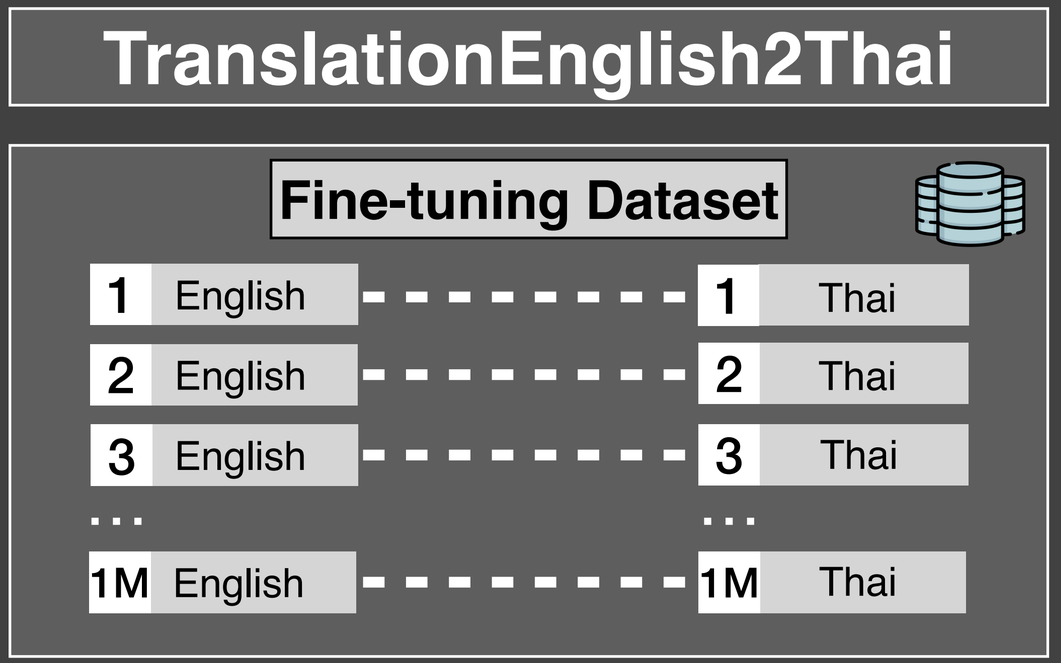
Zhongxin Liu
liu_zx@zju.edu.cn
Zhejiang University
Hangzhou, China

Xiaoguang Mao*
xgmao@nudt.edu.cn
College of Computer Science, National University of Defense Technology
Changsha, China

Yepang Liu
liuyp1@sustech.edu.cn
Department of Computer Science and Engineering, Southern University of Science and Technology
Shenzhen, China

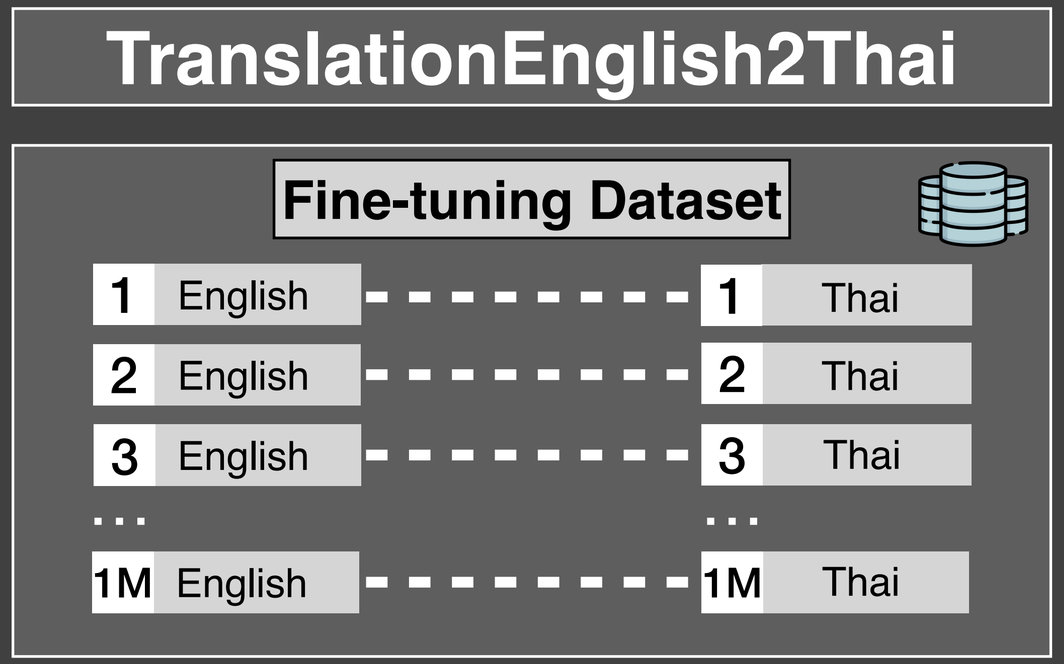
Xin Xia
xin.xia@acm.org
Zhejiang University
Hangzhou, China

Pre-trained Models for SD Activities



Pre-trained Models for SD Activities

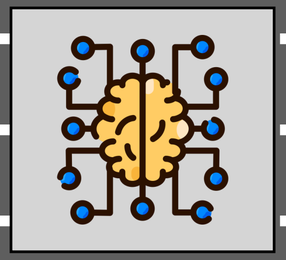
Supervised Dataset



Pre-trained Models for SD Activities

Mask Language Modeling

Pre-Training Procedure

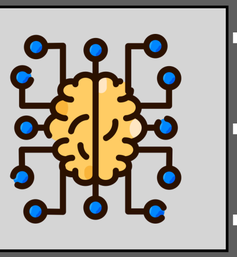
1	The new ***MASK*** is currently under ***MASK***		1	***item, observation***
2	ชื่อขอ ***MASK*** นคือ		2	***งฉั***
3	***MASK*** new ***MASK*** are upsetting and bad		3	***The, Policies***

Pre-trained Models for SD Activities

Mask Language Modeling

Pre-Training Procedure

Self-supervised Dataset

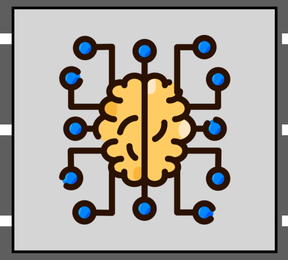
1	The new ***MASK*** is currently under ***MASK***		1	***item, observation***
2	ชื่อขอ ***MASK*** นคือ		2	***งฉั***
3	***MASK*** new ***MASK*** are upsetting and bad		3	***The, Policies***

Pre-trained Models for SD Activities

Mask Language Modeling

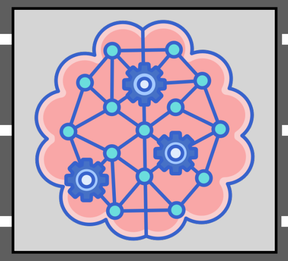
Pre-Training Procedure

Self-supervised Dataset

1	The new ***MASK*** is currently under ***MASK***		1	***item, observation***
2	ชื่อขอ ***MASK*** นคือ		2	งฉั
3	***MASK*** new ***MASK*** are upsetting and bad		3	***The, Policies***

Fine-tuning Procedure

Supervised Dataset

1	Today is a sunny day, so they say		1	วันนี้เป็นวันที่มีแดดดังนั้นพวกเขาจึงพูดว่า
2	The big red fox is on the table		2	จิ้งจอกแดงตัวใหญ่อยู่บนโต๊ะ
3	I am not asking much really		3	ฉันไม่ได้ถามอะไรมากจริงๆ

Pre-trained Models for SD Activities

2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)

Leveraging Small Software Engineering Data Sets with Pre-trained Neural Networks

Romain Robbes
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

Andrea Janes
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

***Abstract*—Many software engineering data sets, particularly those that demand manual labelling for classification, are necessarily small. As a consequence, several recent software engineering papers have cast doubt on the effectiveness of deep neural networks for classification tasks, when applied to these data sets. We provide initial evidence that recent advances in Natural Language Processing, that allow neural networks to leverage large amount of unlabelled data in a pre-training phase, can significantly improve performance.**

***Index Terms*—Data Sets, Deep Learning, Transfer Learning**

I. Introduction

Deep neural networks have revolutionized the field of machine learning, leading to significant improvements over the state of the art in a large variety of tasks and domains (including image recognition, speech recognition, machine translation, summarization, and others). In Software Engineering, the results have been more mixed. While many approaches have promising results (e.g., code completion, variable naming, code translation and summarization), several recent papers have cast doubt about the effectiveness of neural networks in some settings [1], [2]. One reason is the performance on small SE data sets, which we examine in this paper.

The use cases where deep neural networks outperform the state of the art are those where large amount of data is available, such as the ImageNet data set [3], which features more than 14 million labelled images in its largest version. However, many data sets are much smaller, particularly in the case of supervised learning, as each data point must be manually labelled. This manual task is often an expensive process, and all the more in Software Engineering: while ImageNet labelling can be crowdsourced, labelling a SE data set requires significant expertise. It cannot be crowdsourced easily, and is thus vastly more expensive. In practice, many SE data sets are manually labelled by researchers, and are thus limited to hundreds or thousands of data points only. Recent work reports disappointing performance when training deep neural networks from scratch on small SE data sets.

In this paper, we investigate whether the small data set issue can be alleviated in Software Engineering, starting with natural language (NL) SE datasets (we are actively exploring the source code case). We show that a form of transfer learning, namely unsupervised pre-training on large data sets, can be effectively used to improve the performance on small NL SE data sets. Several approaches—detailed in Section III—have shown this strategy to be effective in Computer Vision and (lately) in Natural Language Processing. These approaches are all a variation of a common idea: when training a neural network from scratch, all the model's weights are initialized randomly. Pre-training instead initializes the weights with values obtained by training a model on a related tasks, for which data is easier to gather; in this case, by using large amounts of unlabelled data, no manual labelling is necessary.

II. Small Data Sets in Software Engineering

Many labelling tasks, such as identifying objects in images, drawing bounding boxes for these objects, or identifying the sentiment of a movie review, do not require specific expertise. They can hence be crowdsourced at a reasonable cost, particularly thanks to crowdsourcing platforms such as Amazon Mechanical Turk.

As manual labelling in Software Engineering requires expertise in Software Engineering, it is usually done by the researchers themselves. This approach is expensive, and due to the amount of effort involved, is often limited to hundreds or thousands of data points. This is all the more true since the effort is often duplicated to verify whether annotators are in agreement with each other. Some examples follow.

Bacchelli et al. manually linked 2,139 emails from 6 open-source projects with the source code entities they referenced [4]. Fakhoury et al. labelled 1,700 Java methods (with comments) according to the linguistic antipatterns they exhibited [2]. Ortu et al. labelled 2,000 JIRA issue reports, and 4,000 sentences with the sentiments expressed in them. Novielli et al. labelled 4,423 stack Overflow posts with sentiment [5]. Lin et al. labelled the sentiment of *each word* of 1,500 Stack Overflow sentences according to their sentiment [1]; they also labelled 341 app reviews for sentiment. Villaroel et al. [6] classified 1,200 app reviews, while Maalej at al. [7] classified 4,400 app reviews, in both cases in categories such as bug reports, and feature requests. Zhou et al. hired students to analyze 1,674 API elements to determine whether they have API directive defects [8]. These data sets are several orders of magnitude smaller than the ones where Deep Learning approaches are the state of the art, such as ImageNet.

In two cases, some of these data sets have been used as input for deep learning techniques, with lackluster results. Fakhoury et al. applied a Convolutional Neural Network (CNN) for linguistic antipattern detection; they find that the CNN matches

978-1-7281-1758-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-NIER.2019.00016

ICSE'19 —NIER

Pre-trained Models for SD Activities

2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)

Leveraging Small Software Engineering Data Sets with Pre-trained Neural Networks

Romain Robbes
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

Andrea Janes
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

***Abstract*—Many software engineering data sets, particularly those that demand manual labelling for classification, are necessarily small. As a consequence, several recent software engineering papers have cast doubt on the effectiveness of deep neural networks for classification tasks, when applied to these data sets. We provide initial evidence that recent advances in Natural Language Processing, that allow neural networks to leverage large amount of unlabelled data in a pre-training phase, can significantly improve performance.**

***Index Terms*—Data Sets, Deep Learning, Transfer Learning**

I. Introduction

Deep neural networks have revolutionized the field of machine learning, leading to significant improvements over the state of the art in a large variety of tasks and domains (including image recognition, speech recognition, machine translation, summarization, and others). In Software Engineering, the results have been more mixed. While many approaches have promising results (e.g., code completion, variable naming, code translation and summarization), several recent papers have cast doubt about the effectiveness of neural networks in some settings [1], [2]. One reason is the performance on small SE data sets, which we examine in this paper.

The use cases where deep neural networks outperform the state of the art are those where large amount of data is available, such as the ImageNet data set [3], which features more than 14 million labelled images in its largest version. However, many data sets are much smaller, particularly in the case of supervised learning, as each data point must be manually labelled. This manual task is often an expensive process, and all the more in Software Engineering: while ImageNet labelling can be crowdsourced, labelling a SE data set requires significant expertise. It cannot be crowdsourced easily, and is thus vastly more expensive. In practice, many SE data sets are manually labelled by researchers, and are thus limited to hundreds or thousands of data points only. Recent work reports disappointing performance when training deep neural networks from scratch on small SE data sets.

In this paper, we investigate whether the small data set issue can be alleviated in Software Engineering, starting with natural language (NL) SE datasets (we are actively exploring the source code case). We show that a form of transfer learning, namely unsupervised pre-training on large data sets, can be effectively used to improve the performance on small NL SE data sets. Several approaches—detailed in Section III—have shown this strategy to be effective in Computer Vision and (lately) in Natural Language Processing. These approaches are all a variation of a common idea: when training a neural network from scratch, all the model's weights are initialized randomly. Pre-training instead initializes the weights with values obtained by training a model on a related tasks, for which data is easier to gather; in this case, by using large amounts of unlabelled data, no manual labelling is necessary.

II. Small Data Sets in Software Engineering

Many labelling tasks, such as identifying objects in images, drawing bounding boxes for these objects, or identifying the sentiment of a movie review, do not require specific expertise. They can hence be crowdsourced at a reasonable cost, particularly thanks to crowdsourcing platforms such as Amazon Mechanical Turk.

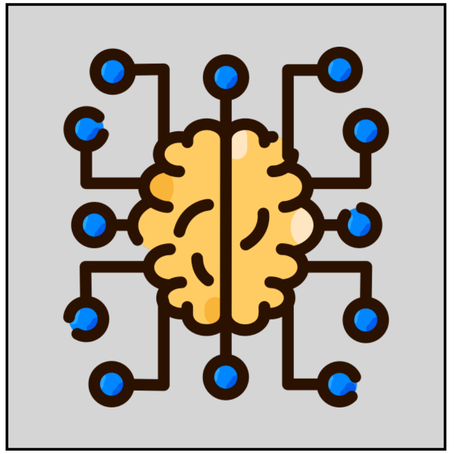
As manual labelling in Software Engineering requires expertise in Software Engineering, it is usually done by the researchers themselves. This approach is expensive, and due to the amount of effort involved, is often limited to hundreds or thousands of data points. This is all the more true since the effort is often duplicated to verify whether annotators are in agreement with each other. Some examples follow.

Bacchelli et al. manually linked 2,139 emails from 6 open-source projects with the source code entities they referenced [4]. Fakhoury et al. labelled 1,700 Java methods (with comments) according to the linguistic antipatterns they exhibited [2]. Ortu et al. labelled 2,000 JIRA issue reports, and 4,000 sentences with the sentiments expressed in them. Novielli et al. labelled 4,423 stack Overflow posts with sentiment [5]. Lin et al. labelled the sentiment of *each word* of 1,500 Stack Overflow sentences according to their sentiment [1]; they also labelled 341 app reviews for sentiment. Villaroel et al. [6] classified 1,200 app reviews, while Maalej at al. [7] classified 4,400 app reviews, in both cases in categories such as bug reports, and feature requests. Zhou et al. hired students to analyze 1,674 API elements to determine whether they have API directive defects [8]. These data sets are several orders of magnitude smaller than the ones where Deep Learning approaches are the state of the art, such as ImageNet.

In two cases, some of these data sets have been used as input for deep learning techniques, with lackluster results. Fakhoury et al. applied a Convolutional Neural Network (CNN) for linguistic antipattern detection; they find that the CNN matches

978-1-7281-1758-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-NIER.2019.00016

ICSE'19 —NIER



Pre-trained Models for SD Activities

2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)

Leveraging Small Software Engineering Data Sets with Pre-trained Neural Networks

Romain Robbes
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

Andrea Janes
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

***Abstract*—Many software engineering data sets, particularly those that demand manual labelling for classification, are necessarily small. As a consequence, several recent software engineering papers have cast doubt on the effectiveness of deep neural networks for classification tasks, when applied to these data sets. We provide initial evidence that recent advances in Natural Language Processing, that allow neural networks to leverage large amount of unlabelled data in a pre-training phase, can significantly improve performance.**

***Index Terms*—Data Sets, Deep Learning, Transfer Learning**

I. Introduction

Deep neural networks have revolutionized the field of machine learning, leading to significant improvements over the state of the art in a large variety of tasks and domains (including image recognition, speech recognition, machine translation, summarization, and others). In Software Engineering, the results have been more mixed. While many approaches have promising results (e.g., code completion, variable naming, code translation and summarization), several recent papers have cast doubt about the effectiveness of neural networks in some settings [1], [2]. One reason is the performance on small SE data sets, which we examine in this paper.

The use cases where deep neural networks outperform the state of the art are those where large amount of data is available, such as the ImageNet data set [3], which features more than 14 million labelled images in its largest version. However, many data sets are much smaller, particularly in the case of supervised learning, as each data point must be manually labelled. This manual task is often an expensive process, and all the more in Software Engineering: while ImageNet labelling can be crowdsourced, labelling a SE data set requires significant expertise. It cannot be crowdsourced easily, and is thus vastly more expensive. In practice, many SE data sets are manually labelled by researchers, and are thus limited to hundreds or thousands of data points only. Recent work reports disappointing performance when training deep neural networks from scratch on small SE data sets.

In this paper, we investigate whether the small data set issue can be alleviated in Software Engineering, starting with natural language (NL) SE datasets (we are actively exploring the source code case). We show that a form of transfer learning, namely unsupervised pre-training on large data sets, can be effectively used to improve the performance on small NL SE data sets. Several approaches—detailed in Section III—have shown this strategy to be effective in Computer Vision and (lately) in Natural Language Processing. These approaches are all a variation of a common idea: when training a neural network from scratch, all the model's weights are initialized randomly. Pre-training instead initializes the weights with values obtained by training a model on a related tasks, for which data is easier to gather; in this case, by using large amounts of unlabelled data, no manual labelling is necessary.

II. Small Data Sets in Software Engineering

Many labelling tasks, such as identifying objects in images, drawing bounding boxes for these objects, or identifying the sentiment of a movie review, do not require specific expertise. They can hence be crowdsourced at a reasonable cost, particularly thanks to crowdsourcing platforms such as Amazon Mechanical Turk.

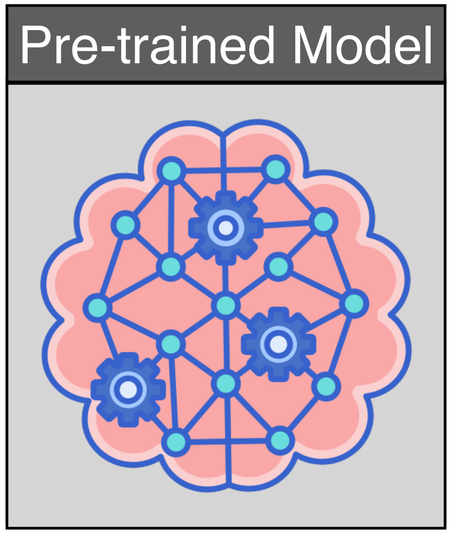
As manual labelling in Software Engineering requires expertise in Software Engineering, it is usually done by the researchers themselves. This approach is expensive, and due to the amount of effort involved, is often limited to hundreds or thousands of data points. This is all the more true since the effort is often duplicated to verify whether annotators are in agreement with each other. Some examples follow.

Bacchelli et al. manually linked 2,139 emails from 6 open-source projects with the source code entities they referenced [4]. Fakhoury et al. labelled 1,700 Java methods (with comments) according to the linguistic antipatterns they exhibited [2]. Ortu et al. labelled 2,000 JIRA issue reports, and 4,000 sentences with the sentiments expressed in them. Novielli et al. labelled 4,423 stack Overflow posts with sentiment [5]. Lin et al. labelled the sentiment of *each word* of 1,500 Stack Overflow sentences according to their sentiment [1]; they also labelled 341 app reviews for sentiment. Villaroel et al. [6] classified 1,200 app reviews, while Maalej at al. [7] classified 4,400 app reviews, in both cases in categories such as bug reports, and feature requests. Zhou et al. hired students to analyze 1,674 API elements to determine whether they have API directive defects [8]. These data sets are several orders of magnitude smaller than the ones where Deep Learning approaches are the state of the art, such as ImageNet.

In two cases, some of these data sets have been used as input for deep learning techniques, with lackluster results. Fakhoury et al. applied a Convolutional Neural Network (CNN) for linguistic antipattern detection; they find that the CNN matches

978-1-7281-1758-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-NIER.2019.00016

ICSE'19 —NIER



Pre-trained Models for SD Activities

2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)

Leveraging Small Software Engineering Data Sets with Pre-trained Neural Networks

Romain Robbes
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

Andrea Janes
Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

***Abstract*—Many software engineering data sets, particularly those that demand manual labelling for classification, are necessarily small. As a consequence, several recent software engineering papers have cast doubt on the effectiveness of deep neural networks for classification tasks, when applied to these data sets. We provide initial evidence that recent advances in Natural Language Processing, that allow neural networks to leverage large amount of unlabelled data in a pre-training phase, can significantly improve performance.**

***Index Terms*—Data Sets, Deep Learning, Transfer Learning**

I. Introduction

Deep neural networks have revolutionized the field of machine learning, leading to significant improvements over the state of the art in a large variety of tasks and domains (including image recognition, speech recognition, machine translation, summarization, and others). In Software Engineering, the results have been more mixed. While many approaches have promising results (e.g., code completion, variable naming, code translation and summarization), several recent papers have cast doubt about the effectiveness of neural networks in some settings [1], [2]. One reason is the performance on small SE data sets, which we examine in this paper.

The use cases where deep neural networks outperform the state of the art are those where large amount of data is available, such as the ImageNet data set [3], which features more than 14 million labelled images in its largest version. However, many data sets are much smaller, particularly in the case of supervised learning, as each data point must be manually labelled. This manual task is often an expensive process, and all the more in Software Engineering: while ImageNet labelling can be crowdsourced, labelling a SE data set requires significant expertise. It cannot be crowdsourced easily, and is thus vastly more expensive. In practice, many SE data sets are manually labelled by researchers, and are thus limited to hundreds or thousands of data points only. Recent work reports disappointing performance when training deep neural networks from scratch on small SE data sets.

In this paper, we investigate whether the small data set issue can be alleviated in Software Engineering, starting with natural language (NL) SE datasets (we are actively exploring the source code case). We show that a form of transfer learning, namely unsupervised pre-training on large data sets, can be effectively used to improve the performance on small NL SE data sets. Several approaches—detailed in Section III—have shown this strategy to be effective in Computer Vision and (lately) in Natural Language Processing. These approaches are all a variation of a common idea: when training a neural network from scratch, all the model's weights are initialized randomly. Pre-training instead initializes the weights with values obtained by training a model on a related tasks, for which data is easier to gather; in this case, by using large amounts of unlabelled data, no manual labelling is necessary.

II. Small Data Sets in Software Engineering

Many labelling tasks, such as identifying objects in images, drawing bounding boxes for these objects, or identifying the sentiment of a movie review, do not require specific expertise. They can hence be crowdsourced at a reasonable cost, particularly thanks to crowdsourcing platforms such as Amazon Mechanical Turk.

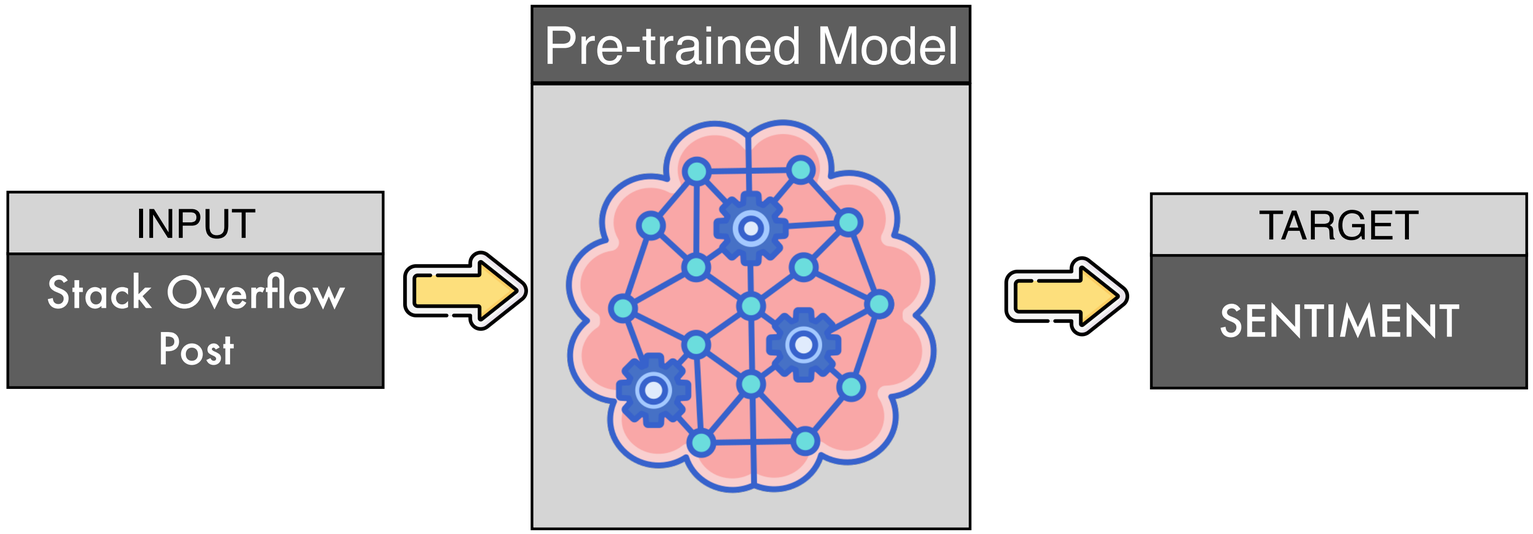
As manual labelling in Software Engineering requires expertise in Software Engineering, it is usually done by the researchers themselves. This approach is expensive, and due to the amount of effort involved, is often limited to hundreds or thousands of data points. This is all the more true since the effort is often duplicated to verify whether annotators are in agreement with each other. Some examples follow.

Bacchelli et al. manually linked 2,139 emails from 6 open-source projects with the source code entities they referenced [4]. Fakhoury et al. labelled 1,700 Java methods (with comments) according to the linguistic antipatterns they exhibited [2]. Ortu et al. labelled 2,000 JIRA issue reports, and 4,000 sentences with the sentiments expressed in them. Novielli et al. labelled 4,423 stack Overflow posts with sentiment [5]. Lin et al. labelled the sentiment of *each word* of 1,500 Stack Overflow sentences according to their sentiment [1]; they also labelled 341 app reviews for sentiment. Villaroel et al. [6] classified 1,200 app reviews, while Maalej at al. [7] classified 4,400 app reviews, in both cases in categories such as bug reports, and feature requests. Zhou et al. hired students to analyze 1,674 API elements to determine whether they have API directive defects [8]. These data sets are several orders of magnitude smaller than the ones where Deep Learning approaches are the state of the art, such as ImageNet.

In two cases, some of these data sets have been used as input for deep learning techniques, with lackluster results. Fakhoury et al. applied a Convolutional Neural Network (CNN) for linguistic antipattern detection; they find that the CNN matches

978-1-7281-1758-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-NIER.2019.00016

ICSE'19 —NIER



Pre-trained Models for SD Activities

Pre-training objectives

objectives are the specific goals or tasks that a model is designed to solve during the pre-training phase.

- MLM (**M**asking **L**anguage **M**odeling)
- NSP (**N**ext **S**entence **P**rediction)
- CLM (**C**ausal **L**anguage **M**odeling)
- PLM (**P**ermuted **L**anguage **M**odeling)

Pre-trained Models for SD Activities

Pre-training objectives

MLM (**M**asking **L**anguage **M**odeling)

The model learns the meaning and structure of words within a sentence by looking at the full context.

Pre-trained Models for SD Activities

Pre-training objectives

MLM (**M**asking **L**anguage **M**odeling)

The model learns the meaning and structure of words within a sentence by looking at the full context.

```
The cat is on the table
```

Pre-trained Models for SD Activities

Pre-training objectives

MLM (**M**asking **L**anguage **M**odeling)

The model learns the meaning and structure of words within a sentence by looking at the full context.

`The cat <MASK> on the table`

Pre-trained Models for SD Activities

Pre-training objectives

NSP (**N**ext **S**entence **P**rediction)

The model learns how to connect sentences and whether there is a logical progression between them

Pre-trained Models for SD Activities

Pre-training objectives

NSP (**N**ext **S**entence **P**rediction)

The model learns how to connect sentences and whether there is a logical progression between them

S1: `The cat is on the table`

S2: `and is asleep`

Pre-trained Models for SD Activities

Pre-training objectives

NSP (**N**ext **S**entence **P**rediction)

The model learns how to connect sentences and whether there is a logical progression between them

S1: `The cat is on the table`
S2: `and is asleep`

OUTPUT: 1

Pre-trained Models for SD Activities

Pre-training objectives

NSP (**N**ext **S**entence **P**rediction)

The model learns how to connect sentences and whether there is a logical progression between them

S1: `and is asleep`
S2: `The cat is on the table`

OUTPUT:0

Pre-trained Models for SD Activities

Pre-training objectives

CLM (**C**ausal **L**anguage **M**odeling)

The model is trained to predict the next word based only on the words preceding the current one.

Pre-trained Models for SD Activities

Pre-training objectives

CLM (**C**ausal **L**anguage **M**odeling)

The model is trained to predict the next word based only on the words preceding the current one.

Does it ring a bell?

Pre-trained Models for SD Activities

Pre-training objectives

CLM (**C**ausal **L**anguage **M**odeling)

VS

MLM (**M**asking **L**anguage **M**odeling)

Pre-trained Models for SD Activities

Pre-training objectives

PLM (**P**ermuted **L**anguage **M**odeling)

The model predicts tokens in a random, permuted order that a fixed left-to-right or even unidirectional context.

```
The cat is on the table and is asleep
```

Pre-trained Models for SD Activities

Pre-training objectives

PLM (**P**ermuted **L**anguage **M**odeling)

The model predicts tokens in a random, permuted order that a fixed left-to-right or even unidirectional context.

```
The cat is on the table and is asleep
Permuted tokens = [3, 2, 4, 1, 9, 6, 8, 7, 5]
```

Pre-trained Models for SD Activities

Pre-training objectives

PLM (**P**ermuted **L**anguage **M**odeling)

The model predicts tokens in a random, permuted order that a fixed left-to-right or even unidirectional context.

```
is cat on The asleep is and table the
Permuted tokens = [3, 2, 4, 1, 9, 6, 8, 7, 5]
```

Pre-trained Models for SD Activities

UNIDIRECTIONAL

Pre-training objectives

PLM (**P**ermuted **L**anguage **M**odeling)

The model predicts tokens in a random, permuted order that a fixed left-to-right or even unidirectional context.

```
is cat on The asleep table is and asleep the
Permuted tokens = [3, 2, 4, 1, 9, 6, 8, 7, 5]
```

Pre-trained Models for SD Activities

```
public void bubbleSort(int[] array) {
    boolean sorted = false;
    int temp;
    while (!sorted) {
        sorted = true;
        for (int i = 0; i < array.length - 1; i++) {
            if (a[i] > a[i+1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                sorted = false;
            }
        }
    }
}
```

Pre-training

Pre-trained Models for SD Activities

```
public void bubbleSort(int[] <MASK>) {
    boolean sorted = false;
    int <MASK>;
    while (!sorted) {
        <MASK> = true;
        for (int i = 0; i < <MASK>.length - 1; i++) {
            if (a[i] > a[<MASK> +1]) {
                <MASK> = a[i];
                a[<MASK>] = a[i <MASK> 1];
                a[i+1] = temp;
                <MASK> = false;
            }
        }
    <MASK>
}
```

Pre-training

Pre-trained Models for SD Activities

```java
public void bubbleSort(int[] array) {
    boolean sorted = false;
    int temp;
    while (!sorted) {
        sorted = true;
        for (int i = 0; i < array.length - 1; i++) {
            if (a[i] > a[i+1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                sorted = false;
            }
        }
    }
}
```

Pre-training

Pre-trained Models for SD Activities

```java
public void bubbleSort(int[] array) {
    boolean sorted = false;
    int temp;
    while (!sorted) {
        sorted = true;
        for (int i = 1; i < array.length - 1; i++) {
            if (a[i] > a[i+1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                sorted = false;
            }
        }
    }
}
```

BUG FIXING

Fine-tuning

Pre-trained Models for SD Activities

```java
public void bubbleSort(int[] array) {
    boolean sorted = false;
    int temp;
    while (!sorted) {
        sorted = true;
        for (int i = 0; i < array.length - 1; i++) {
            if (a[i] > a[i+1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                sorted = false;
            }
        }
    }
}
```

BUG FIXING

Fine-tuning

Pre-trained Models for SD Activities

Multi-objective Pre-training

MLM

NSP

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a re-

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal

Pre-trained Models for SD Activities

Transfer Learning

1. Pre-train—then—Fine-tune

Pre-trained Models for SD Activities

Transfer Learning

1. Pre-train—then—Fine-tune
2. Multi-task Pre-training

Pre-trained Models for SD Activities

Multi-task Pre-training in the SE domain: A case study

Using Deep Learning to Generate Complete Log Statements

Antonio Mastropaolo
antonio.mastropaolo@usi.ch
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Luca Pascarella
luca.pascarella@usi.ch
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Gabriele Bavota
gabriele.bavota@usi.ch
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

ABSTRACT

Logging is a practice widely adopted in several phases of the software lifecycle. For example, during software development log statements allow engineers to verify and debug the system by exposing fine-grained information of the running software. While the benefits of logging are undisputed, taking proper decisions about *where* to inject log statements, *what* information to log, and at which *log level* (*e.g.,* error, warning) is crucial for the logging effectiveness. In this paper, we present LANCE (**L**og st**A**teme**N**t re**C**omm**E**nder), the first approach supporting developers in all these decisions. LANCE features a Text-To-Text-Transfer-Transformer (T5) model that has been trained on 6,894,456 Java methods. LANCE takes as input a Java method and injects in it a full log statement, including a human-comprehensible logging message and properly choosing the needed log level and the statement location. Our results show that LANCE is able to (i) properly identify the location in the code where to inject the statement in 65.9% of Java methods requiring it; (ii) selecting the proper log level in 66.2% of cases; and (iii) generate

Although technically possible, logging everything (*e.g.,* every exception) is inefficient and impracticable [53]. On the one hand, a coarse-grained logging risks hiding runtime failures, missing log messages useful for diagnoses [45].

On the other hand, a fine-grained logging risks increasing the overhead of log management and analysis [26]. To optimize the quantity and quality of data generated, developers insert log statements in strategic positions, specify appropriate log levels (*e.g.,* error, debug, info), and define compact but comprehensible text messages. Nonetheless, it remains a non-trivial task for developers to decide where, what, and at which level to log [26, 46].

For these reasons, researchers have proposed techniques to support developers in deciding what parts of the system to log [45], the log level for logging statements [26, 27, 36, 46], and the structure of log messages [28]. For example, Jia *et al.* [23] proposed an approach based on association rules to place error logs after code branches such as *if* statements. Li *et al.* [25] studied the use of topic modeling for log placement at method-level. Also, two recent works

Pre-trained Models for SD Activities

Logging, is a widespread activity among developers since log statements are used to record valuable runtime information about applications.

Pre-trained Models for SD Activities

Logging, is a widespread activity among developers since log statements are used to record valuable runtime information about applications.

Log statements represent the mechanism through which we perform logging. Each logging statement features three different **log components** a log level, a log messages and the position in the underlying code snippet

Pre-trained Models for SD Activities

```
public void stopContext (){
      if ( _context != null && _publisher != null ){
            try {
                  _publisher.destroy(_context);
            }
            catch (Exception e) {
                  logger.err(“Cannot destroy the context")
            }
      }
}
```

Pre-trained Models for SD Activities

End-to-End Support for Logging Activities

Using Deep Learning to Generate Complete Log Statements

Antonio Mastropaolo
antonio.mastropaolo@usi.ch
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Luca Pascarella
luca.pascarella@usi.ch
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Gabriele Bavota
gabriele.bavota@usi.ch
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

ABSTRACT

Logging is a practice widely adopted in several phases of the software lifecycle. For example, during software development log statements allow engineers to verify and debug the system by exposing fine-grained information of the running software. While the benefits of logging are undisputed, taking proper decisions about *where* to inject log statements, *what* information to log, and at which *log level* (*e.g.,* error, warning) is crucial for the logging effectiveness. In this paper, we present LANCE (**L**og st**A**teme**N**t re**C**omm**E**nder), the first approach supporting developers in all these decisions. LANCE features a Text-To-Text-Transfer-Transformer (T5) model that has been trained on 6,894,456 Java methods. LANCE takes as input a Java method and injects in it a full log statement, including a human-comprehensible logging message and properly choosing the needed log level and the statement location. Our results show that LANCE is able to (i) properly identify the location in the code where to inject the statement in 65.9% of Java methods requiring it; (ii) selecting the proper log level in 66.2% of cases; and (iii) generate

Although technically possible, logging everything (*e.g.,* every exception) is inefficient and impracticable [53]. On the one hand, a coarse-grained logging risks hiding runtime failures, missing log messages useful for diagnoses [45].

On the other hand, a fine-grained logging risks increasing the overhead of log management and analysis [26]. To optimize the quantity and quality of data generated, developers insert log statements in strategic positions, specify appropriate log levels (*e.g.,* error, debug, info), and define compact but comprehensible text messages. Nonetheless, it remains a non-trivial task for developers to decide where, what, and at which level to log [26, 46].

For these reasons, researchers have proposed techniques to support developers in deciding what parts of the system to log [45], the log level for logging statements [26, 27, 36, 46], and the structure of log messages [28]. For example, Jia *et al.* [23] proposed an approach based on association rules to place error logs after code branches such as *if* statements. Li *et al.* [25] studied the use of topic modeling for log placement at method-level. Also, two recent works

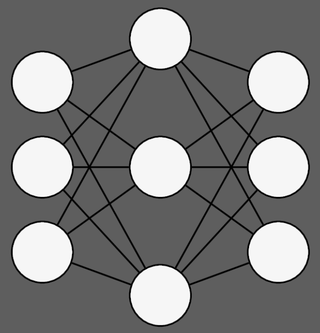
Pre-trained Models for SD Activities

INPUT

```
public ConnectionConsumer createConnectionConsumer
    (final Destination destination)
       throws JMSException{

   if(LOGGER.isTraceEnabled())
   {
       ActiveMQRALogger.LOGGER.
         trace("Create connectionConsumer");
   }
   else {
       throw new IllegalStateException(ISE);
   }
}
```

LANCE



OUTPUT

```
public ConnectionConsumer createConnectionConsumer
    (final Destination destination)
       throws JMSException{

   if(LOGGER.isTraceEnabled())
   {
       ActiveMQRALogger.LOGGER.
         trace("Create connectionConsumer");
   }
   else {
       throw new IllegalStateException(ISE);
   }
}
```


Pre-trained Models for SD Activities

Training of LANCE

1. Multiple Pre-training Tasks/Objective
2. Injection of Log Statements

Pre-trained Models for SD Activities

Input

1

```
protected boolean isBoundaryEvent(Object obj) {
    if (obj instanceof <MASK>) {
      mxCell cell = (mxCell) <MASK>;
      <MASK> cell.getId().
             startsWith("boundary-event-");
    }
    return false;
  }
```

2

```
protected Polyline(<MASK> dx,
                   double dy, Polyline next) {
      this.dx = dx;
      this.<MASK> = dy;
      <MASK>.next = next;
 }
```

...

Target

1

1) mxCell
2) obj
3) return

2

1) double
2) dy
3) this

...

~6.7M

Pre-trained Models for SD Activities

Input

1

```
@Override public void
     actionPerformed(ActionEvent ev){
     boolean ok =
          toolsDialog.showDialog(config, true);
     if(!ok){
          return;
     }
}
```

2

```
public void stopContext (){
     if ( _context != null
         && _publisher != null ){
           try {
              _publisher.destroy(_context);
           }
           catch (Exception e) {}
     }
}
```

...

Target

1

```
@Override public void
     actionPerformed(ActionEvent ev){
     <LOG_STMT>
     boolean ok =
          toolsDialog.showDialog(config, true);
     if(!ok){
          return;
     }
}
```

2

```
public void stopContext (){
     if ( _context != null
         && _publisher != null ){
           try {
              _publisher.destroy(_context);
           }
           catch (Exception e) { <LOG_STMT> }
     }
}
```

...

~133K

Pre-trained Models for SD Activities

Input

1

Masking

```
public LocalDateTimeToStringConverter
                    <MASK>() {
    return new lDTimeToStringConverter <MASK>
}
```

Log-Pos

```
private void infolog(String msg){
     processingLog += msg + ".\n";
}
```

2

Masking

```
public String getTableName() <MASK>
    return <MASK>;
}
```

Log-Pos

```
public void stopCounting(){
    runCounting = false;
    stopLogReport();
}
```

...

Target

1

Masking

```
1) lDTimeToStringConverter

2) ();
```

Log-Pos

```
private void infolog(String msg){
    <LOG_STMT>
    processingLog += msg + ".\n";
}
```

2

Masking

```
1) {

2) tableName
```

Log-Pos

```
public void stopCounting(){
     runCounting = false;
     <LOG_STMT>
     stopLogReport();
}
```

...

~6.8M

Pre-trained Models for SD Activities

Fine-tuning

Input

1

```
public ConnectionConsumer createConnectionConsumer(final
Destination destination) throws JMSException {
   if (ActiveMQRALogger.LOGGER.isTraceEnabled()) {
      ActiveMQRALogger.LOGGER.trace(
         “Create connectionConsumer”);
   else {
      logger.info("Throwing Exception”)
      throw new IllegalStateException(ISE);
   }
}
```

2

```
public ConnectionConsumer createConnectionConsumer(final
Destination destination) throws JMSException {
   if (ActiveMQRALogger.LOGGER.isTraceEnabled()) {
      ActiveMQRALogger.LOGGER.trace(
         “Create connectionConsumer”);
   else {
      logger.info("Throwing Exception”)
      throw new IllegalStateException(ISE);
   }
}
```

...

Target

1

```
public ConnectionConsumer createConnectionConsumer(final
Destination destination) throws JMSException {
   if (ActiveMQRALogger.LOGGER.isTraceEnabled()) {
      ActiveMQRALogger.LOGGER.trace(
         “Create connectionConsumer”);
   else {
      logger.info("Throwing Exception”)
      throw new IllegalStateException(ISE);
   }
}
```

2

```
public ConnectionConsumer createConnectionConsumer(final
Destination destination) throws JMSException {
   if (ActiveMQRALogger.LOGGER.isTraceEnabled()) {
      ActiveMQRALogger.LOGGER.trace(
         “Create connectionConsumer”);
   else {
      logger.info("Throwing Exception”)
      throw new IllegalStateException(ISE);
   }
}
```

...

~100K

Pre-trained Models for SD Activities

```
public HandlerResult handle(ProcessState state, ProcessInstance process) {
        Secret secret = (Secret) state.getResource();
        String secretValue = secret.getValue();
        if (StringUtils.isNotBlank(secretValue)) {
            try {
                secretsService.delete(secret.getAccountId(), secret.getValue());
            } catch (IOException e) {
                log.error("Error deleting secret {}: {}", secret.getId(), e.getMessage());
                log.error("Failed to delete secret from storage [{}]", secret.getId(), e);
                [...]
            }
        }
        return null;
}
```

LANCE log

Target log

Pre-trained Models for SD Activities

```
private synchronized CswSubscription deleteCswSubscription (String subscriptionId) throws CswException {

        [...]
        try {
            ServiceRegistration sr = registeredSubscriptions.remove(subscriptionId);
            [...]

        } catch (Exception e) {

            LOGGER.debug("Exception trying to delete subscription's configuration for subscription ID {}",
                         logSanitizedId, e);
```

LANCE log

```
            LOGGER.debug("Could not delete subscription for {}", logSanitizedId, e);
```

Target log

```
        }
}
```

Pre-trained Models for SD Activities

Transfer Learning

1. Pre-train—then—Fine-tune
2. Multi-task Pre-training

Pre-trained Models for SD Activities

Transfer Learning

1. Pre-train—then—Fine-tune
2. Multi-task Pre-training
3. Chaining Tasks

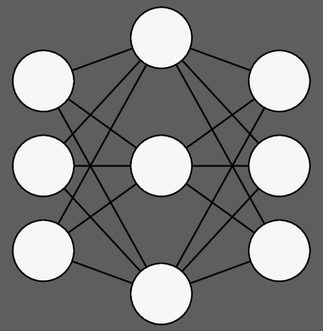
Pre-trained Models for SD Activities

INPUT

```
int main () {
    char username[8];
    int allow = 0;
    printf("Enter your username, please: ");
    gets(username);
    …
    return 0;
}
```

Vulnerable Code

Pre-trained Model



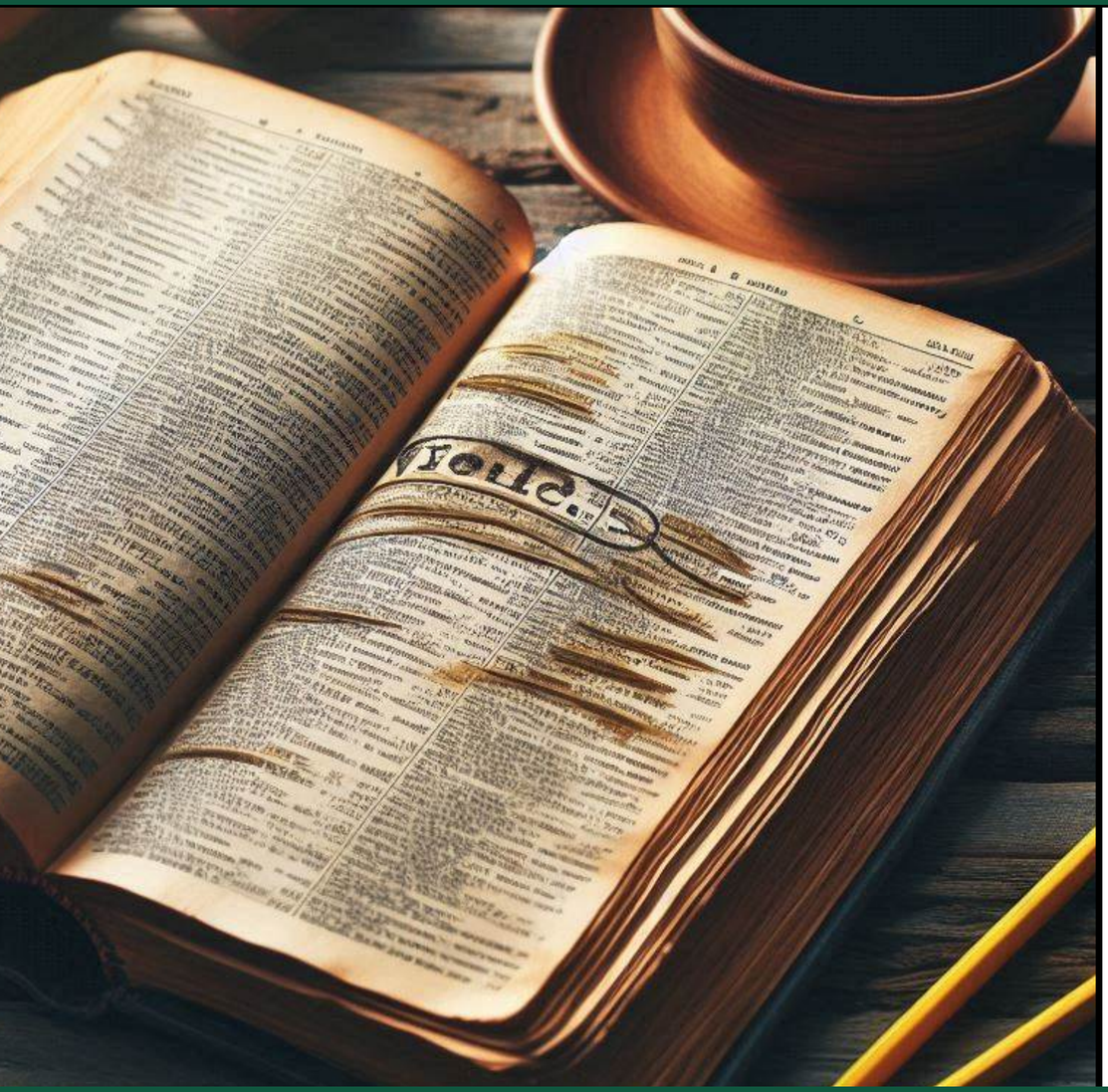
OUTPUT

```
int main () {
   char* username, *nlptr;
   int allow = 0;
   username = malloc(8 * sizeof(*username));
   if (!username)
       return EXIT_FAILURE;
   printf("Enter your username, please: ");
   fgets(username,LENGTH, stdin);
   ...
   return 0;
}
```

Patched Code

DL-based Vulnerability Patching

Pre-trained Models for SD Activities



Data Scarcity

Pre-trained Models for SD Activities

Bugs are a "superclass" of vulnerabilities

We have plenty of data in the form of bug-fixing commits

Pre-trained Models for SD Activities

Bugs are a "superclass" of vulnerabilities

We have plenty of data in the form of bug-fixing commits

SUPERVISED PRE-TRAINING

Pre-trained Models for SD Activities

Input

1

```
int main () {
    char username[8];
    int allow = 0;
    printf("Enter your username, please: ");
    gets(username);
    …
    return 0;
}
```

2

```
int main(int argc, char *argv[]) {
   char buffer[10];
   strcpy(buffer, argv[1]);
   printf(“Buffer contents: %s\n", buffer);
   return 0;
}
```

…

Target

1

```
int main () {
   char* username, *nlptr;
   int allow = 0;
   username = malloc(8 * sizeof(*username));
   fgets(username,LENGTH, stdin);
   ...
   return 0;
}
```

2

```
int main(int argc, char *argv[]) {
   char buffer[10];
   strncpy(buffer, argv[1], sizeof(buffer) - 1);
   buffer[sizeof(buffer) - 1] = '\0';
   printf(“Buffer contents: %s\n", buffer);
   return 0;
}
```

…

~100K

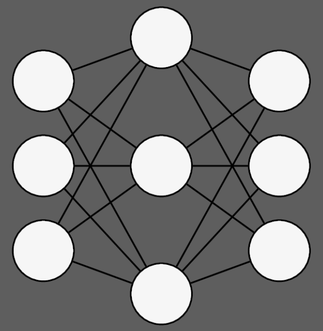
Pre-trained Models for SD Activities

INPUT

```
int main () {
    char username[8];
    int allow = 0;
    printf("Enter your username, please: ");
    gets(username);
    …
    return 0;
}
```

Vulnerable Code

Pre-trained Model



OUTPUT

```
int main () {
   char* username, *nlptr;
   int allow = 0;
   username = malloc(8 * sizeof(*username));
   if (!username)
       return EXIT_FAILURE;
   printf("Enter your username, please: ");
   fgets(username,LENGTH, stdin);
   ...
   return 0;
}
```

Patched Code

DL-based Vulnerability Patching

Pre-trained Models for SD Activities

Transfer Learning

1. Pre-train—then—Fine-tune
2. Multi-task Pre-training
3. Chaining Tasks

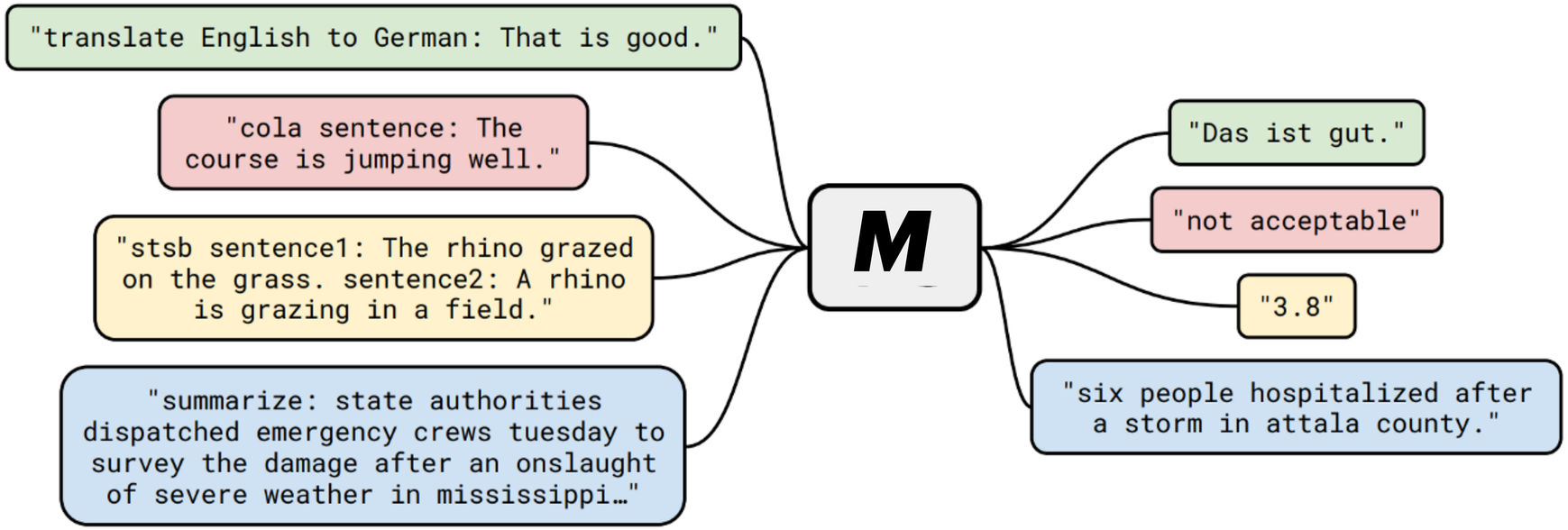
Pre-trained Models for SD Activities

Transfer Learning

1. Pre-train—then—Fine-tune
2. Multi-task Pre-training
3. Chaining Tasks
4. Multi-task Fine-tuning

Pre-trained Models for SD Activities

Multi-task Fine-tuning



Pre-trained Models for SD Activities

Multi-task Fine-tuning

Using Transfer Learning for Code-Related Tasks

Antonio Mastropaolo, Nathan Cooper, David Nader Palacio, *Student Member, IEEE*, Simone Scalabrino, Denys Poshyvanyk, *Senior Member, IEEE*, Rocco Oliveto, and Gabriele Bavota

Abstract—Deep learning (DL) techniques have been used to support several code-related tasks such as code summarization and bug-fixing. In particular, pre-trained transformer models are on the rise, also thanks to the excellent results they achieved in Natural Language Processing (NLP) tasks. The basic idea behind these models is to first pre-train them on a generic dataset using a self-supervised task (e.g., filling masked words in sentences). Then, these models are fine-tuned to support specific tasks of interest (e.g., language translation). A single model can be fine-tuned to support multiple tasks, possibly exploiting the benefits of *transfer learning*. This means that knowledge acquired to solve a specific task (e.g., language translation) can be useful to boost performance on another task (e.g., sentiment classification). While the benefits of transfer learning have been widely studied in NLP, limited empirical evidence is available when it comes to code-related tasks. In this paper, we assess the performance of the Text-To-Text Transfer Transformer (T5) model in supporting four different code-related tasks: (i) automatic bug-fixing, (ii) injection of code mutants, (iii) generation of assert statements, and (iv) code summarization. We pay particular attention in studying the role played by pre-training and multi-task fine-tuning on the model's performance. We show that (i) the T5 can achieve better performance as compared to state-of-the-art baselines; and (ii) while pre-training helps the model, not all tasks benefit from a multi-task fine-tuning.

Index Terms—Deep learning, empirical software engineering

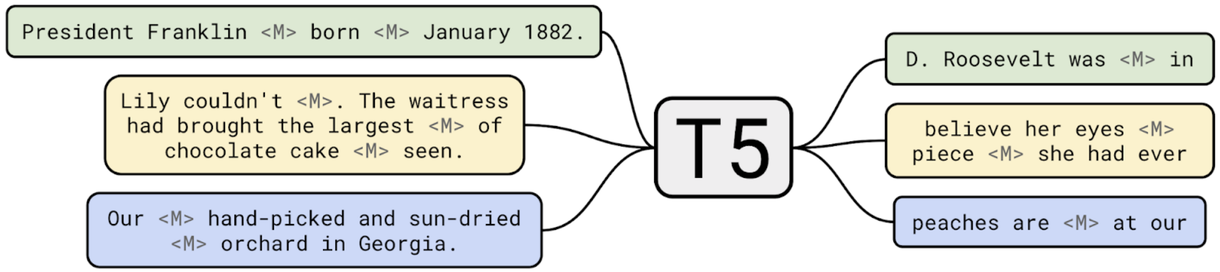
1 Introduction

Several code-related tasks have been recently automated by researchers exploiting Deep Learning (DL) techniques [81]. Several of these works customize DL models proposed in the Natural Language Processing (NLP) field to support code-related tasks, and most of them share one common characteristic: *They shape the problem at hand as a text-to-text transformation, in which the input and the output of the model are text strings*. For instance, Tufano *et al.* [78] used automating bug fixing [15], [25], [48], [75], learning generic code changes [73], supporting code migration [52], [53], code summarization [24], [32], [39], [42], code reviews [77], [78], pseudo-code generation [55], code deobfuscation [31], [79], injection of code mutants [76], generation of assert statements [82], clone detection [74], [83], traceability [49] and code completion [5], [11], [17], [17], [34], [35], [70], [84].

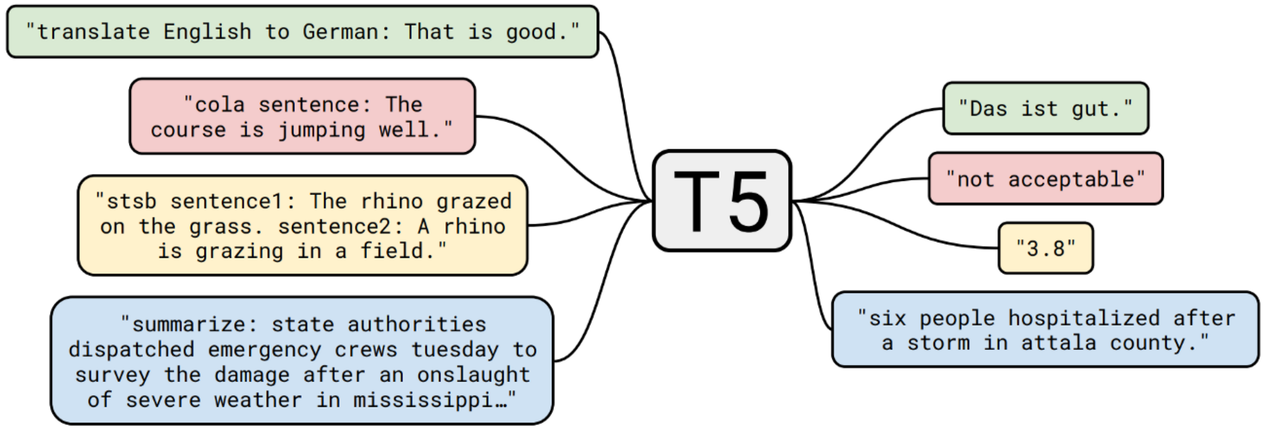
Recent years have seen the rise of *transfer learning* in the field of natural language processing. The basic idea is to first

Pre-trained Models for SD Activities

Multi-task Fine-tuning

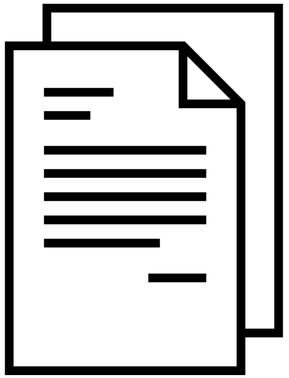


Text as Input and
Text as Output



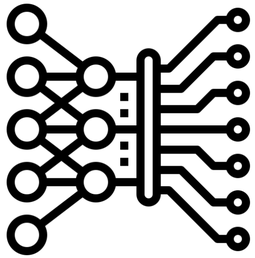
Pre-trained Models for SD Activities

1



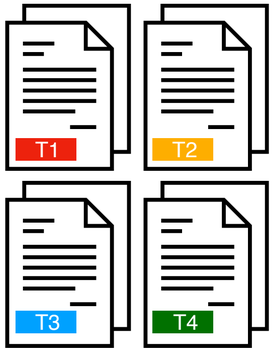
Pre-training dataset

2



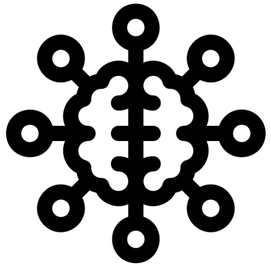
T5 Pre-training

3



Text-to-text task specific dataset

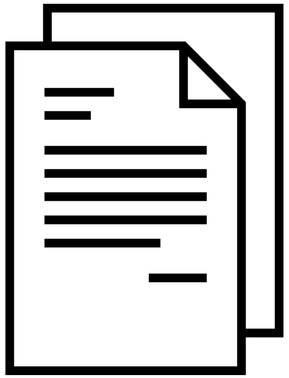
4



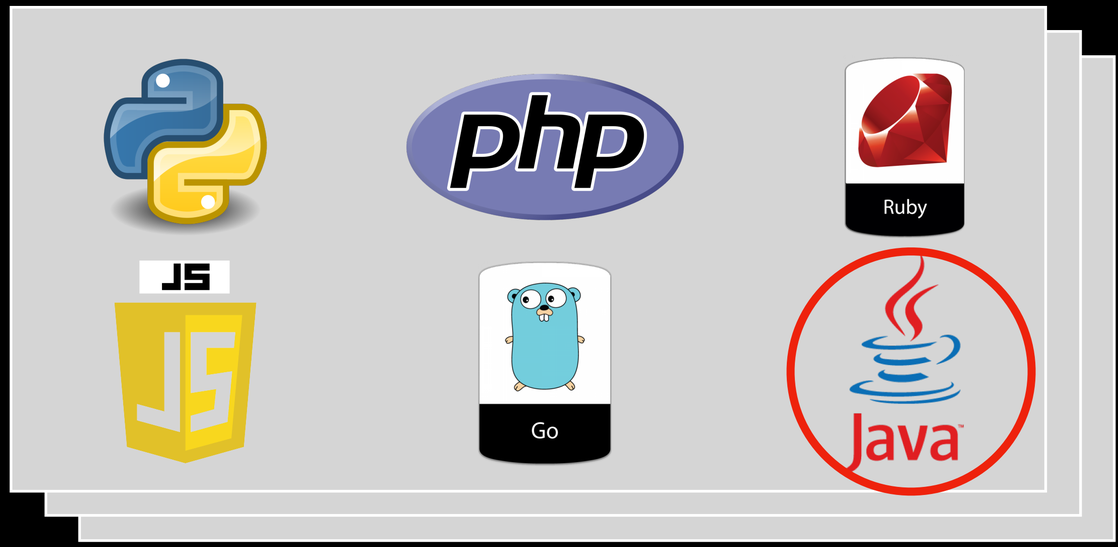
T5 Fine-tuning on tasks-mixture

Pre-trained Models for SD Activities

1



Pre-training dataset

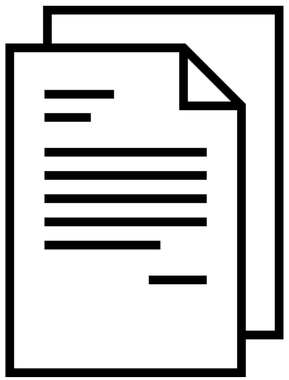


CodeSearchNet

Husain et al.

Pre-trained Models for SD Activities

1



Pre-training dataset

Creates an UnicastProcessor with the given internal buffer capacity hint

```
@CheckReturnValue @NonNull
public static <T> UnicastProcessor <T>
create (int capacityHint ){
         return new UnicastProcessor
              <T>(capacityHint);
}
```

Creates a text encryptor that uses stronger password – based encryption. Encrypted text is hex – encoded

```
public static TextEncryptor
 delux(CharSequence password,
       CharSequence salt){
       return new HexEncodingTextEncryptor
       (stronger(password, salt));
}
```

Creates a connection to the consumer

```
public ConnectionConsumer createConnectionConsumer
(final Destination destination) throws JMSException
   if (ActiveMQRALogger.LOGGER.isTraceEnabled()) {
      ActiveMQRALogger.LOGGER.trace(
         "Create connectionConsumer");
   else {
      logger.info("Throwing Exception")
      throw new IllegalStateException(ISE);
   }
}
```

Assembles the Distinguished Name that should be used the given username

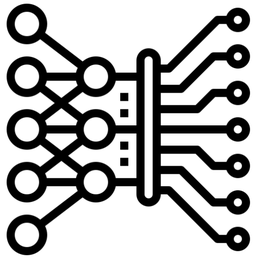
```
public DistinguishedName buildDn(
                          String username){
    DistinguishedName dn = new
          DistinguishedName (userDnBase);
          dn.add(usernameAttribute, username);
          return dn;
}
```

Pre-trained Models for SD Activities

1

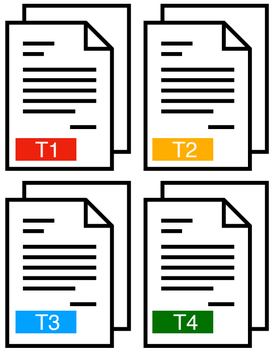
Pre-training dataset

2



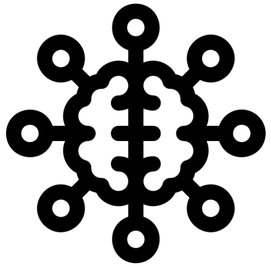
T5 Pre-training

3



Text-to-text task specific dataset

4



T5 Fine-tuning on tasks-mixture

Pre-trained Models for SD Activities

Input

1	`protected boolean isBoundaryEvent(Object obj) {`
	`  if (obj instanceof <MASK>) {`
	`  mxCell cell = (mxCell) <MASK>;`
	`  <MASK> cell.getId().`
	`  startsWith("boundary-event-");`
	`  }`
	`  return false;`
	`  }`

2	***Creates an <MASK> with the given internal <MASK> capacity hint.***

...

Target

1	1) mxCell 2) obj 3) return
2	1) UnicastProcessor 2) buffer

...

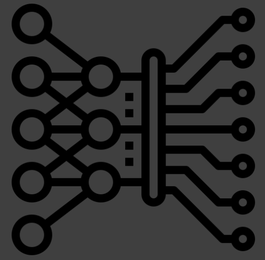
~2.6M

Pre-trained Models for SD Activities

1

Pre-training dataset

2

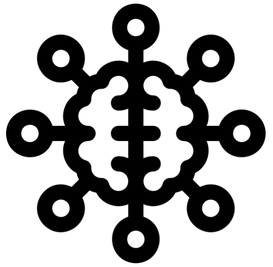


T5 Pre-training

3

Text-to-text task specific dataset

4



T5 Fine-tuning on tasks-mixture

Pre-trained Models for SD Activities

3

Text-to-text task specific dataset

Code-Related Tasks Selection

- *Automatic Bug-Fixing*
- *Code Summarization*
- *Injection of Code Mutants*
- *Assert Statements Generation*

Pre-trained Models for SD Activities

3

Text-to-text task specific dataset

An empirical study on learning bug-fixes patches in the wild via Neural Machine Translation

TOSEM, —Tufano *et al.*—

T1

Improved automatic summarisation of code routines via Attention to File Context

MSR, —Haque *et al.*—

T2

Learn how to mutate source code from bug-fixes

ICSME, —Tufano *et al.*—

T3

On learning meaningful assert statements for unit test cases

ICSE, —Watson *et al.*—

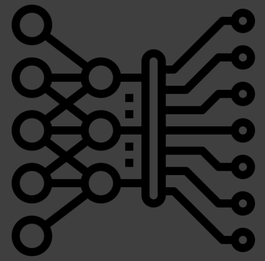
T4

Pre-trained Models for SD Activities

1

Pre-training dataset

2

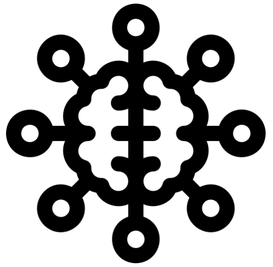


T5 Pre-training

3

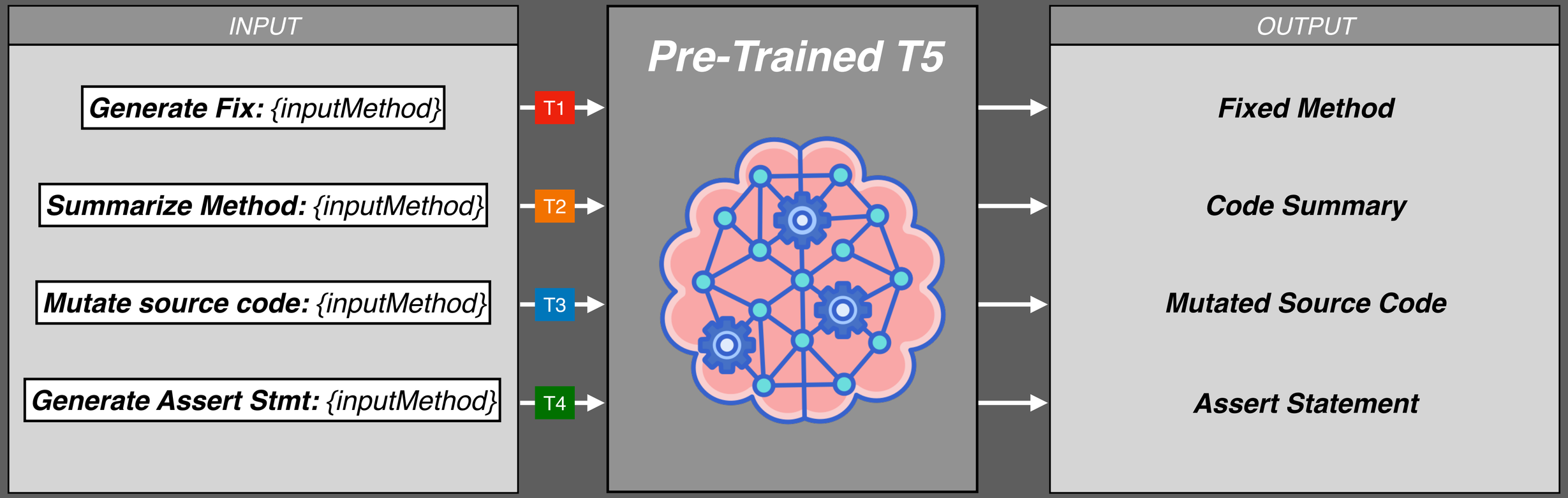
Text-to-text task specific dataset

4

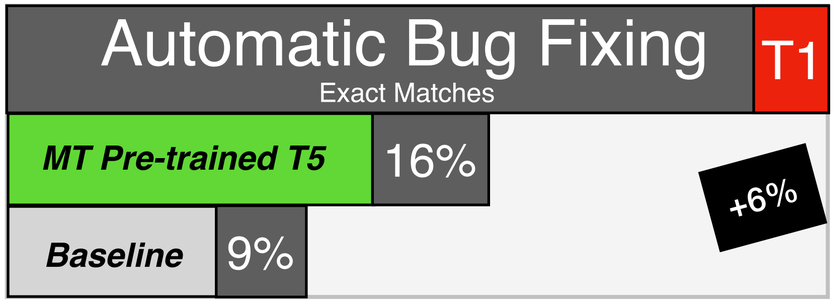


T5 Fine-tuning on tasks-mixture

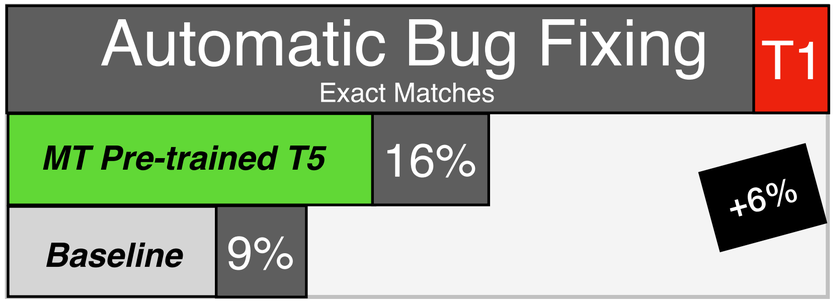
Pre-trained Models for SD Activities

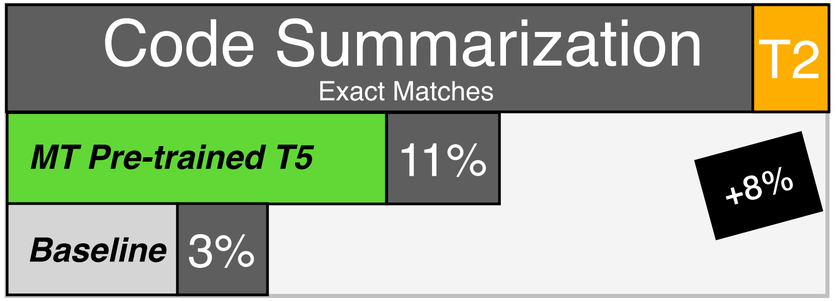


Pre-trained Models for SD Activities

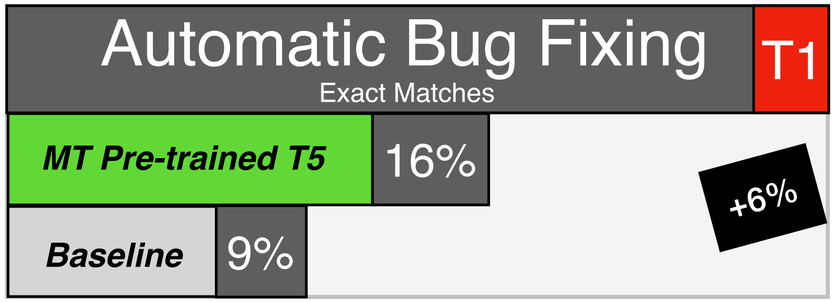


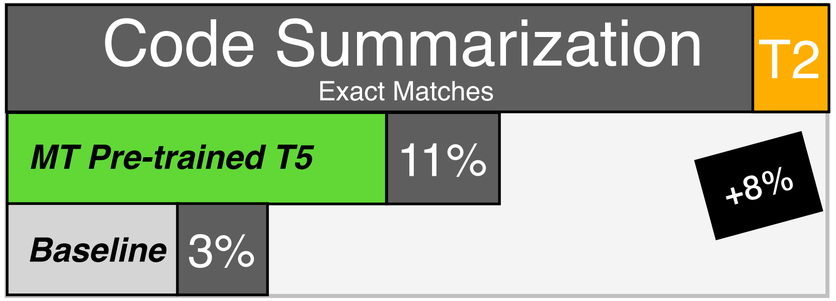
Pre-trained Models for SD Activities

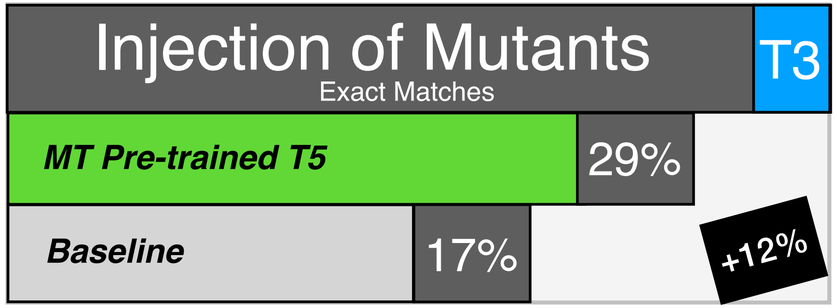




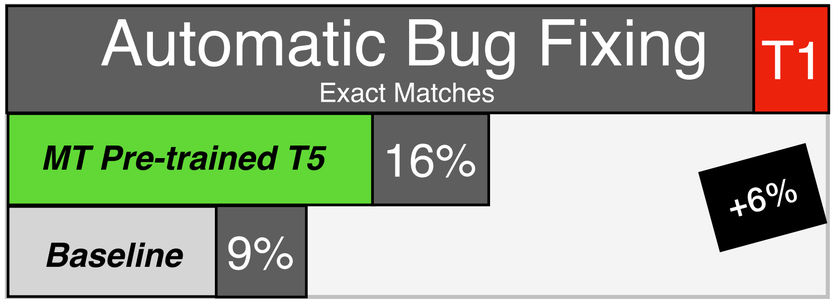
Pre-trained Models for SD Activities

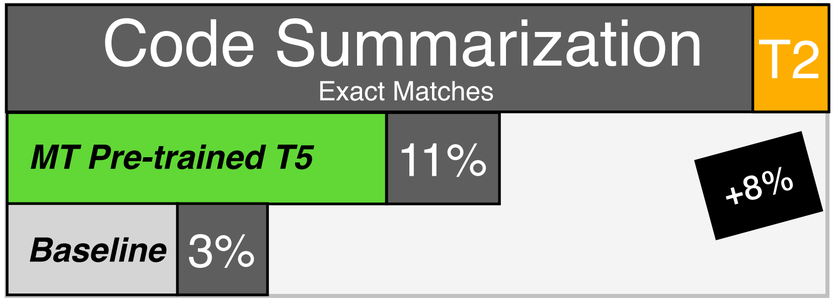


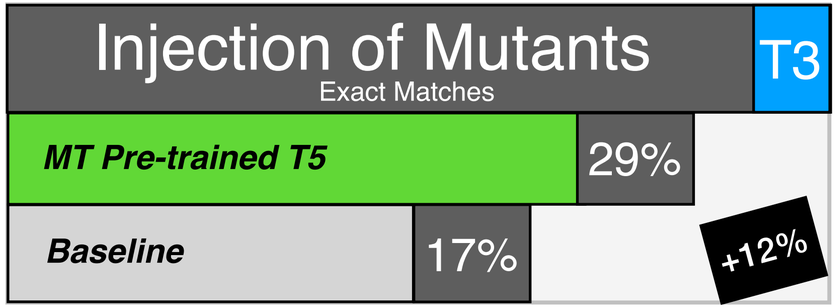


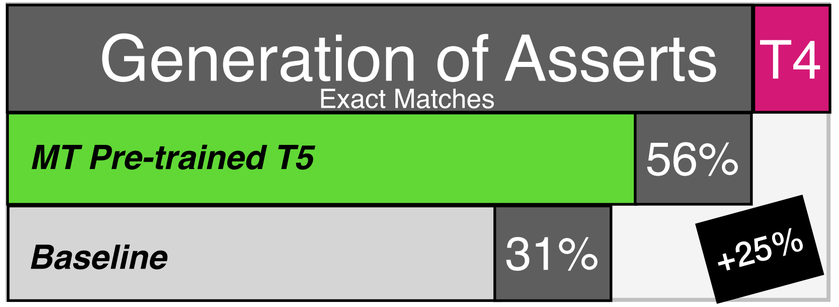


Pre-trained Models for SD Activities









Pre-trained Models for SD Activities

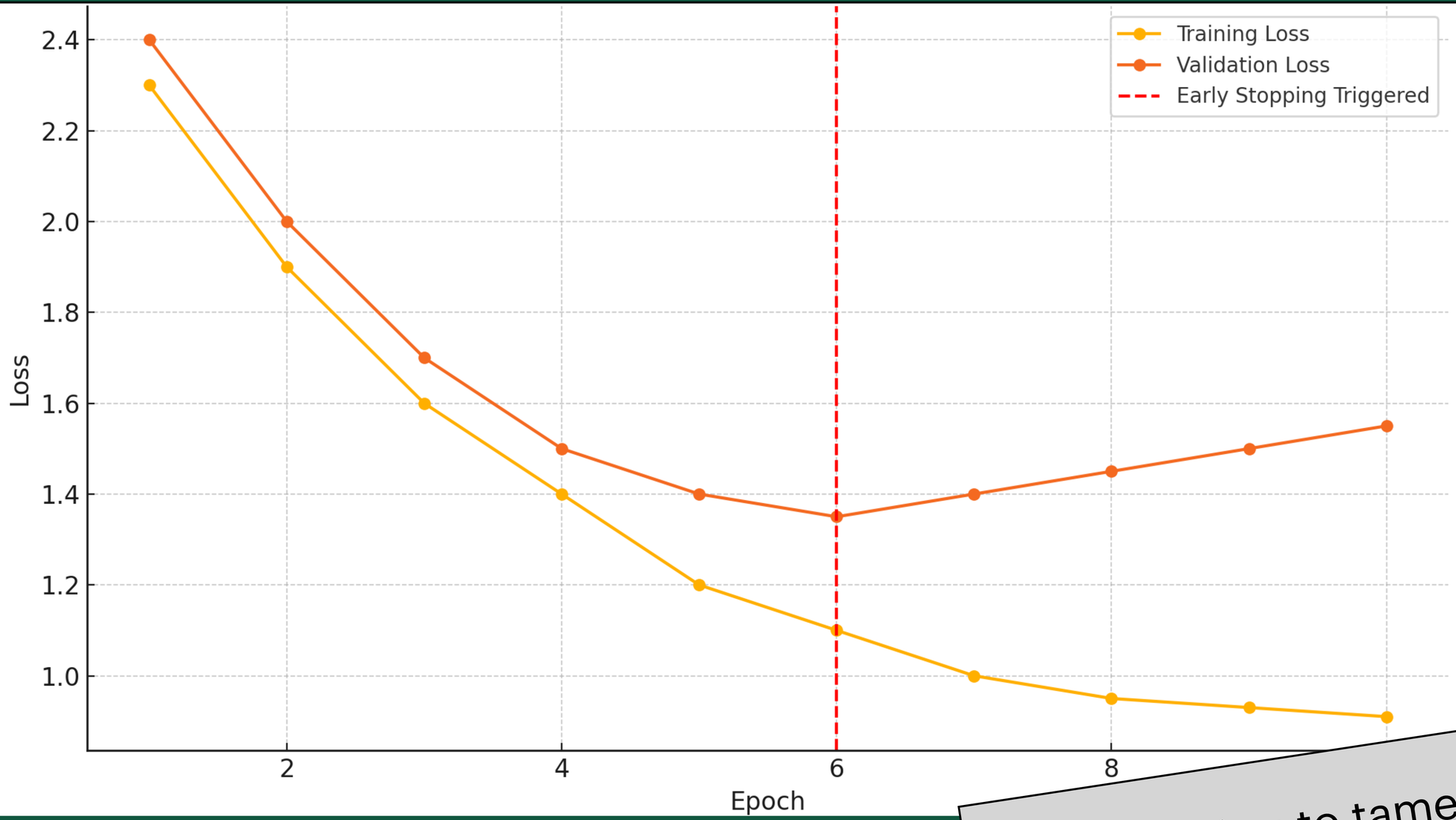
Still alive guys?

Pre-trained Models for SD Activities

Selecting The Best-Performing Model

For how many epochs we let the model (un)learn?

Pre-trained Models for SD Activities



Early Stopping to tame overfitting