

CEN 5016: Software Engineering

Fall 2024



University of
Central Florida

Dr. Kevin Moran

Week 10 - Class 1: Introduction to Software Engineering Research





- *Industry Speaker Next Week!*
- *Exams will be graded by Tuesday*
- *Research Paper Presentation Selection*
 - *Due by end of next week - more details Monday*

Software Engineering

Software Engineering

The methods and techniques by which developers design, create, test, and manage software

Software Engineering

The methods and techniques by which developers design, create, test, and manage software

Research Goal: Design tailored *automated approaches* to help facilitate *developer needs* throughout the software development and maintenance lifecycle.

PRACTICAL SIGNIFICANCE

PRACTICAL SIGNIFICANCE

Blend *scientific discovery* with *practical significance*

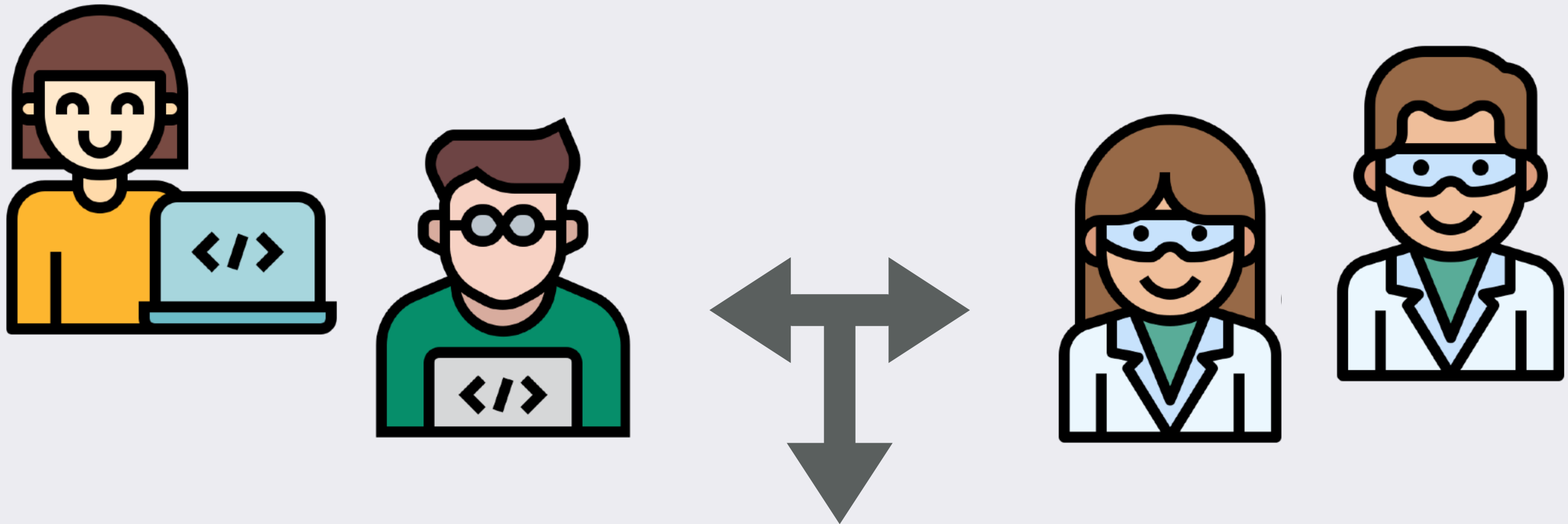
PRACTICAL SIGNIFICANCE

Blend *scientific discovery* with *practical significance*



How Can We Design Practical Automation?

UNDERSTANDING DEVELOPER NEEDS



MINING SOFTWARE REPOSITORIES



MINING SOFTWARE REPOSITORIES

MINING SOFTWARE REPOSITORIES



Source Code
Files



Software
Documentation



Screenshots



Screen
Recordings

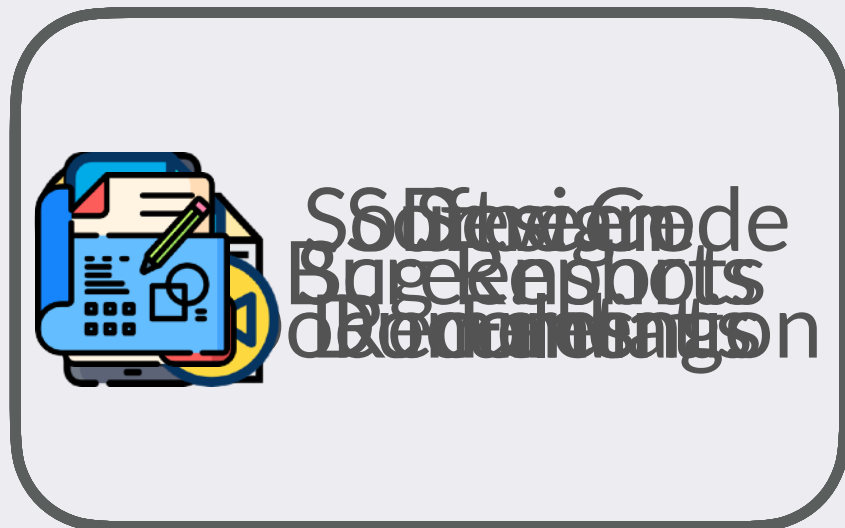


Bug Reports



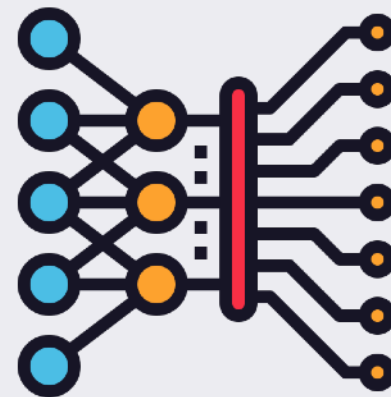
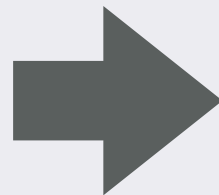
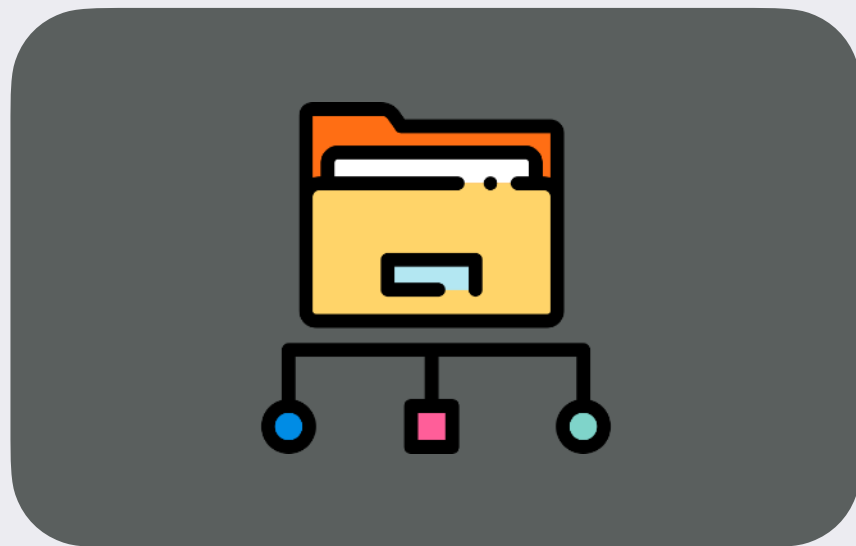
Design
Documents

LEARNING PATTERNS FROM SOFTWARE DATA

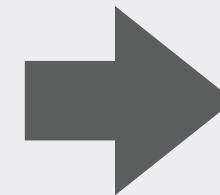


LEARNING PATTERNS FROM SOFTWARE DATA

Software
Repository Data



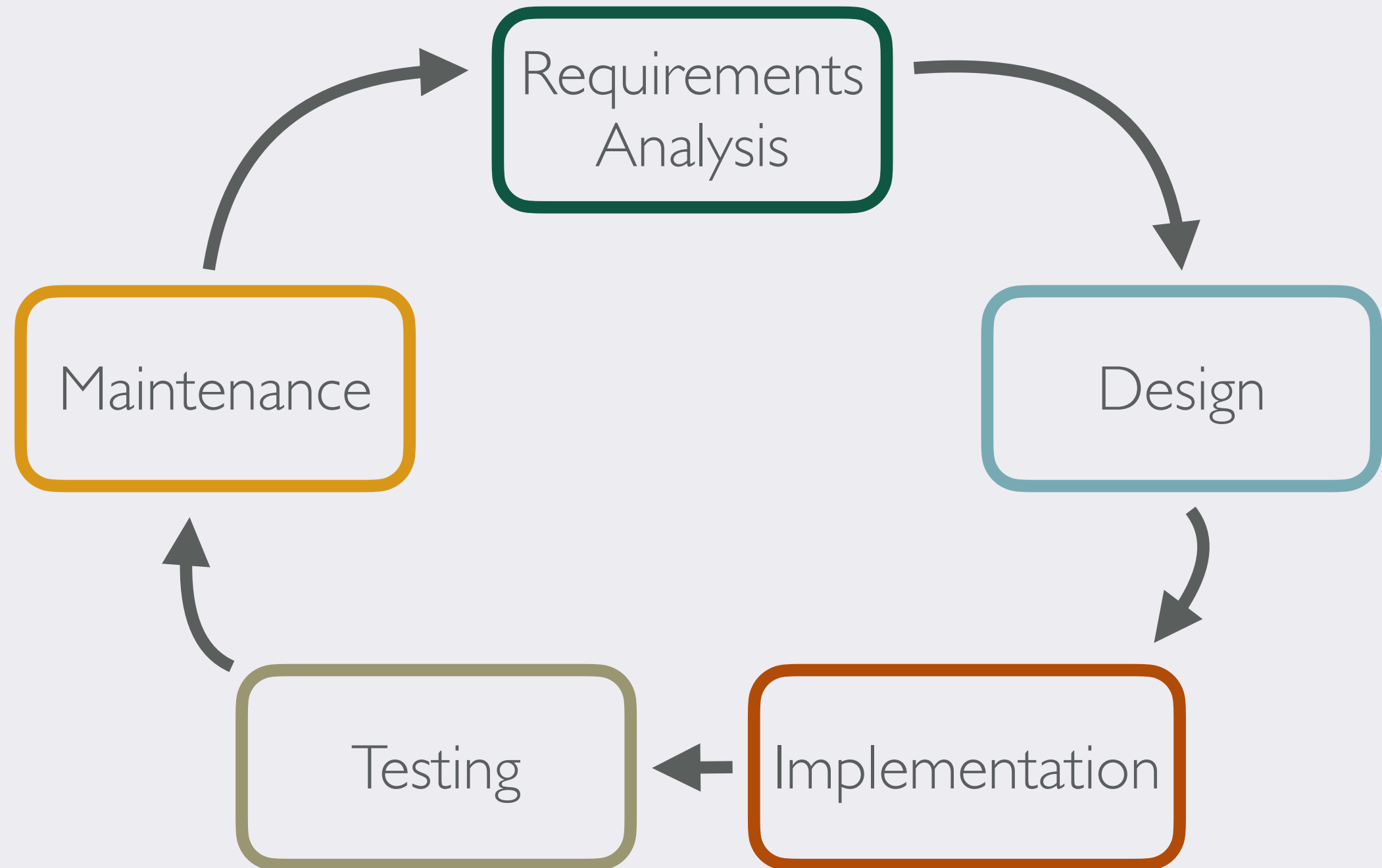
Machine
Learning



Salient
Patterns

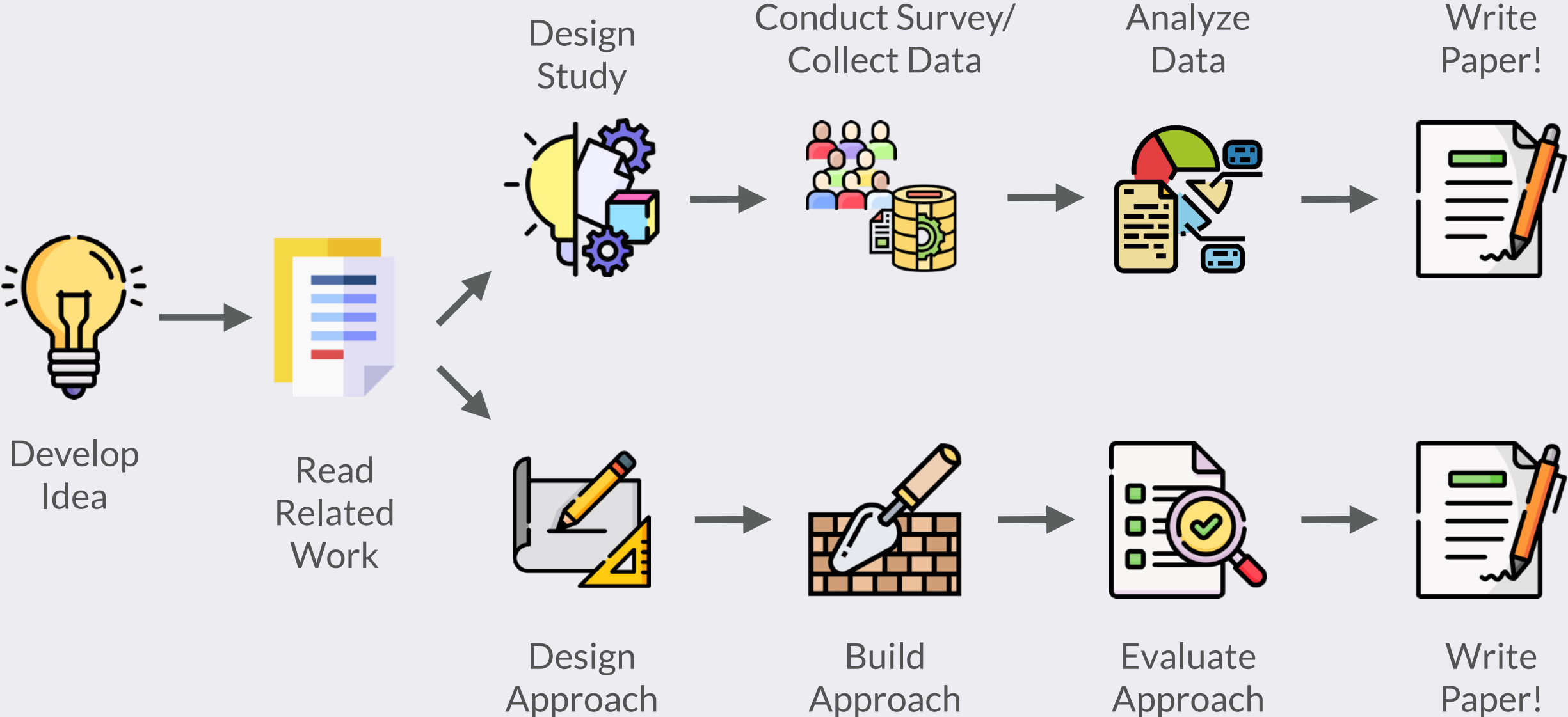


SOFTWARE DEVELOPMENT LIFECYCLE



HOW SOFTWARE ENGINEERING RESEARCH WORKS

SE RESEARCH PROJECT ROADMAP



TYPICAL SE RESEARCH TOPICS

AI and software engineering:

- Search-based software engineering
- Machine learning with and for SE
- Recommender systems
- Autonomic systems and self adaptation
- Program synthesis
- Program repair

Testing and analysis:

- Software testing
- Program analysis
- Debugging and Fault localization
- Programming languages
- Performance
- Mobile applications

Software analytics:

- Mining software repositories
- Apps and app store analysis
- Software ecosystems
- Configuration management
- Software visualization

Dependability:

- Formal methods
- Validation and Verification
- Reliability and Safety
- Privacy and Security
- Embedded and cyber-physical systems

Software evolution:

- Evolution and maintenance
- API design and evolution
- Release engineering and DevOps
- Software reuse
- Refactoring
- Program comprehension
- Reverse engineering

Social aspects of software engineering:

- Human aspects of software engineering
- Human-computer interaction
- Distributed and collaborative software engineering
- Agile methods and software processes
- Software economics
- Crowd-based software engineering
- Ethics in software engineering
- Green and sustainable technologies

Requirements, modeling, and design:

- Requirements Engineering
- Privacy and Security by Design
- Modeling and Model-Driven Engineering
- Software Architecture and Design
- Variability and product lines
- Software services

SE RESEARCH VENUES

Conferences

International Conference on
Software Engineering (ICSE)

Symposium on the Foundations
of Software Engineering (FSE)

International Conference on
Automated Software Engineering (ASE)

International Conference on Software
Maintenance & Evolution (ICSME)

International Conference on Mining
Software Repositories (MSR)

International Symposium on Software
Testing and Analysis (ISSTA)

Journals

IEEE Transactions on
Software Engineering

ACM Transactions on
Software Engineering
& Methodology

Springer Empirical
Software Engineering

Deep Learning & Software Engineering
-
A Retrospective and New Directions

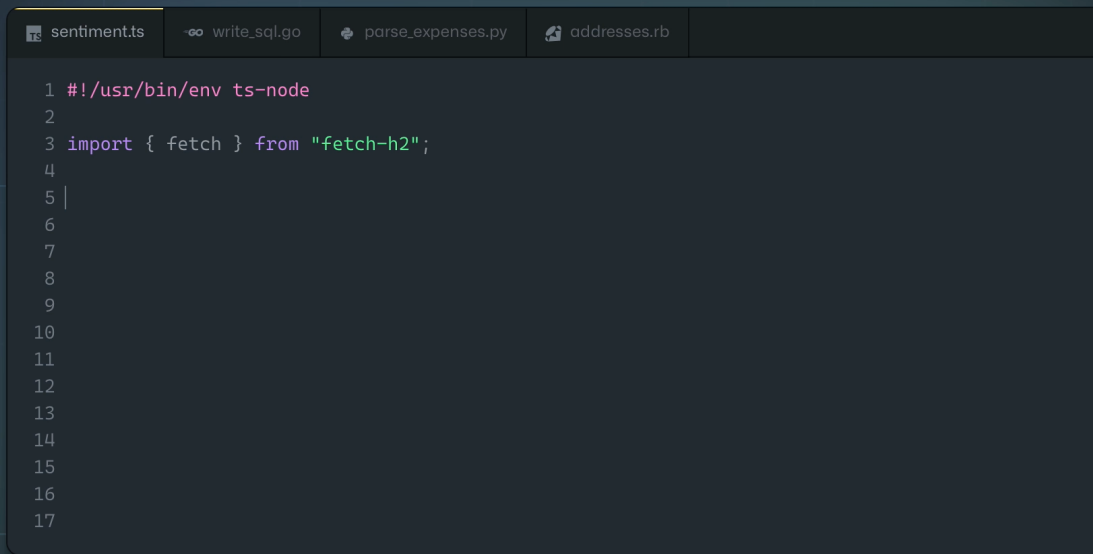
Kevin Moran, Ph.D.
Assistant Professor, CS
Director of the SAGE Research Lab
University of Central Florida



Technical Preview

Your AI pair programmer

With GitHub Copilot, get suggestions for whole lines or entire functions right inside your editor.

[Sign up >](#)

```
sentiment.ts write_sql.go parse_expenses.py addresses.rb
1  #!/usr/bin/env ts-node
2
3  import { fetch } from "fetch-h2";
4
5  |
6
7
8
9
10
11
12
13
14
15
16
17
```

Talk Outline

- **Topic 1 - Background:** The Evolution of Machine Learning (ML) to Deep Learning (DL)
- **Topic 2- DL4SE:** The Current State of Research
- **Topic 3 – Looking Forward:** Future Directions and Paths Forward

Topic 1 – Background: The Evolution of Machine Learning to Deep Learning

What is Machine Learning?

A branch of *Artificial Intelligence* that allows computers to *infer patterns* from data, which can be used for the *prediction* of new data points

The Hierarchy of Artificial Intelligence

Artificial Intelligence

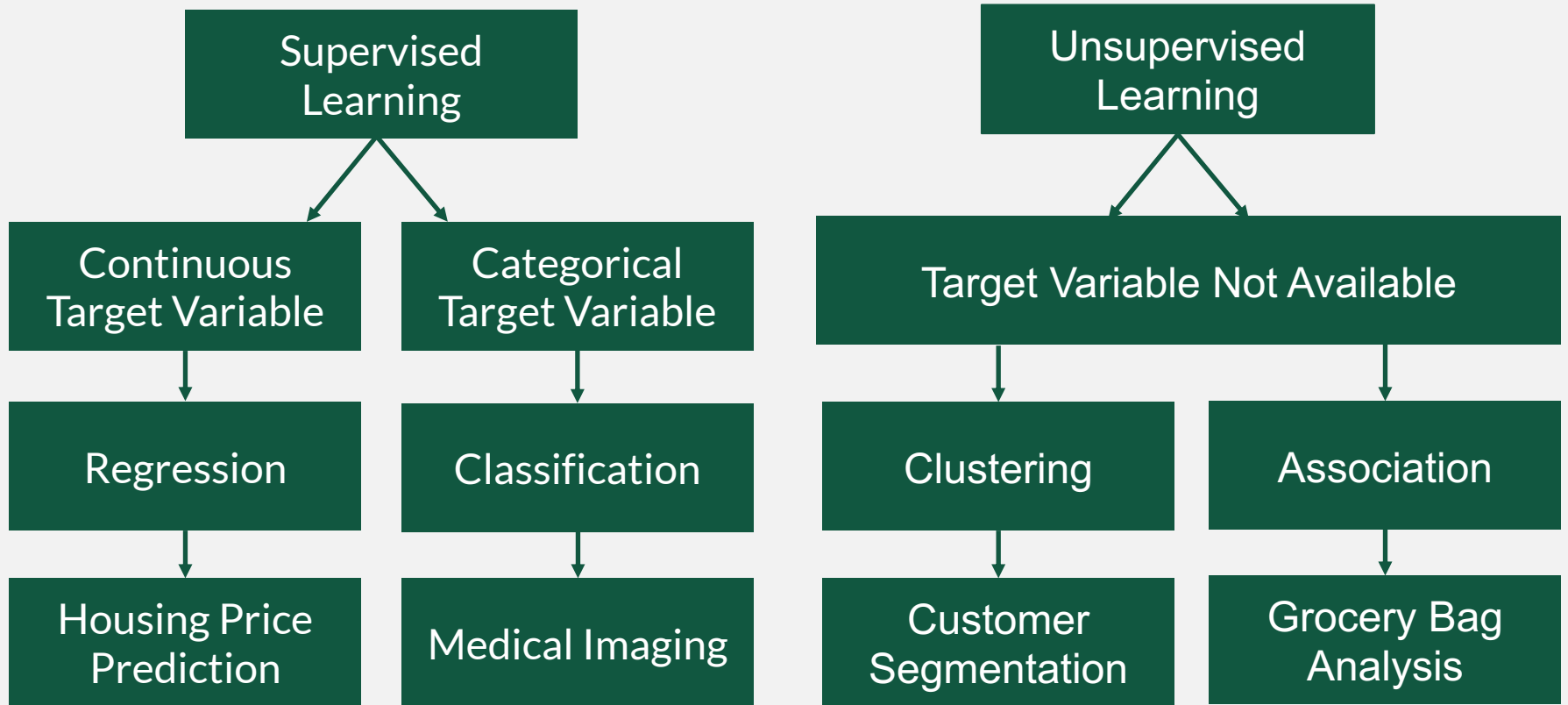


Machine Learning

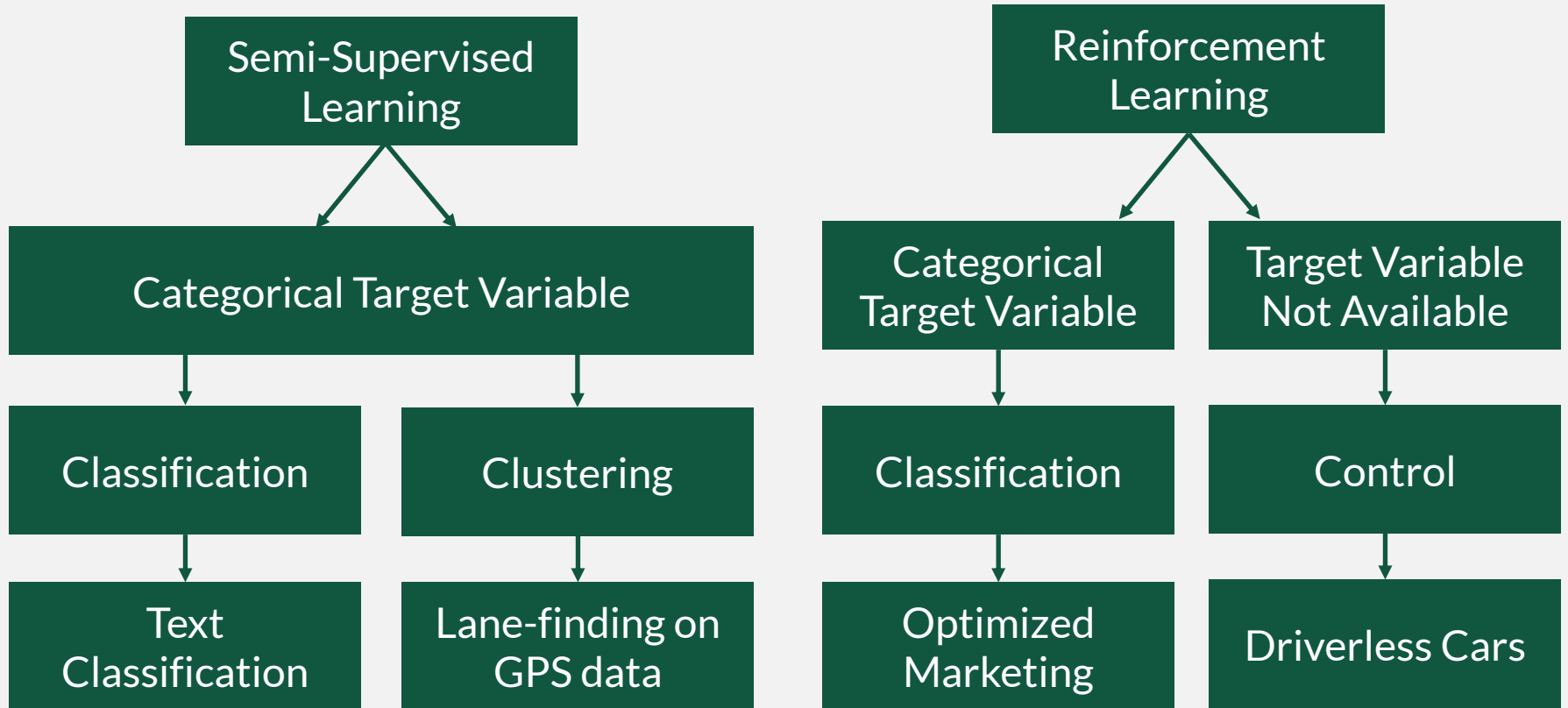
Representational Learning

Deep Learning

Machine Learning Taxonomy



Machine Learning Taxonomy



ML Representations

Supervised Learning

K Nearest Neighbor
Naïve Bayes
Decision Trees
Linear Regression
Support Vector Machine

Unsupervised Learning

K-means clustering
Association rule learning

Semi-Supervised Learning

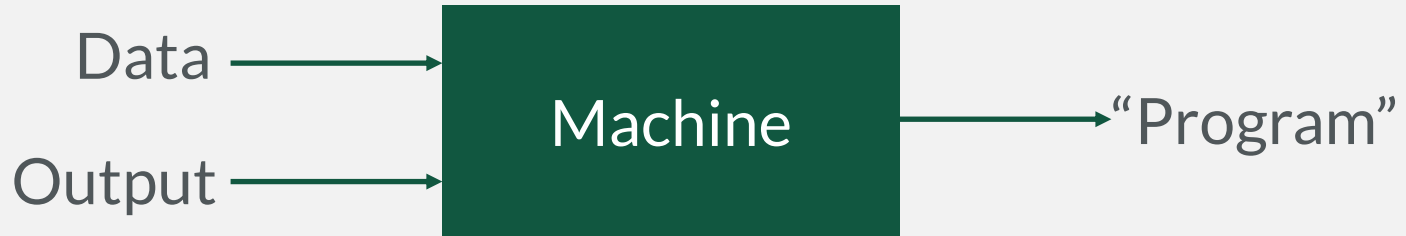
Self-Training of Existing Classifiers
Hidden Markov Models
Multiple Gaussian Distributions
Semi-supervised support vector machines

Reinforcement Learning

Q-Learning
Temporal Difference

Canonical Representation

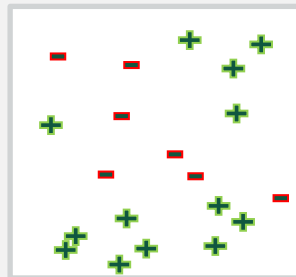
Machine Learning vs. Traditional Programming



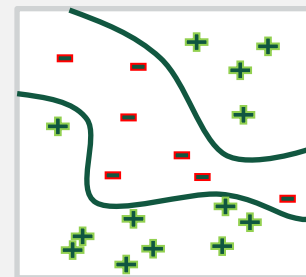
When do We Need Machine Learning?

Three Conditions:

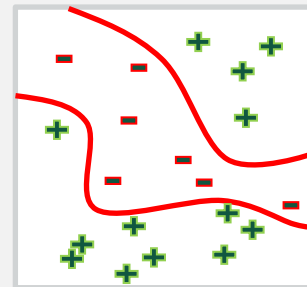
1. We have an Existing Dataset
2. A pattern exists in the data
3. The pattern is not easily defined by an equation



The Data



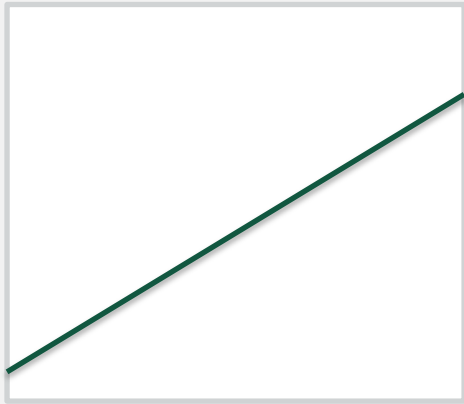
The Pattern



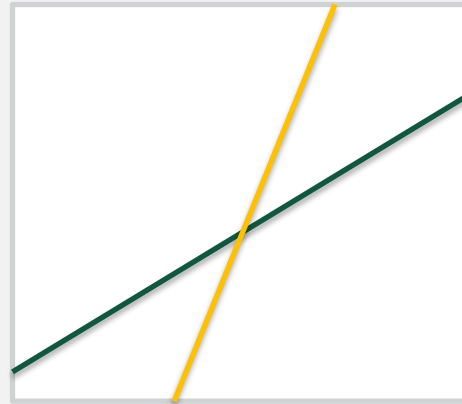
No Possible Equation

The Computational Learning Process

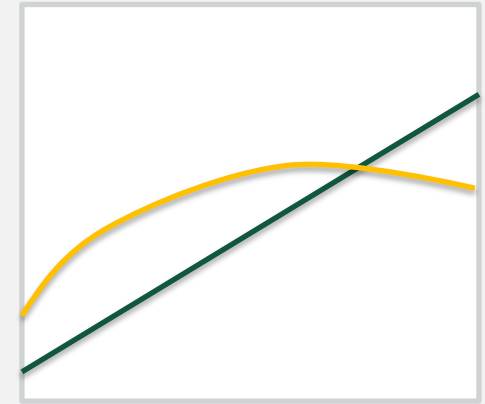
Learning Process



Target Function



First Approx.



Second Approx.

Supervised ML Applied to Image Classification

Important Note!

Our future examples focus on *Supervised Learning* for *Images*

However, the same principles apply to other types of data (*natural language* and *code*) and learning methods (*Unsupervised* and *Reinforcement*).

The Five Elements of the Learning Process

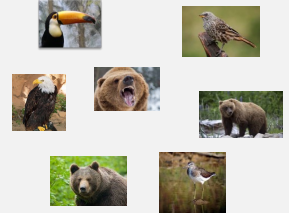
Input



Output

{Bird, Bear}

Data



Target
Function

$F(x)$

Hypothesis

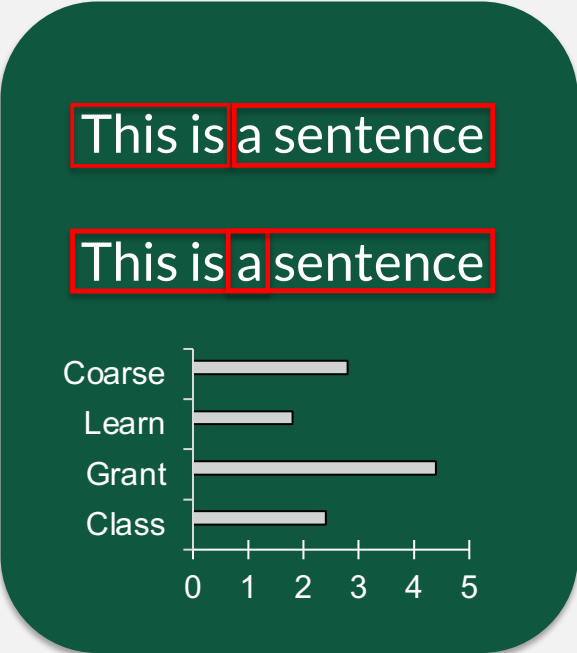
$G(x) \approx F(x)$

Feature Engineering for “Canonical” Machine Learning

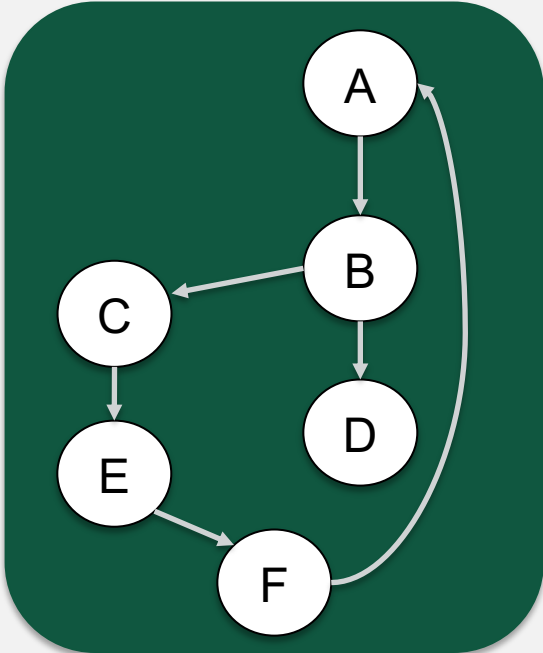
Images



Text



Source Code



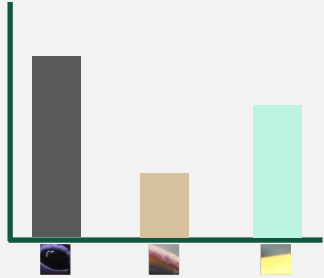
“Canonical” ML Image Classification

On the Large-Scale *ImageNet* Dataset,
which contains millions of images
from over 1000 categories

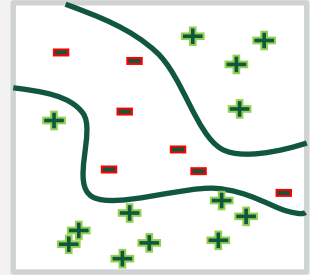
Canonical ML techniques have only been
able to achieve ~ *60% accuracy*

Shortcomings of Traditional ML Techniques

Manually
Derived
Features



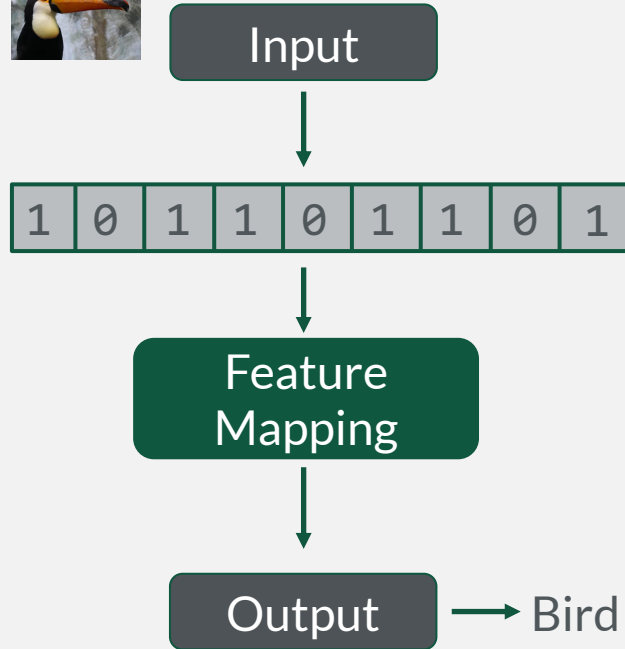
Complex
Kernels



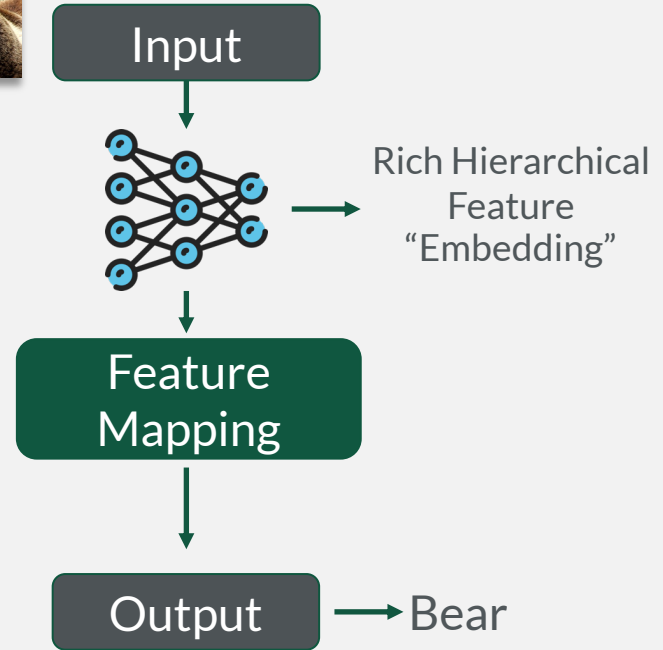
Shallow
Representation



The Advent of Deep Learning



“Canonical” Machine Learning



Deep Learning

ML Representations

Supervised Learning

K Nearest Neighbor
Naïve Bayes
Decision Trees
Linear Regression
Support Vector Machine
Neural Networks (Convolutional, Recurrent, Feed-forward, etc.)

Unsupervised Learning

K-means clustering
Association rule learning
Autoencoders
Deep Belief Networks
Generative Adversarial Networks (GANs)

Semi-Supervised Learning

Self-Training of Existing Classifiers
Hidden Markov Models
Multiple Gaussian Distributions
Semi-supervised support vector machines
Neural networks
Autoencoders

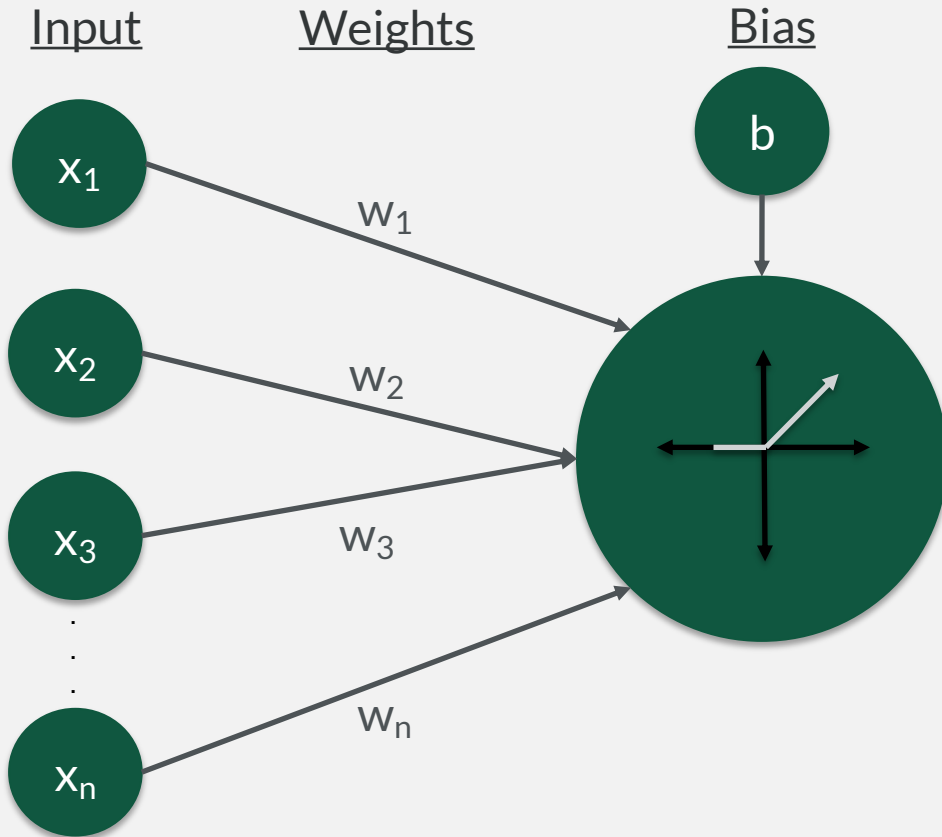
Reinforcement Learning

Q-Learning
Temporal Difference
Deep Adversarial Networks
Deep Q-Learning

Canonical Representation

Deep Representation

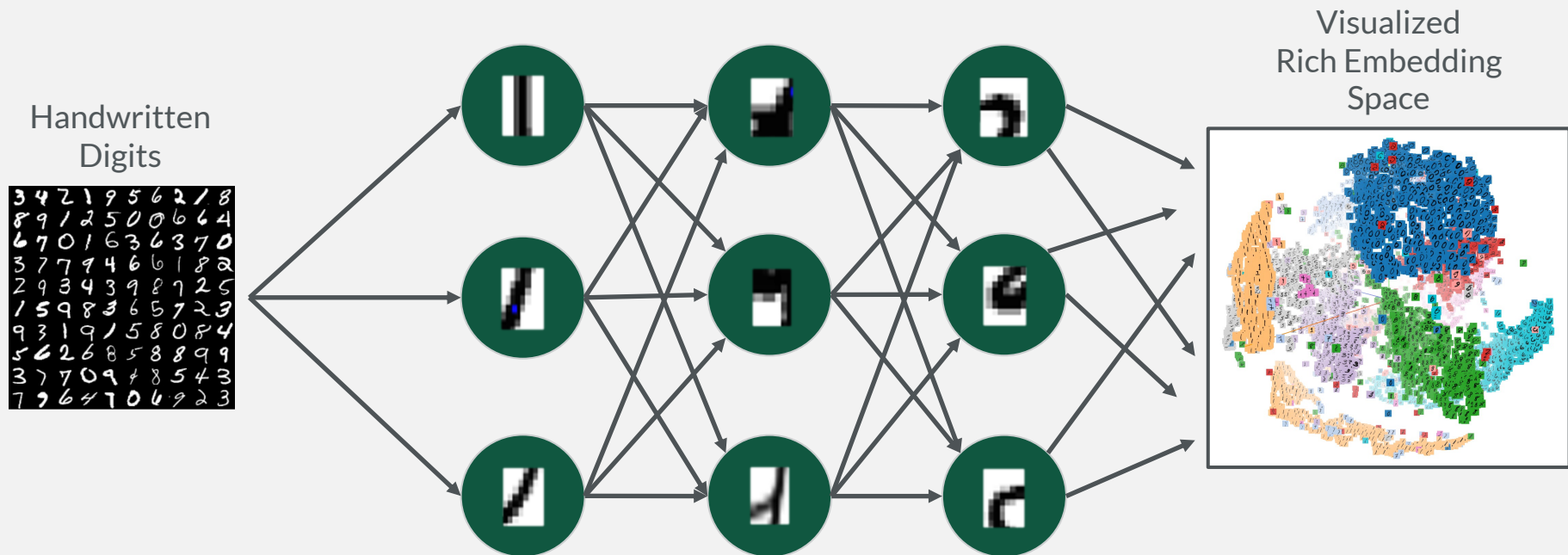
Neurons: The Building Blocks of Rich Features



Additional Activation Functions

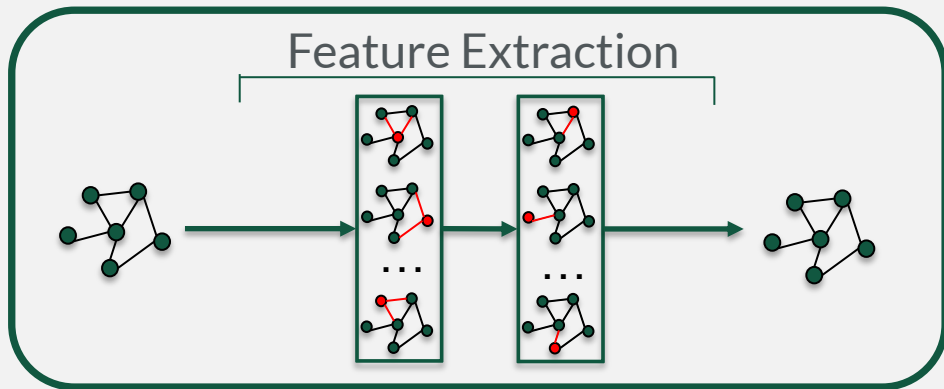
- Identity
- Binary Step
- Sigmoid
- Tanh
- Leaky ReLU
- Softmax

Neural “Networks” for Rich Embeddings

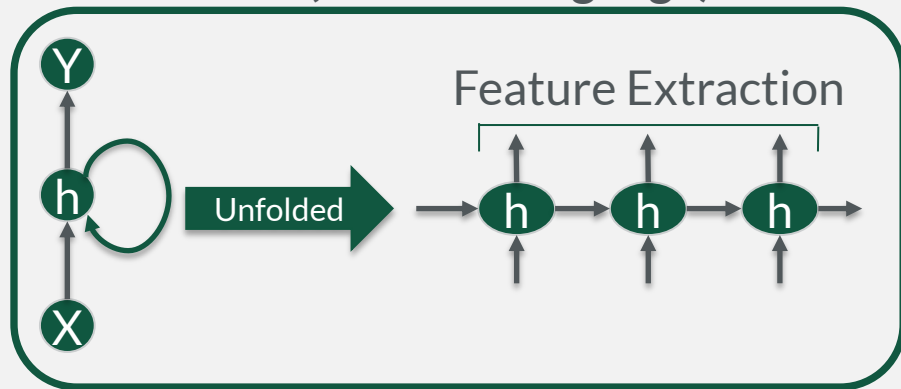


Automated Feature Discovery

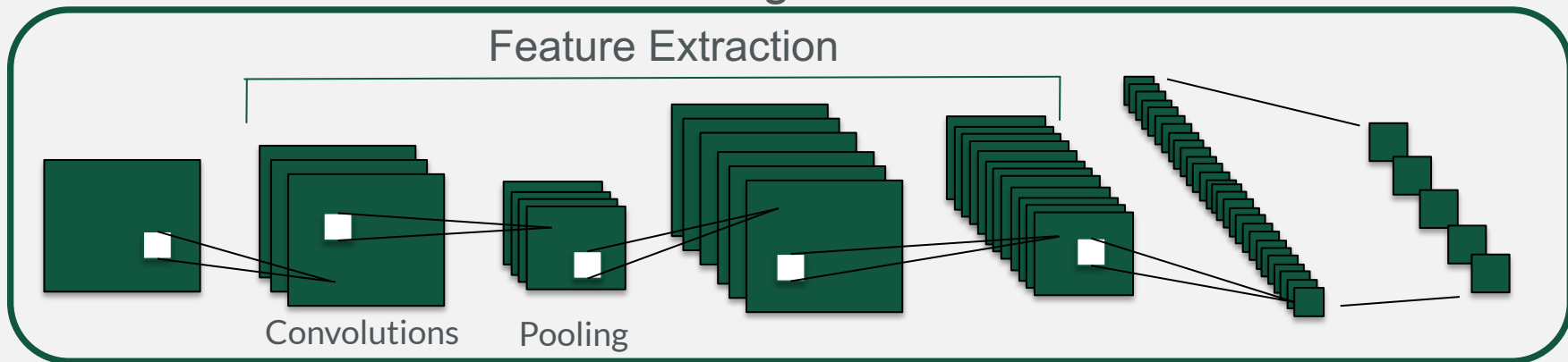
Code



Text (Natural Language)



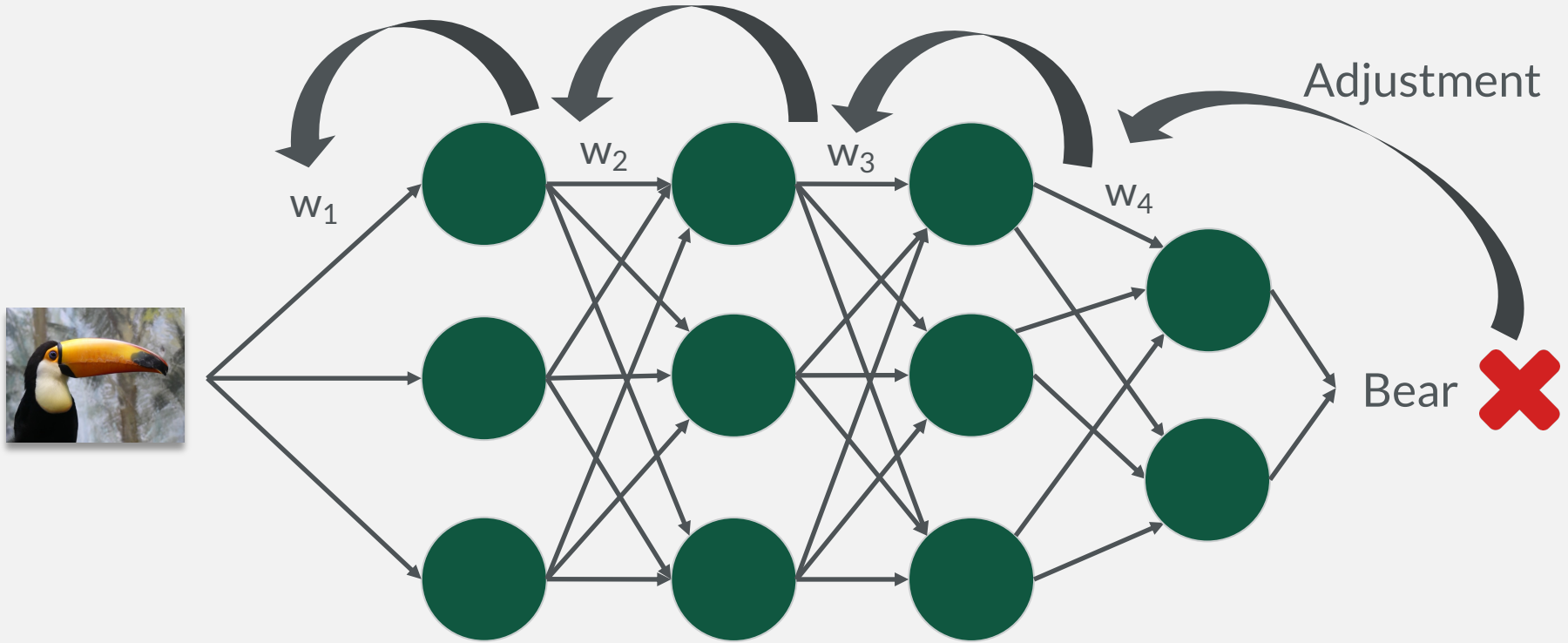
Images



How Can a Model Learn from Deep Embeddings?

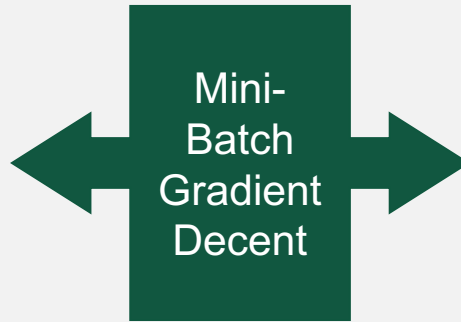
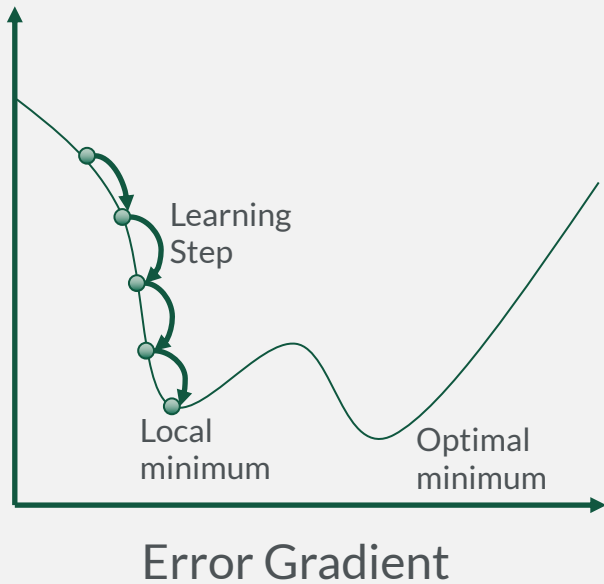
Adjust the Neuron “*weights*” according to *errors* made on a given task.

How Can a Model Learn from Deep Embeddings?

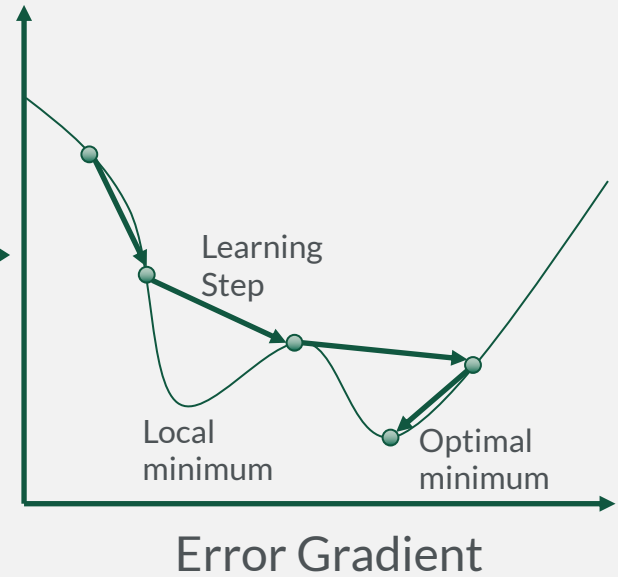


How Should the Weights be Updated?

Batch Gradient Decent



Stochastic Gradient Decent



CNN-Accuracy

ConvNets have *surpassed human levels of accuracy* on the ImageNet classification dataset

Deep Learning Advantages and Drawbacks

Advantages

- Does not require manual feature engineering
- Capable of Learning Rich, Hierarchical Data Representations
- Can be trained for a given task end-to-end

Disadvantages

- Require massive datasets to function effectively
- Computationally expensive to train
- Models can be difficult to interpret (Black Box)

Topic 2 – DL4SE: The Current State of Research

Mining Software Repositories



Google Play



Automation in Software Engineering Research



Source Code
Files



Software
Documentation



Screenshots



Screen
Recordings



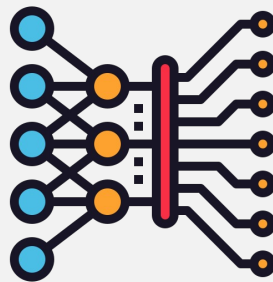
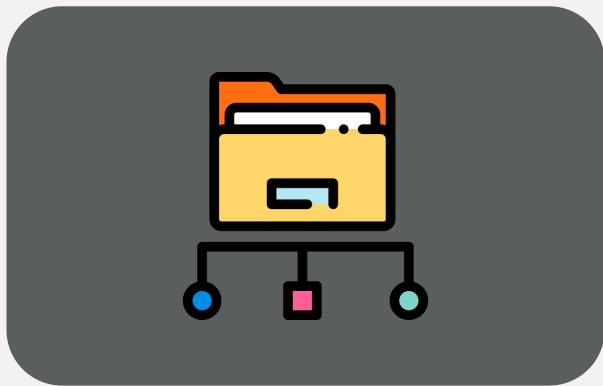
Bug Reports



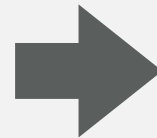
Design
Documents

Automation in Software Engineering Research

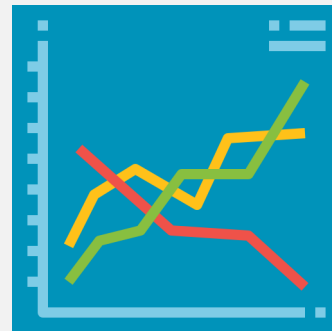
Software
Repository Data



Deep
Learning



Salient
Patterns



What is the current state-of-the-art of DL4SE?

Systematic Literature Review

###

A Systematic Literature Review on the Use of Deep Learning in Software Engineering Research

CODY WATSON, Washington & Lee University

NATHAN COOPER, William & Mary

DAVID NADER PALACIO, William & Mary

KEVIN MORAN, George Mason University

DENYS POSHYVANYK, William & Mary

An increasingly popular set of techniques adopted by software engineering (SE) researchers to automate development tasks are those rooted in the concept of Deep Learning (DL). The popularity of such techniques largely stems from their automated feature engineering capabilities, which aid in modeling software artifacts. However, due to the rapid pace at which DL techniques have been adopted it is difficult to distill the current successes, failures, and opportunities of the current research landscape. In an effort to bring clarity to this cross-cutting area of work, from its modern inception to the present, this paper presents a systematic literature review of research at the intersection of SE & DL. The review canvases work appearing in the most prominent SE and DL conferences and journals and spans 84 papers across 22 unique SE tasks. We center our analysis around the *components of learning*, a set of principles that govern the application of machine learning techniques (ML) to a given problem domain, discussing several aspects of the surveyed work at a granular level. The end result of our analysis is a *research roadmap* that both delineates the foundations of DL techniques applied to SE research, and likely areas of fertile exploration for the future.

CCS Concepts: • **Software and its engineering** → *Software creation and management; Software development techniques;*

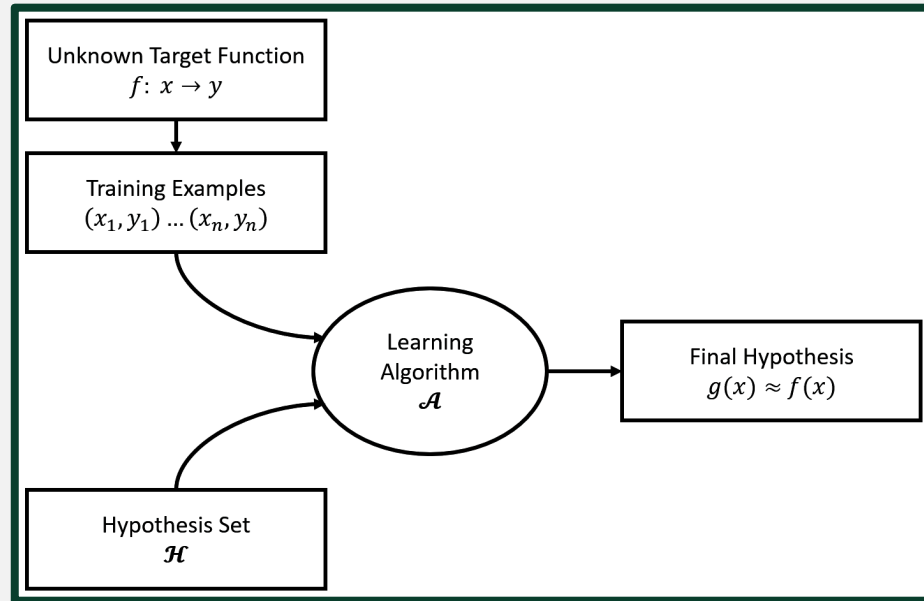
Additional Key Words and Phrases: deep learning, neural networks, literature review, software engineering, machine learning

Systematic Literature Review

Research Questions (RQs) centered upon the “components of learning”

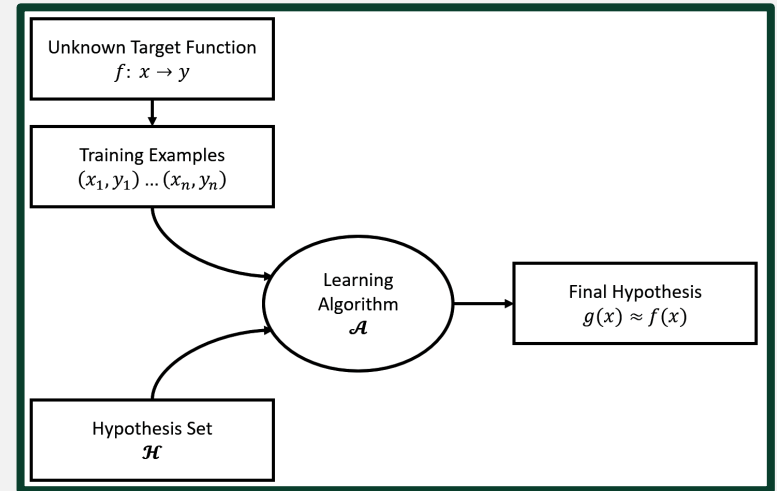
Systematic Literature Review

Research Questions (RQs) centered upon the “components of learning”



Systematic Literature Review

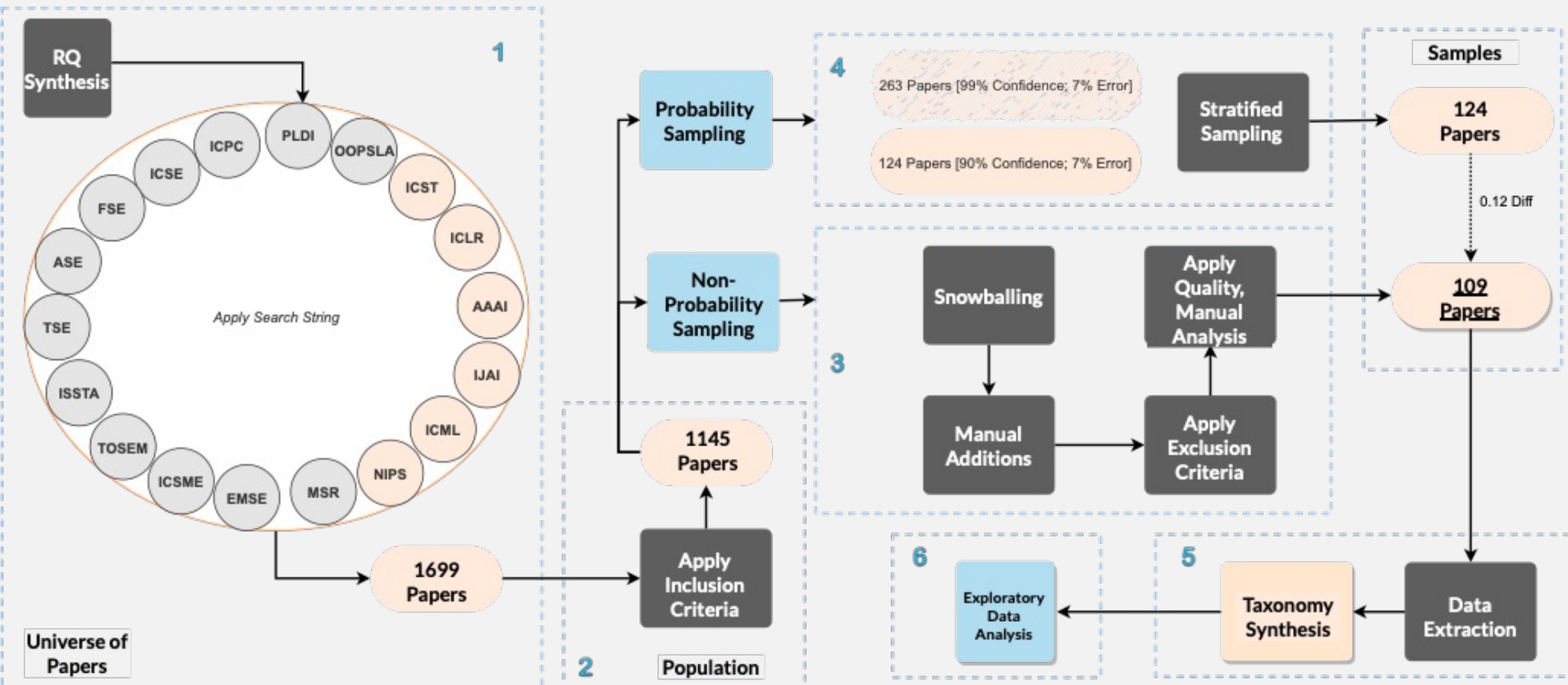
- **RQ₁**: Target Function (*SE Task*)
- **RQ₂**: Data (*Training/Testing Data*)
- **RQ₃**: Learning Model (*Algorithm + Hypothesis Set*)
- **RQ₄**: Final Hypothesis (*Results*)
- **RQ₅**: Reproducibility and Replicability



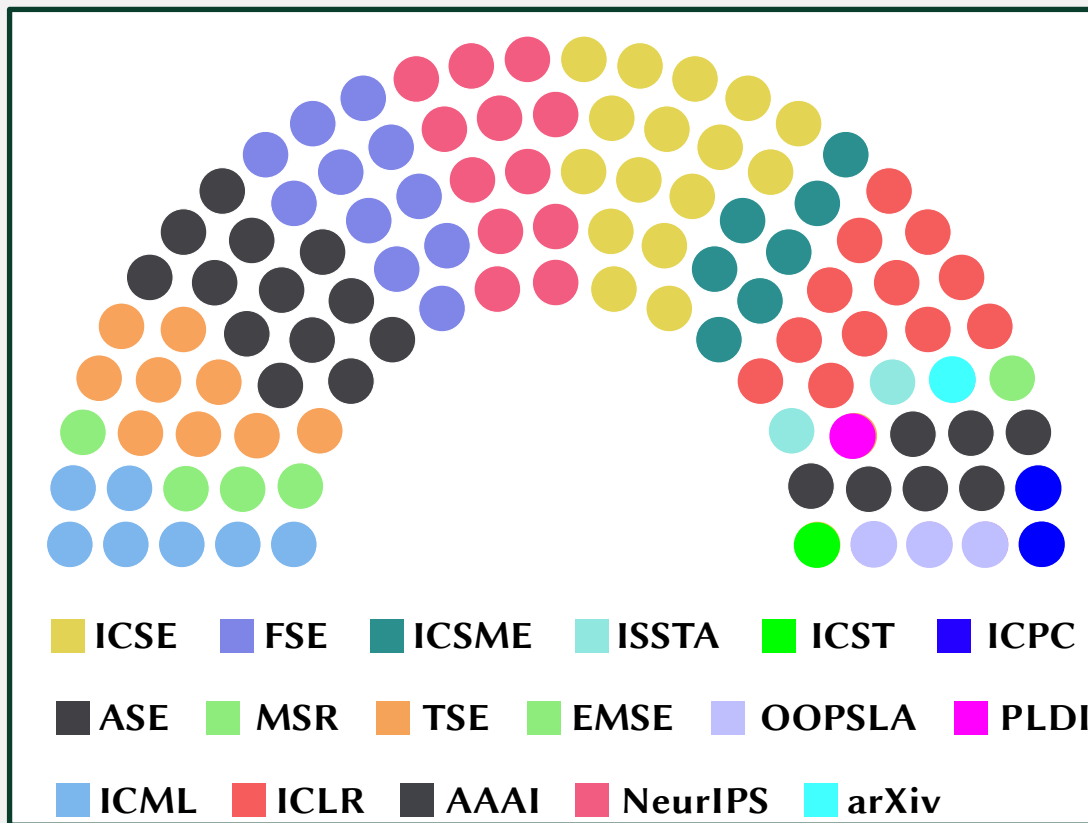
Systematic Literature Review

- **Time Period:** 2009-(mid)2019
- **Venues:** ICLR, NeurIPS, FSE, ICML, MSR, ISSTA, ICST, ICSE, ASE, ICSME, TSE, TOSEM, EMSE, OOPSLA, ICPC, PLDI, AAI, IJCAI.
- **Methodology:** Following *Kitchenham, et.al.*

SLR Search Process

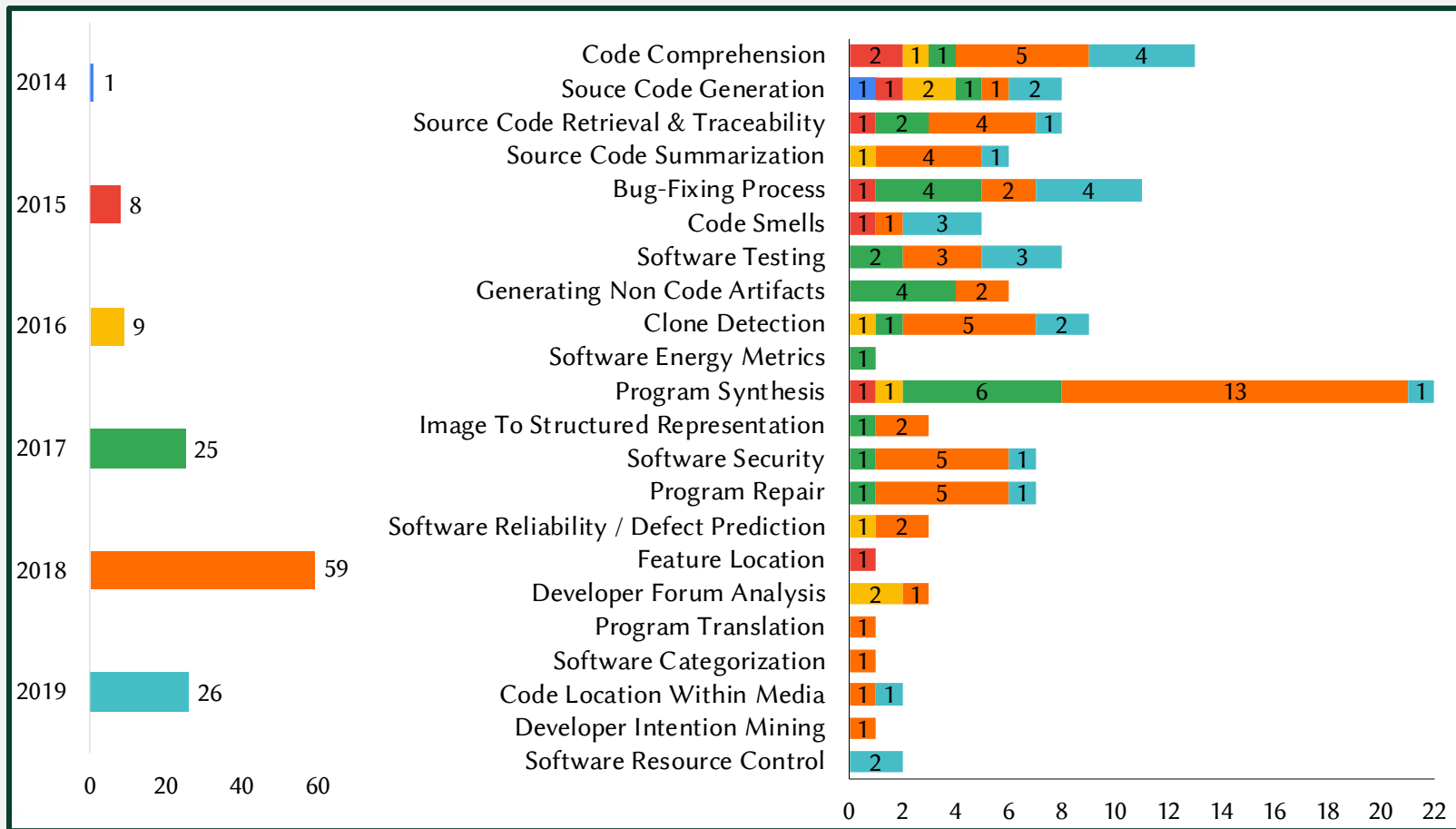


Publication Distribution By Venue



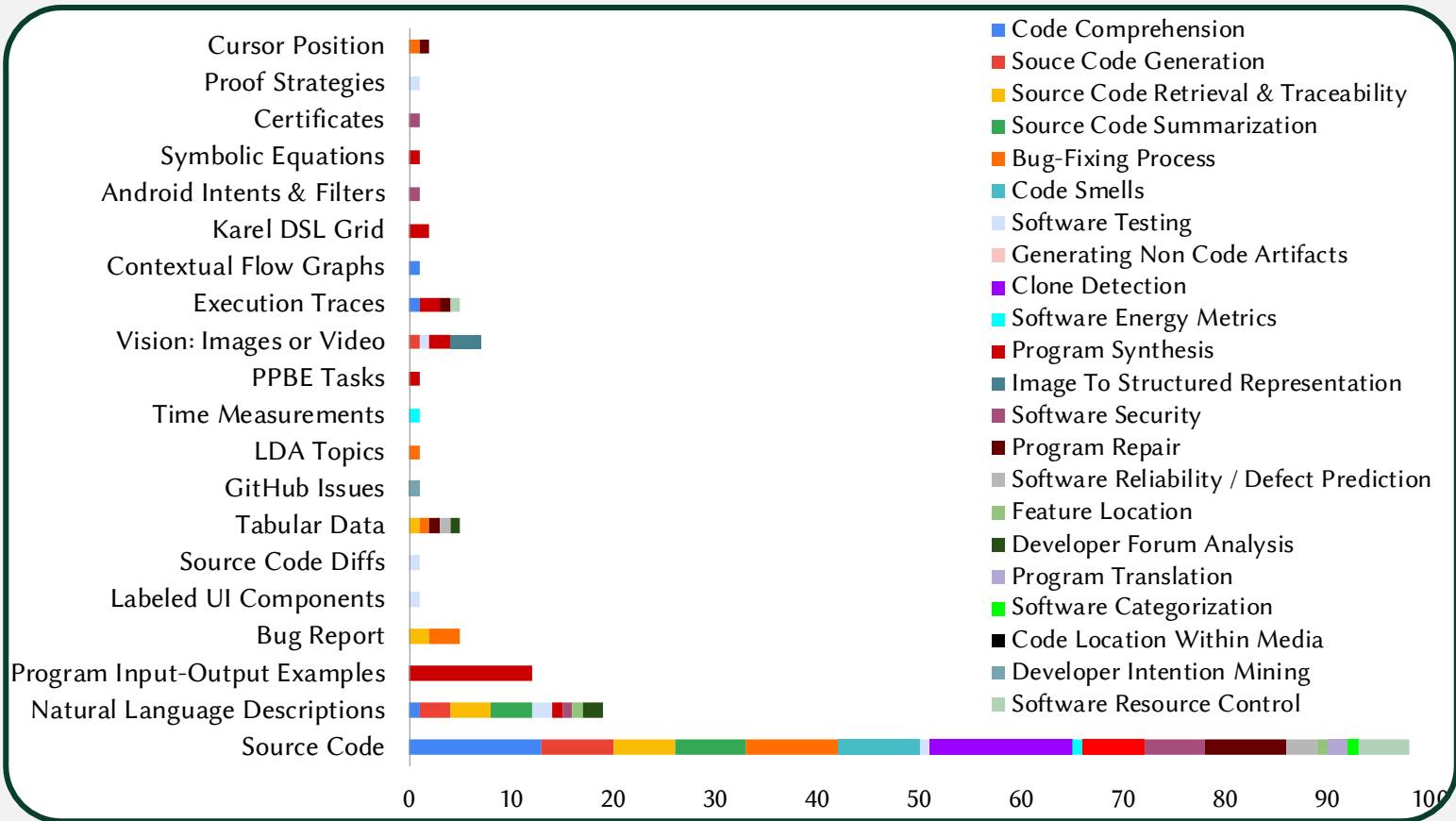
RQ₁: Target Function (SE Task)

DL4SE Publications Over Time and SE Tasks

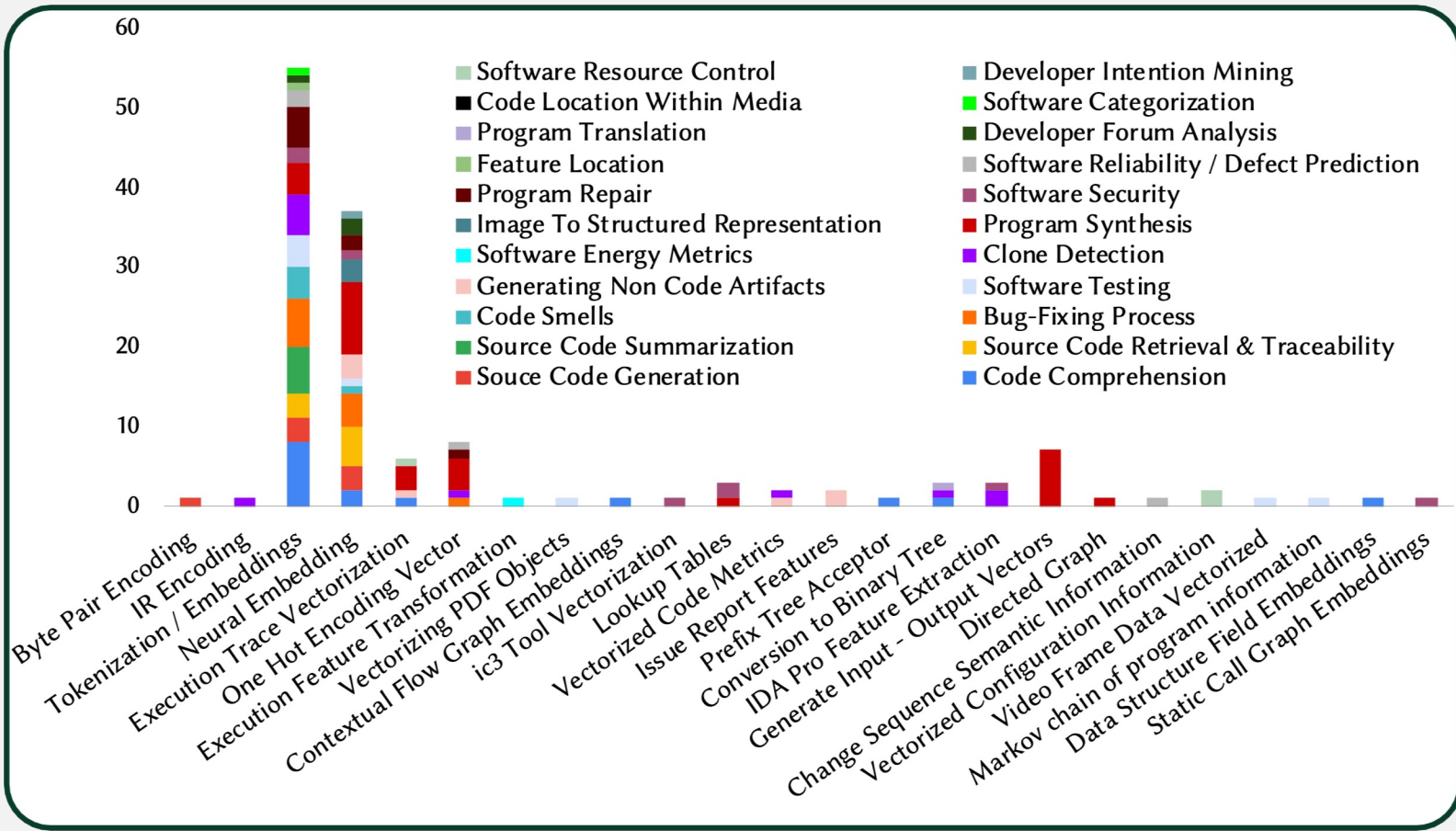


RQ₂: Data (Training/Testing Data)

Data Used in DL4SE Approaches by SE Task

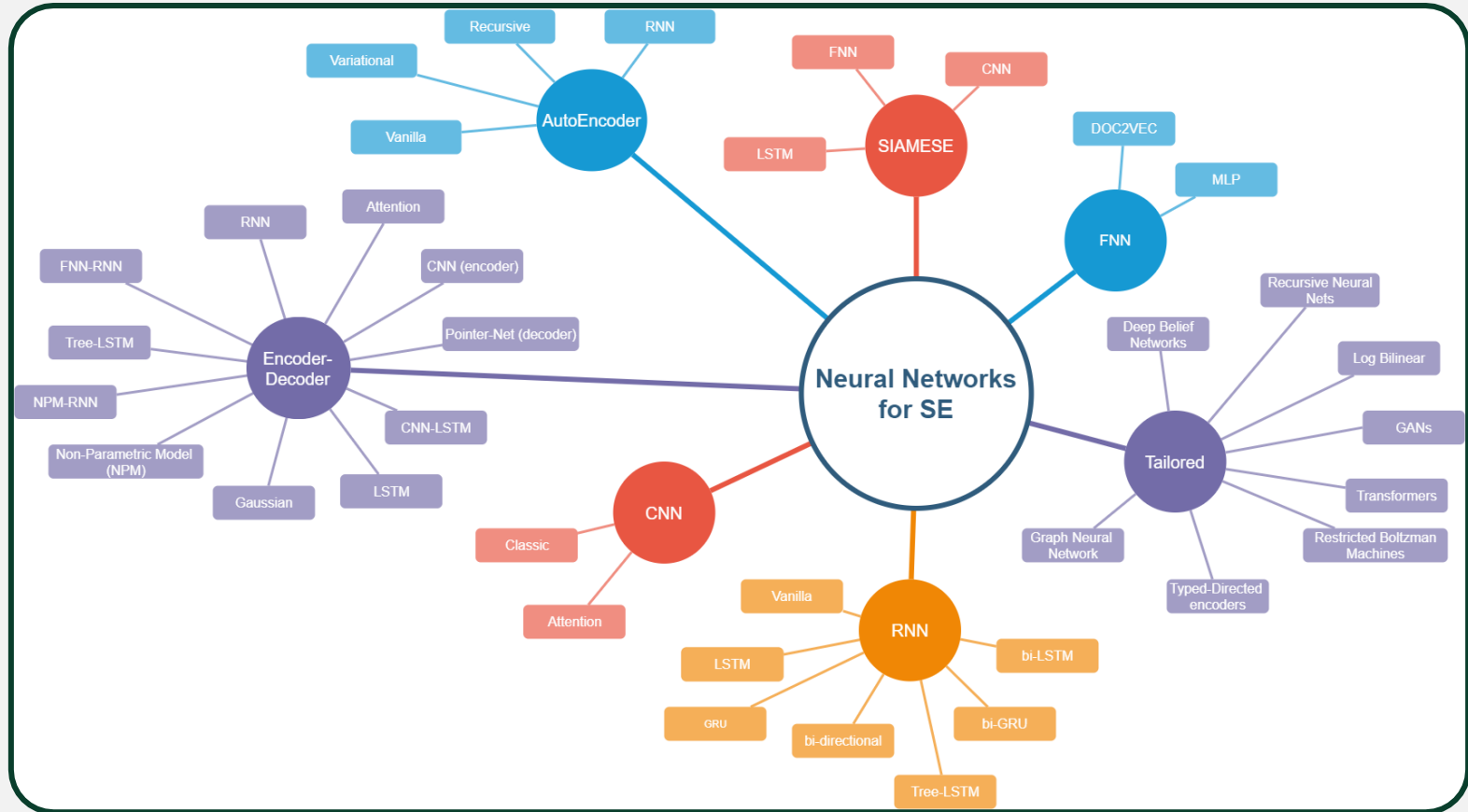


Data Processing Techniques by SE Task

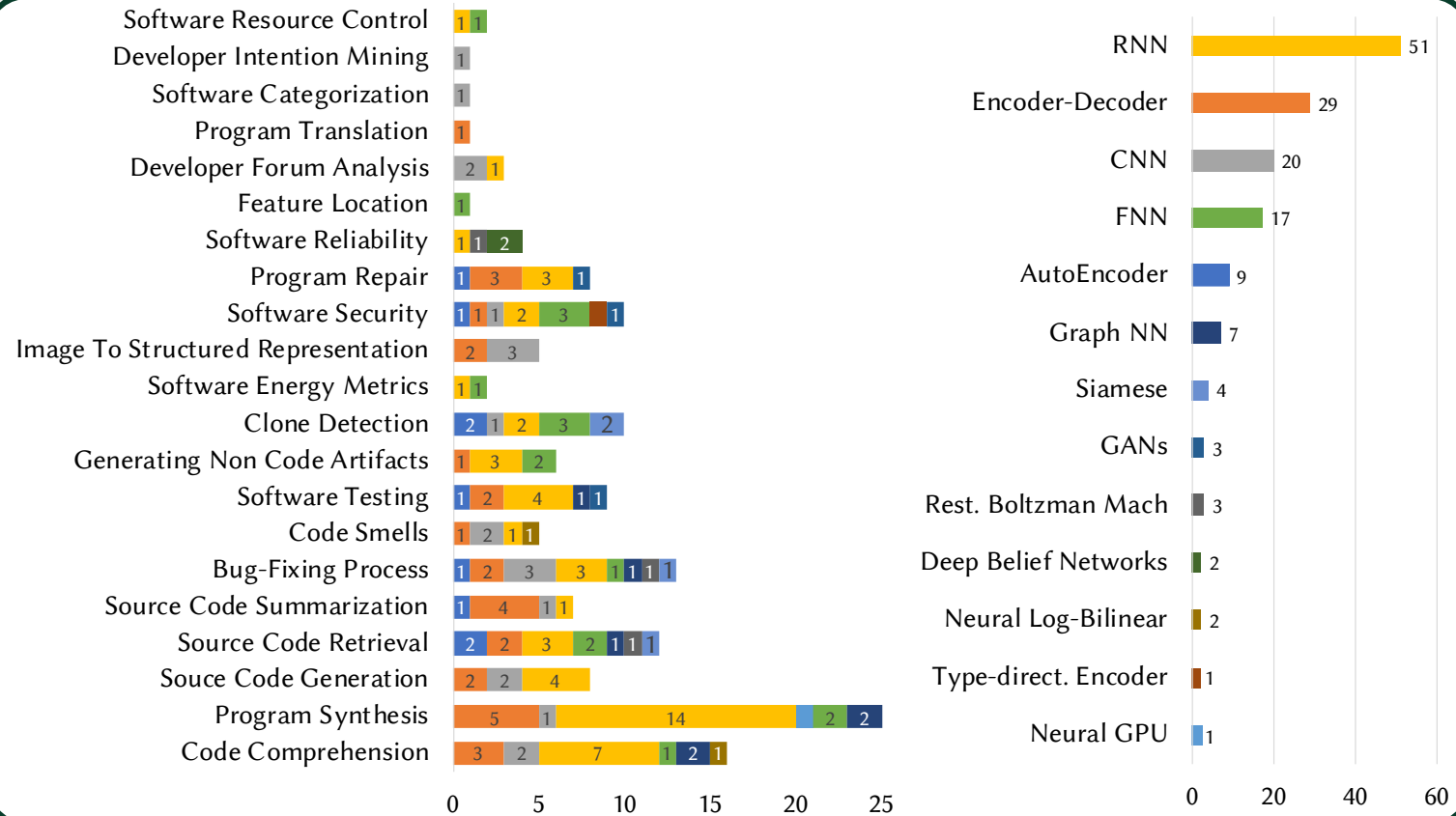


RQ₂: Learning Model (Algorithm + Hypothesis Set)

DL4SE Neural Network Architectures

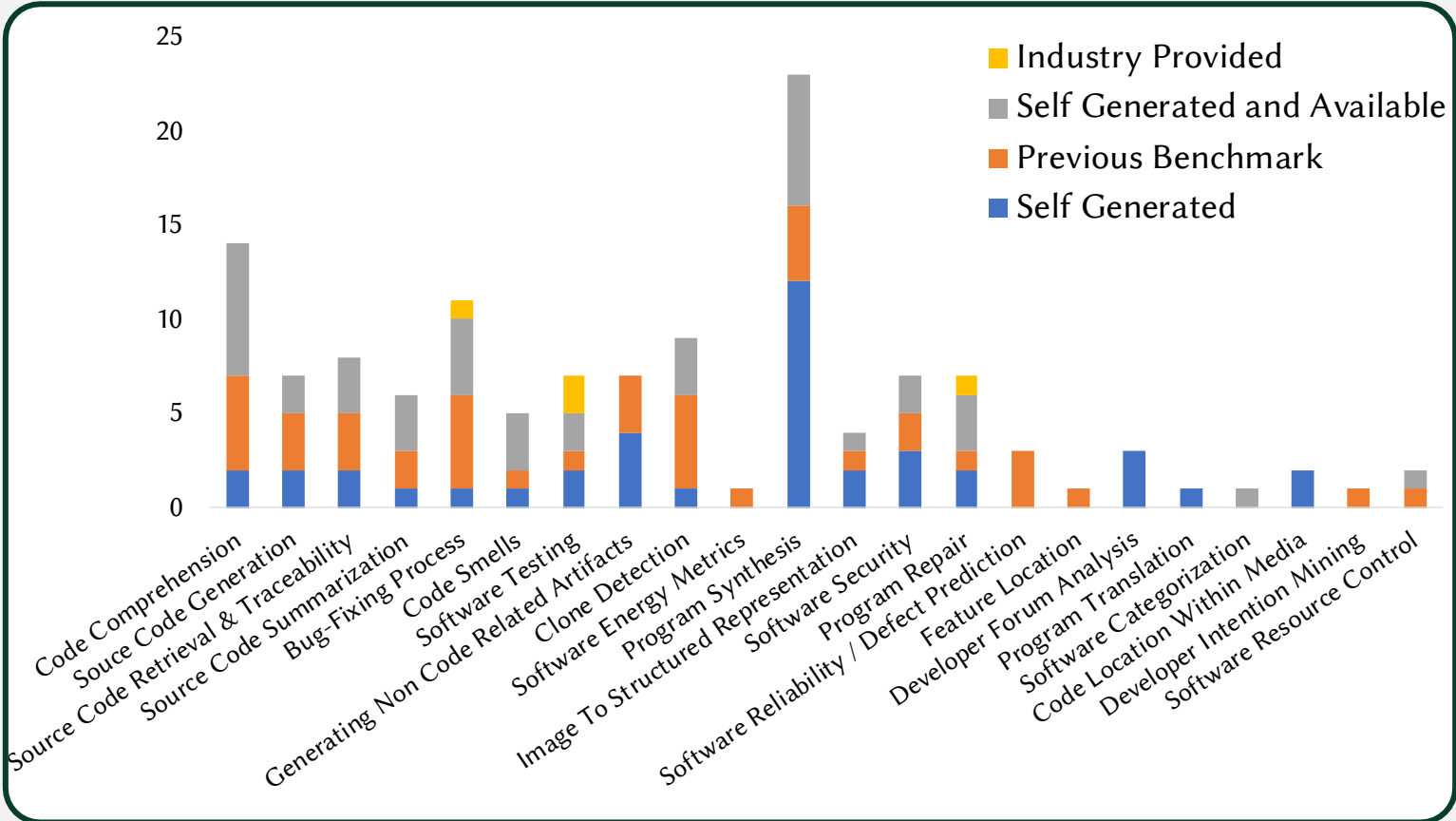


DL4SE Neural Network Architectures

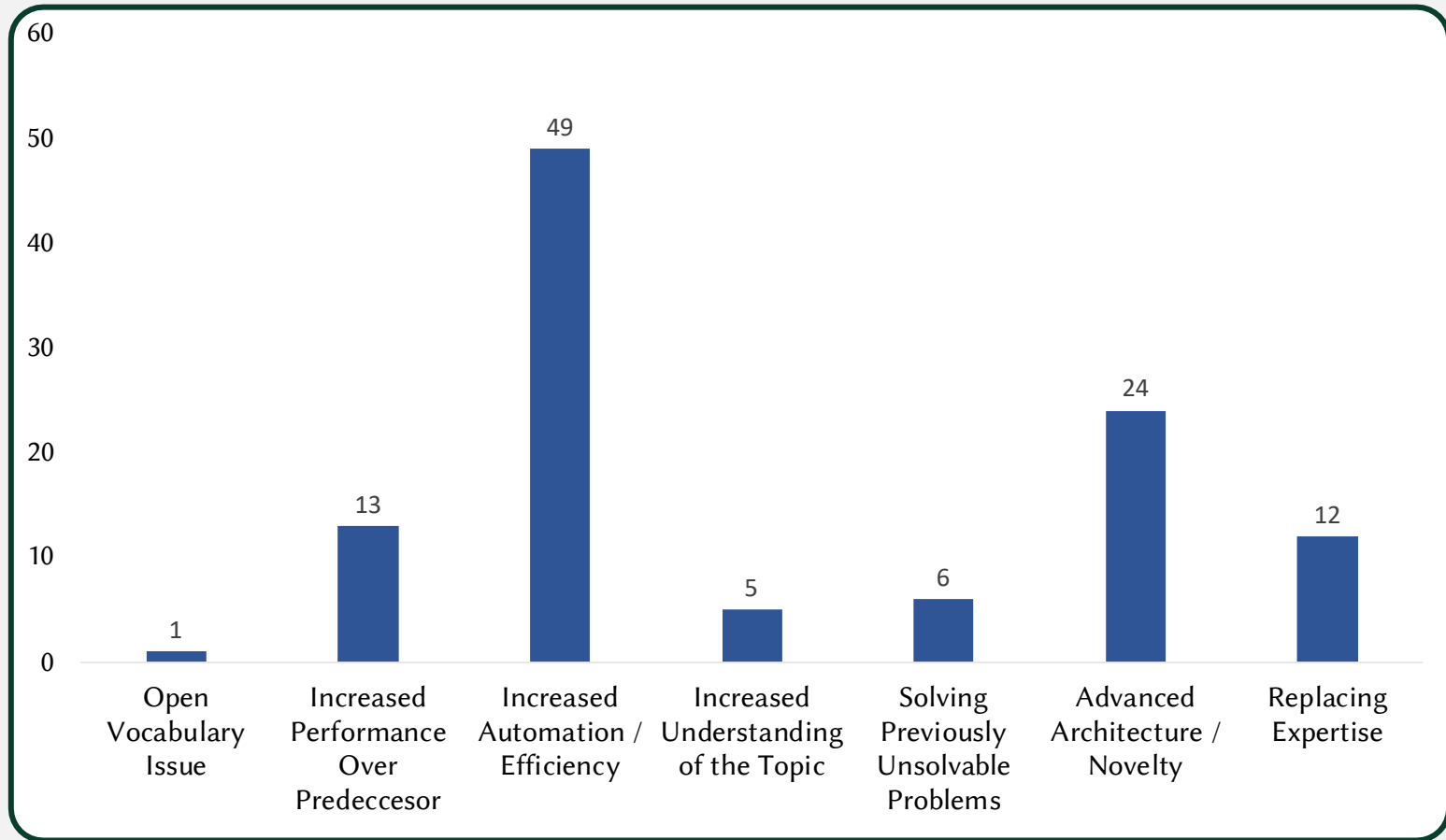


RQ₄: Final Hypothesis (Results)

DL4SE Benchmarks



Claimed DL4SE Impact



Consideration of Occam's Razor

Easy over Hard: A Case Study on Deep Learning

Wei Fu, Tim Menzies
Com.Sci., NC State, USA
wfu@ncsu.edu, tim.menzies@gmail.com

ABSTRACT

While deep learning is an exciting new technique, the benefits of this method need to be assessed with respect to its computational cost. This is particularly important for deep learning since these learners need hours (to weeks) to train the model. Such long training time limits the ability of (a) a researcher to test the stability of their conclusion via repeated runs with different random seeds; and (b) other researchers to repeat, improve, or even refute that original work.

For example, recently, deep learning was used to find which questions in the Stack Overflow programmer discussion forum can be linked together. That deep learning system took 14 hours to execute. We show here that applying a very simple optimizer called DE to fine tune SVM, it can achieve similar (and sometimes better) results. The DE approach terminated in 10 minutes; i.e. 84 times faster hours than deep learning method.

We offer these results as a cautionary tale to the software analytics community and suggest that not every new innovation should be applied without critical analysis. If researchers deploy some new and expensive process, that work should be baselined against some simpler and faster alternatives.

KEYWORDS

Search based software engineering, software analytics, parameter tuning, data analytics for software engineering, deep learning, SVM,

semantically related, they are considered as *linkable* knowledge units.

In their paper, they used a convolution neural network (CNN), a kind of deep learning method [42], to predict whether two KUs are linkable. Such CNNs are highly computationally expensive, often requiring network composed of 10 to 20 layers, hundreds of millions of weights and billions of connections between units [42]. Even with advanced hardware and algorithm parallelization, training deep learning models still requires hours to weeks. For example:

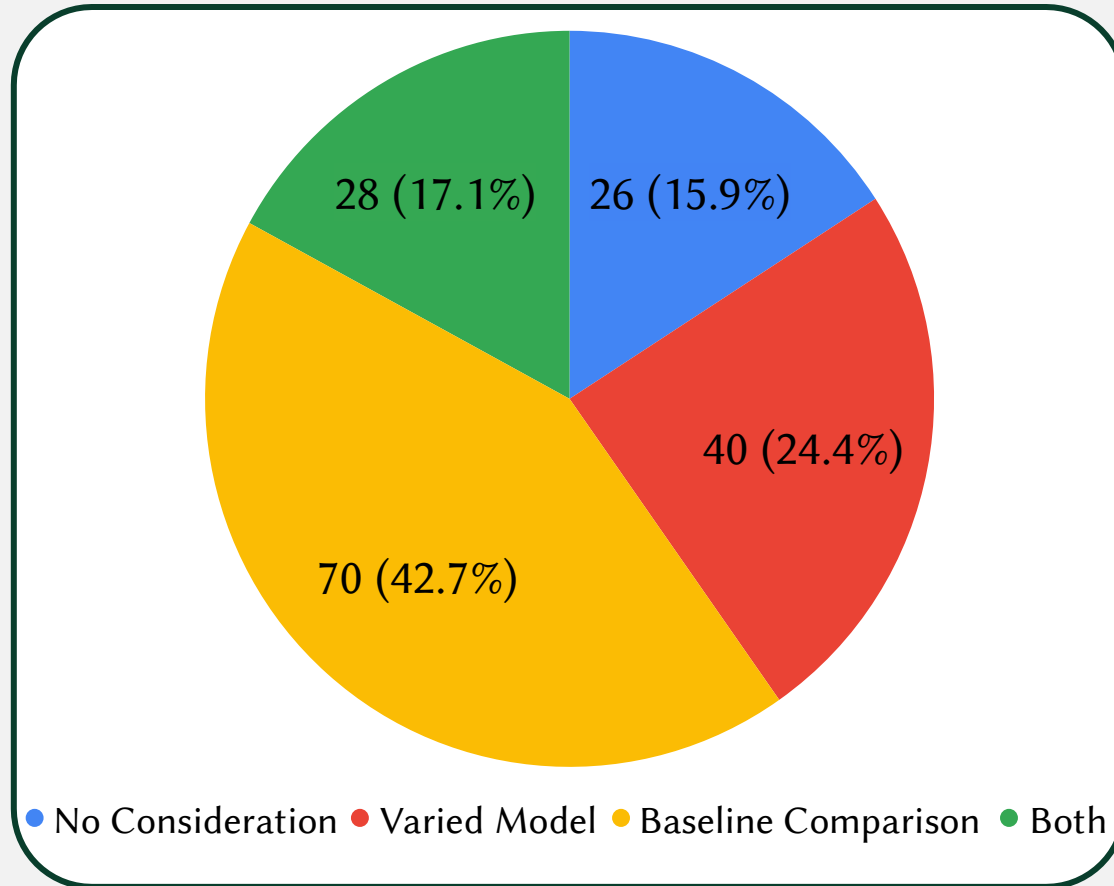
- XU report that their analysis required 14 hours of CPU.
- Le [40] used a cluster with 1,000 machines (16,000 cores) for three days to train a deep learner.

This paper debates what methods should be recommended to those wishing to repeat the analysis of XU. We focus on whether using simple and faster methods can achieve the results that are currently achievable by the state-of-art deep learning method. Specifically, we repeat XU's study using DE (differential evolution [62]), which serves as a hyper-parameter optimizer to tune XU's baseline method, which is a conventional machine learning algorithm, support vector machine (SVM). Our study asks:

RQ1: Can we reproduce XU's baseline results (Word Embedding + SVM)? Using such a baseline, we can compare our methods to those of XU.

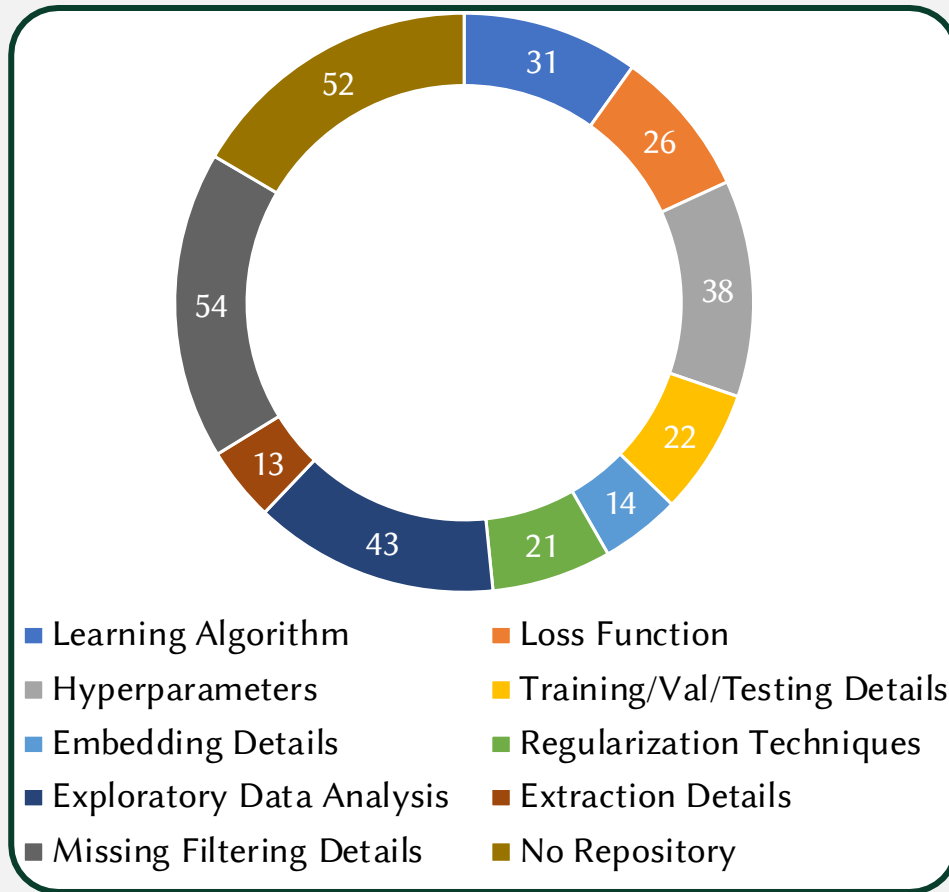
RQ2: Can DE tune a standard learner such that it outperforms XU's deep learning method? We apply differential evolution to tune

Consideration of Occam's Razor

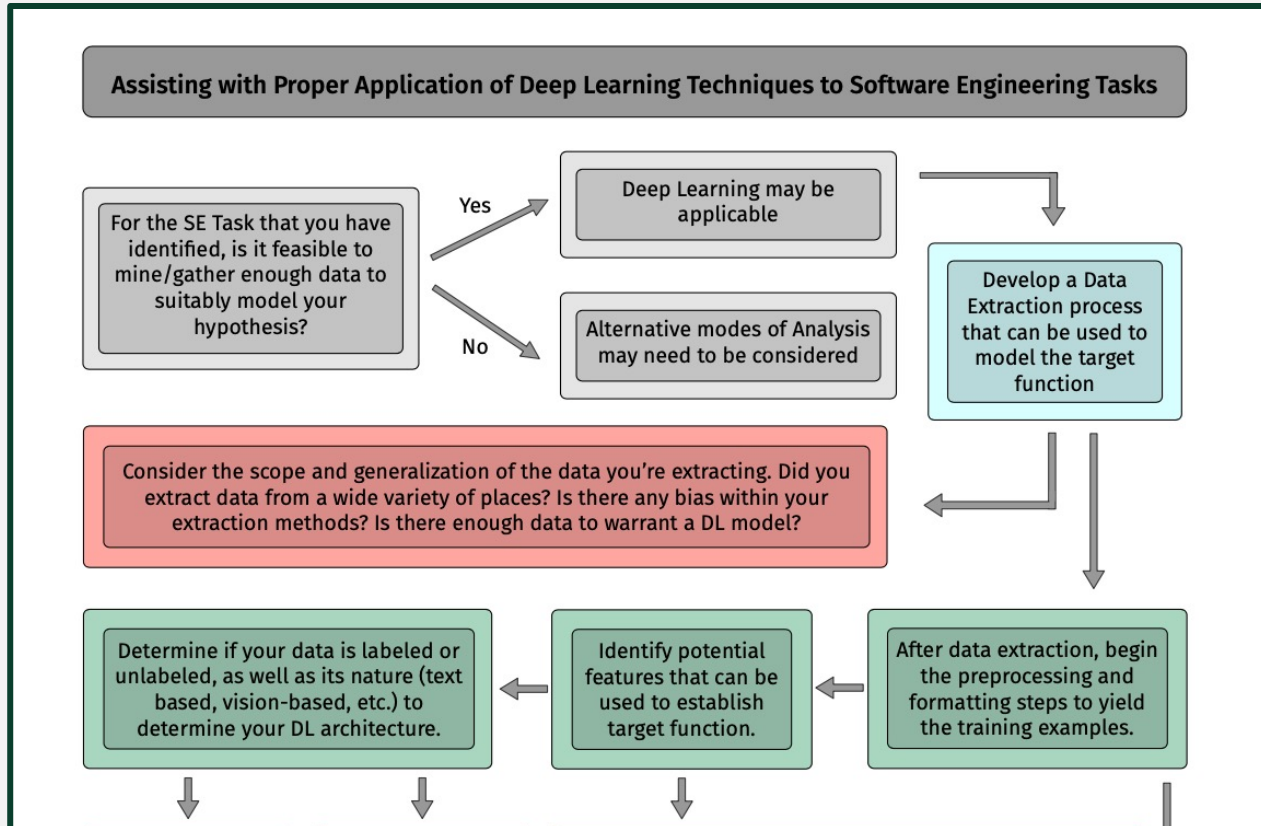


RQ₅: Reproducibility & Replicability

Non-Reproducibility Factors



Resulting Guidelines



Topic 3 – Looking Ahead: Future Directions and Paths Forward



NSF Workshop on Deep Learning & Software Engineering

November 10th & 11th, 2019

San Diego, California



DL4SE and SE4DL

DL4SE: Leveraging Deep Learning Techniques in order to automate or improve existing software engineering tasks

SE4DL: Where Deep Learning Techniques are viewed as a new form of software development that needs tool and process support

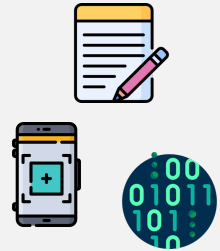
Future Work on DL4SE

Future Research Directions in DL4SE

Combining Features Learned via DL with Existing Empirical Knowledge



Leveraging & Combining Heterogenous Sources of SE Data



Developing Architectures tailored for SE Data



Systematic & Reproducible Research Methodology



Ethical & Social Considerations of DL4SE



Future Research Directions in DL4SE (cont'd)

Designing new
Effectiveness
Metrics for SE-
specific Tasks



HCI Aspects of AI-
assisted Developer
Tools



Development
of Tailored
“Clean”
Community
Datasets



New
Application
Areas



New
Data
Sources



Ethical and Social Considerations of DL4SE

↑ 21 AI is emitting secrets #45

Answered by nat dtjm asked this question in Report Bugs


dtjm 2 days ago

I tried to get it to tell me secrets and it did:

```
1 package main
2
3 func main() {
4
5
6 // send an email with sendgrid.com
7 // https://sendgrid.com/docs/User_Guide/Settings/email.html
8 // https://sendgrid.com/docs/User_Guide/Settings/api_keys.html
9 // https://github.com/sendgrid/sendgrid-go
10 // https://github.com/sendgrid/sendgrid-go/blob/master/examples/helpers/mail/main.go
11 // https://github.com/sendgrid/sendgrid-go/blob/master/examples/helpers/mail/main.go
12 // https://github.com/sendgrid/sendgrid-go/blob/master/examples/helpers/mail/main.go
13 // https://github.com/sendgrid/sendgrid-go/blob/master/examples/helpers/mail/main.go
14 // https://github.com/sendgrid/sendgrid-go/blob/master/examples/helpers/mail/main.go
15
16 apiKey :=
17
18 }
```

```
18 from := ""
19
20 =====
21
22 Accept Solution
23 "SG.f72"
24 from := ""
25
26 =====
27
28 Accept Solution
29 "SG.d3"
30 from := ""
31
32 =====
33
34 Accept Solution
35 "SG.f"
36 sender := ""
37
38 =====
39
40 Accept Solution
41 "SG.V4"
42 host := "https://api.sendgrid.com"
43 request := sendgrid.NewRequest(apiKey, "/v3/mail/send", host)
44 request.Method = "POST"
45 request.Body = []byte(`{
46   "personalizations": [
47     {
48       "to": [
49         {
50           "email": ""
51         }
52       ]
53     }
54   ]
55 }`)
56 }
```





Ethical and Social Considerations of DL4SE

 **nat** 3 days ago Maintainer edited ✓ ...

Hi folks, this is addressed in the FAQ: <http://copilot.github.com/#faq-does-github-copilot-ever-output-personal-data>

Does GitHub Copilot ever output personal data?
Because GitHub Copilot was trained on publicly available code, its training set included public personal data included in that code. From our internal testing, we found it to be extremely rare that GitHub Copilot suggestions included personal data verbatim from the training set. In some cases, the model will suggest what appears to be personal data – email addresses, phone numbers, access keys, etc. – but is actually made-up information synthesized from patterns in training data. For the technical preview, we have implemented a rudimentary filter that blocks emails when shown in standard formats, but it's still possible to get the model to suggest this sort of content if you try hard enough.

These secrets are almost entirely fictional, synthesized from the training data. GitHub already has a secret scanning feature that integrates with 50+ partners to disable tokens that are accidentally committed to public repos: <https://docs.github.com/en/code-security/secret-security/about-secret-scanning>

  6  6  2 3 replies

HCI Aspects of AI-Assisted Developer Tools

When Code Completion Fails: a Case Study Real-World Completions

Vincent J. Hellendoorn
Department of Computer Science
UC Davis
Davis, USA
vhellendoorn@ucdavis.edu

Sebastian Proksch
Department of Informatics
University of Zurich
Zürich, Switzerland
proksch@ifi.uzh.ch

Harald C. Gall
Department of Informatics
University of Zurich
Zürich, Switzerland
gall@ifi.uzh.ch

Alberto Bacchelli
Department of Informatics
University of Zurich
Zürich, Switzerland
bacchelli@ifi.uzh.ch

Abstract—Code completion is commonly used by software developers and is integrated into all major IDE’s. Good completion tools can not only save time and effort but may also help avoid incorrect API usage. Many proposed completion tools have shown promising results on synthetic benchmarks, but these benchmarks make no claims about the realism of the completions they test. This lack of grounding in real-world data could hinder our scientific understanding of developer needs and of the efficacy of code completion tools that were applied by 66 real developers, which we study and contrast with artificial completions to inform future research and tools in this area. We find that synthetic benchmarks misrepresent many aspects of real-world completions, which completion tools were far less accurate on real-world data. Worse, on the few completions that consumed most of the least predictable time, prediction accuracy was less than 20%—an effect that is invisible in synthetic benchmarks. Our findings have ramifications for future benchmarks, tool design and real-world efficacy. Benchmarks must account for completions that developers use most, such as intra-project APIs; models should be designed to be amenable to quantifying performance on the least predictable completions, which are both most time-consuming and far more essential to intra-project data suggest. We publicly release our findings (https://doi.org/10.5281/zenodo.2565673) and replicate our work (https://doi.org/10.5281/zenodo.2565673).

that happens locally takes place in a very different order and context from what is eventually committed, so that simulations in published code probably do not reflect the developer’s working context [12], [13]. Code completion tool design could greatly benefit from real-world observations from a developer study to improve their code completion tool [5].

An empirical foundation is needed to establish the characteristics of real-world code completion usage and the objectives that code completion tools should meet to have real-world efficacy. Our case study provides the first step toward that foundation by analyzing over 15,000 completions from a dataset of 66 developers in Visual Studio Code, applied by first in terms of their character set, and then in terms of their structured records.

RO

IN-IDE CODE GENERATION FROM NATURAL LANGUAGE: PROMISE AND CHALLENGES

Frank F. Xu
Carnegie Mellon University
fangzhxu@cs.cmu.edu

Bogdan Vasilescu
Carnegie Mellon University
vasil@scu@cs.cmu.edu

Graham Neubig
Carnegie Mellon University
gneubig@cs.cmu.edu

A PREPRINT
February 1, 2021

ABSTRACT

A great part of software development involves conceptualizing or communicating the underlying procedures and logic that needs to be expressed in programs. One major difficulty of programming is turning *concepts* into *code*, especially when dealing with the APIs of unfamiliar libraries. Recently, there has been a proliferation of machine learning methods that generate and retrieval from *natural language queries*, but these have primarily been evaluated purely based on code accuracy or overlap of generated code with developer-written code, and the actual effect of these methods on goal accuracy or workflow is surprisingly untested. In this paper, we perform the first comprehensive investigation of the productivity and accuracy, how does it affect the developer experience, and what are the challenges of using such technology inside the IDE, asking “at the current state of technology and what are the fine-grained code edits). We ask developers with various backgrounds to facilitate the study, we first develop a plugin for the IDE that implements a hybrid approach of productivity, code quality, and orchestrate virtual environments to enable collection of many tasks ranging from basic file manipulation to machine learning. While qualitative surveys of developer correctness are essential to productivity, code quality, or program correctness are not enough. We also conduct structured interviews to understand the effectiveness of future machine learning approaches that could improve the effectiveness of code generation. Our study illustrates a road for future empirical studies on this

19 Jan 2021

New Application Areas and Data-Sources

Potential SE Tasks

Software
Testing

Code
Review

Bug
Triaging

Troubleshooting
Tasks

Requirements
Engineering

Potential Data Sets

Tailored
for SE
Tasks

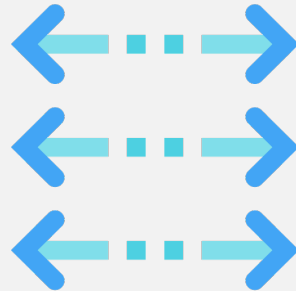
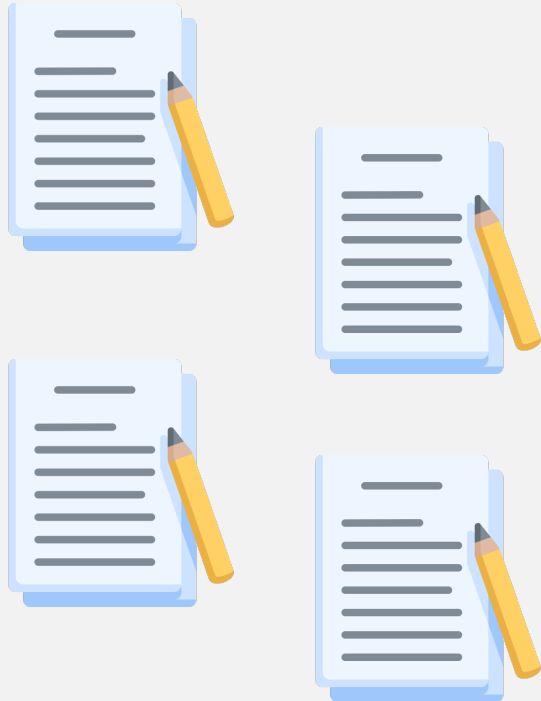
Graphical
Software
Artifacts

IDE
Instrumentation

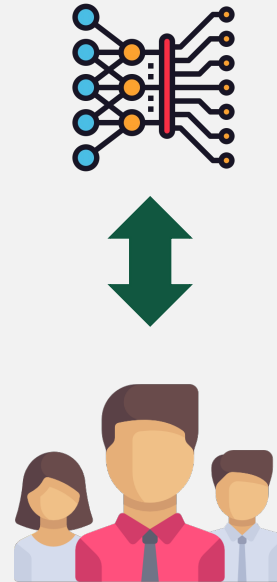
EDA for
Datasets

Combining Empirical Knowledge with Deep Learning

Empirical SE Studies



Deep Learning Tools



Future Work on SE4DL

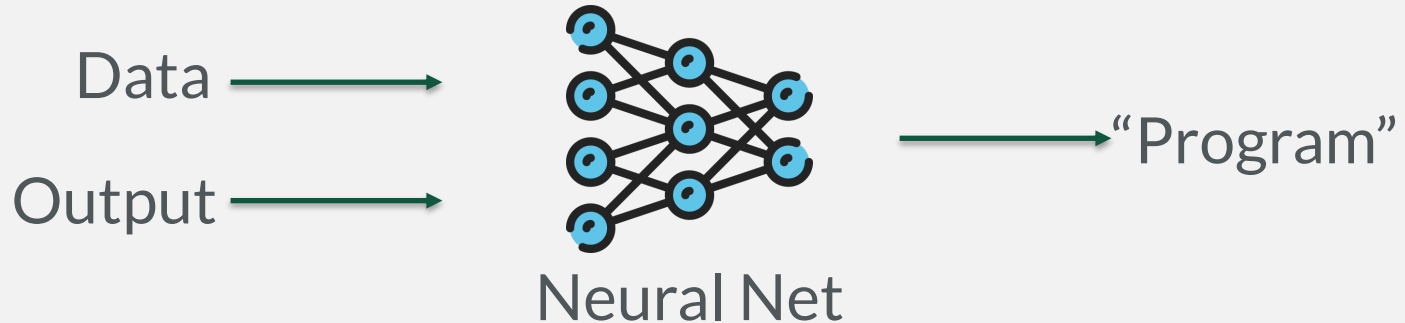
“Gradient descent can write code better than you. I’m sorry”

-Andrej Karpathy, Director of AI at Tesla

“Neural networks are not just another classifier, they represent the beginning of a fundamental shift in how we write software. They are Software 2.0.”

-Andrej Karpathy, Director of AI at Tesla

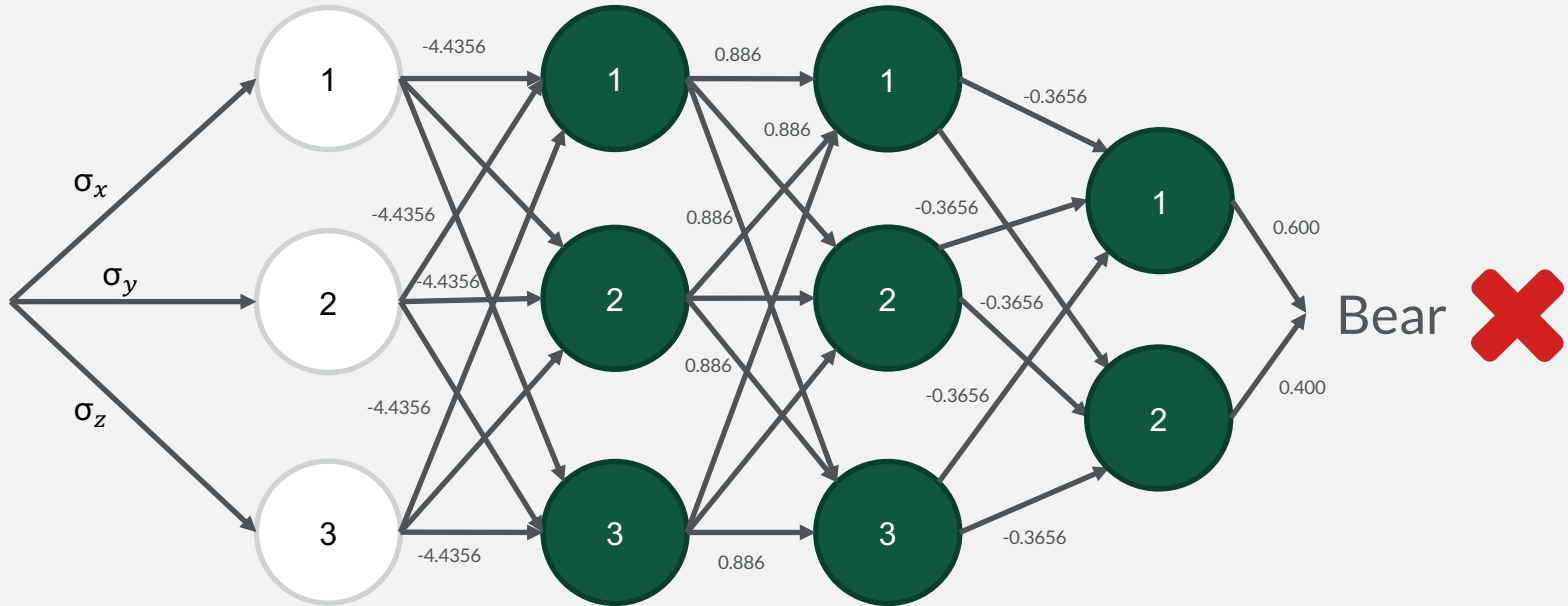
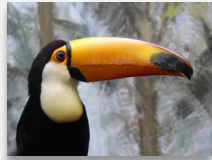
Software 1.0 vs. Software 2.0



Software 1.0

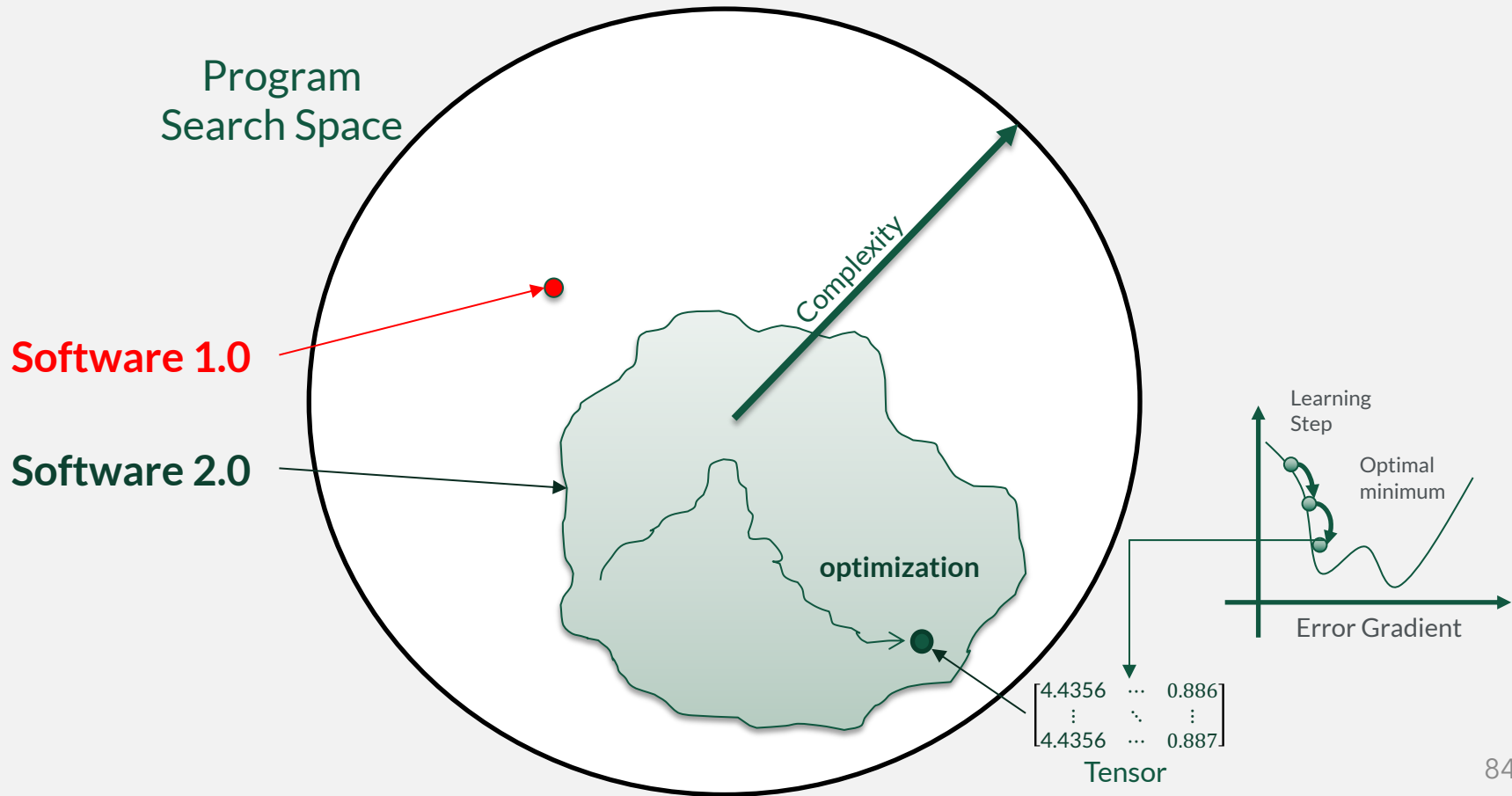
```
1.  /**
2.   * Add element in the list
3.   * @param element to add
4.   * @return true if element added, false otherwise
5.   */
6.  public boolean addElement (Element elem) {
7.      if(myList != null){
8.          myList.add(elem);
9.          return true;
10.     }
11.     return false;
12. }
```

Software 2.0 = DL-based systems



How is Deep Learning Software 2.0?

Optimization by Gradient Descent to Find “The Program”



Real-world DL-based System (Software 2.0)

Config

Data
Collection

Data
Verification

Machine
Resources
Management

Serving
Infrastructure

Feature
Extraction

ML
Code

Analysis
Tools

Process
Management
Tools

Monitoring

Yesterday's Devs vs. Tomorrow's Devs



Machine
Resources
Management

Analysis Tools

Process
Management
Tools

Serving
Infra-
structure

ML
Code

Monitor
ing



Configuration

Data
Collection

Data
Verification

ML Code

Feature
Extraction

Will Deep Learning encompass all software?

Will Deep Learning encompass all software?

Not quite ...

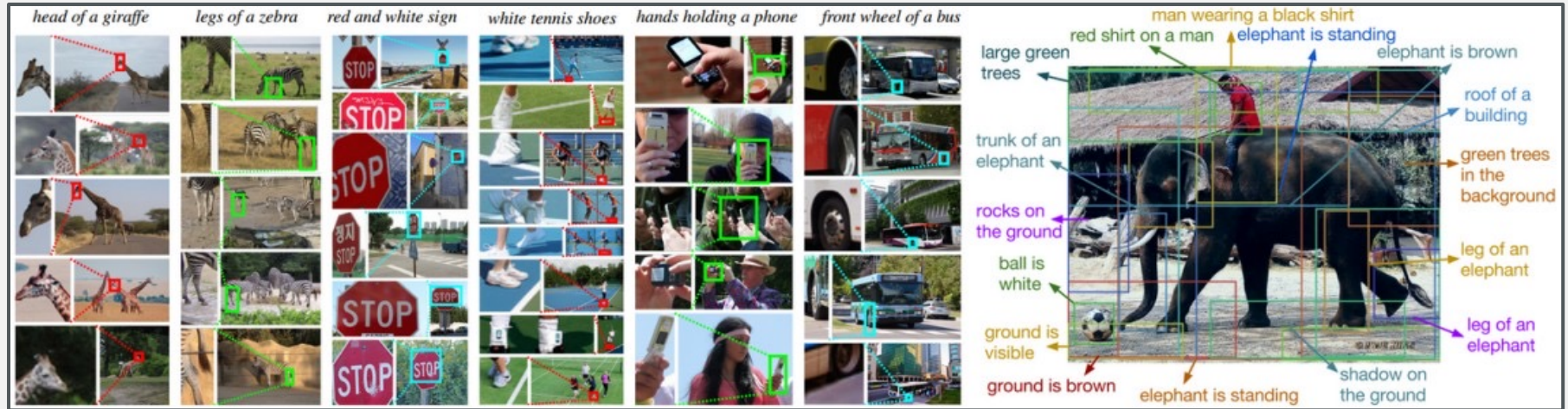
Will Deep Learning encompass all software?

Not quite ...

But the applications of DL are numerous and growing!

The Transition to Software 2.0

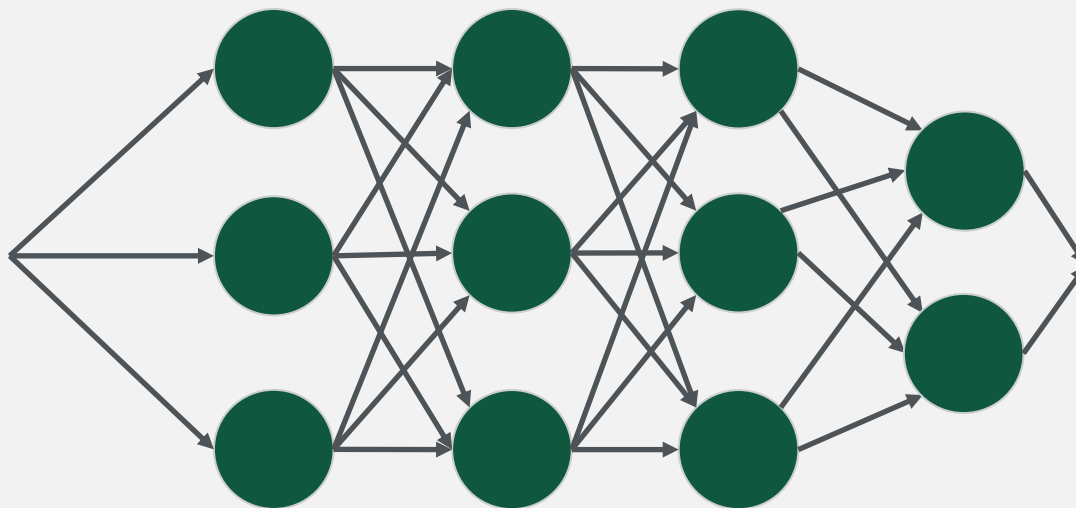
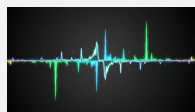
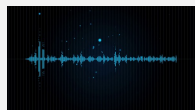
Image Recognition and Understanding



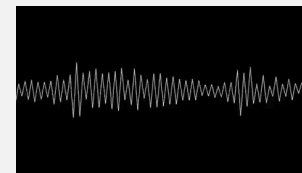
The Transition to Software 2.0

Speech Synthesis

Audio and
Transcription Corpus

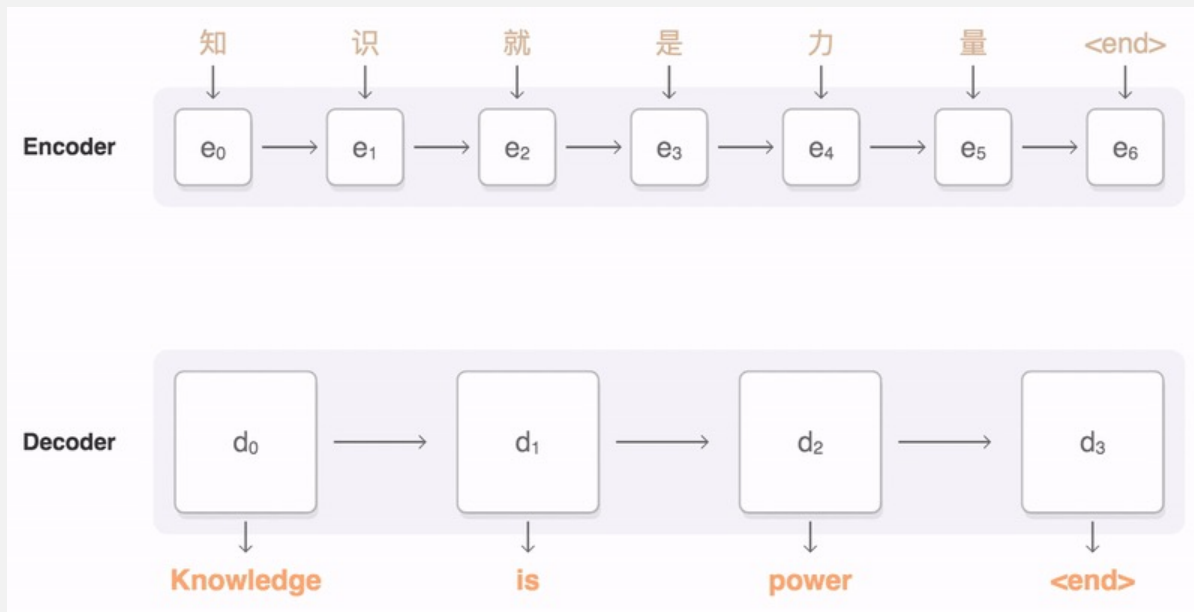


Synthesized Voice



The Transition to Software 2.0

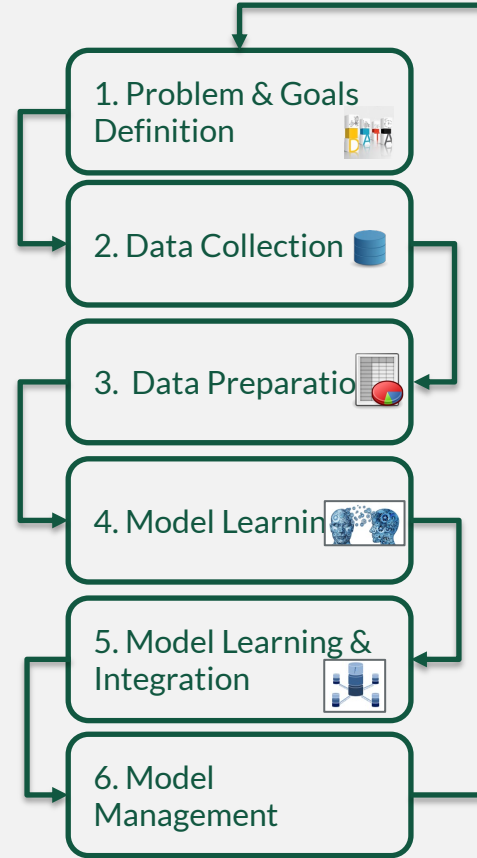
Machine Translation



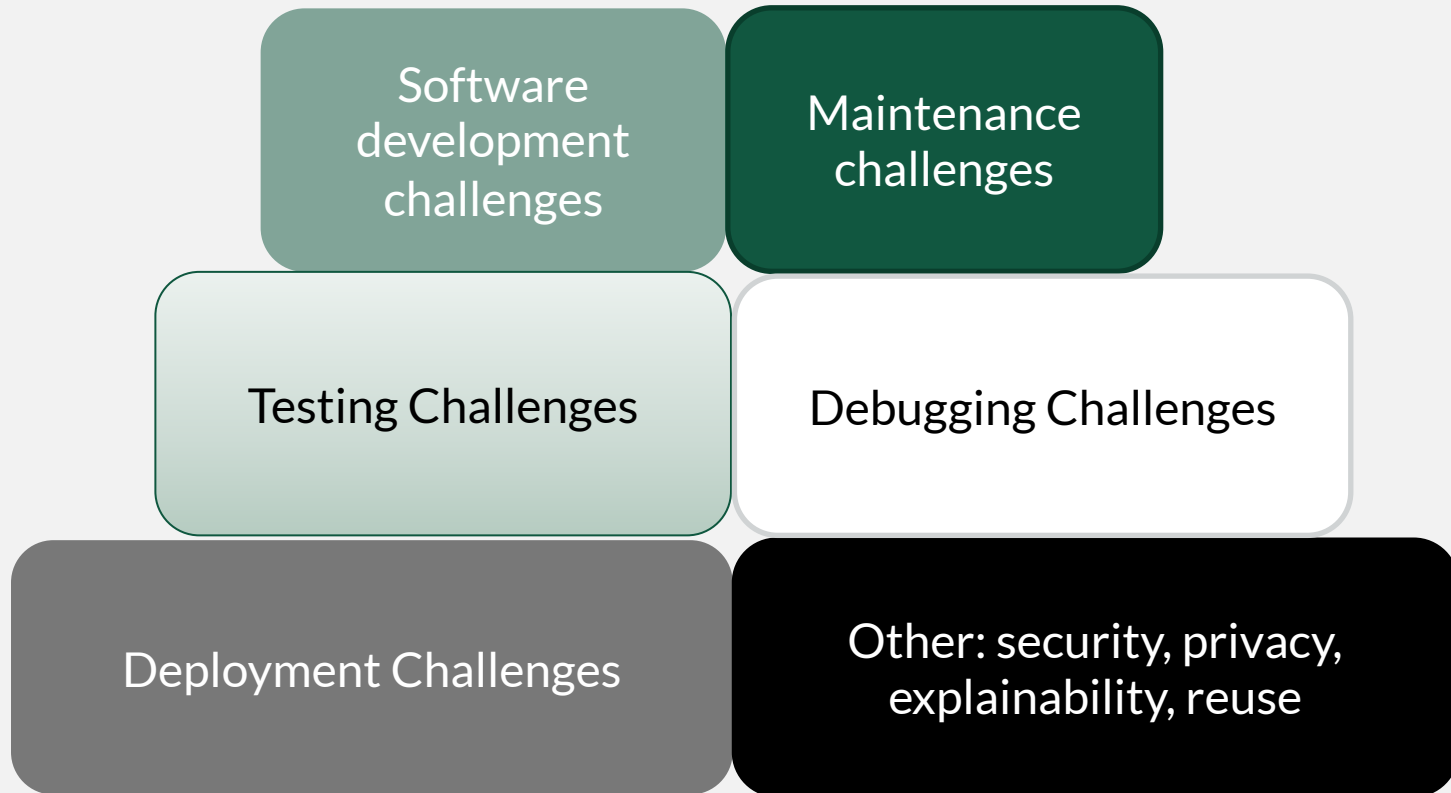
Benefits of Software 2.0

- Computationally homogeneous
- Simple to bake into silicon
- Constant running time
- Constant memory use
- Portable
- Agile
- System is capable of “self-optimization”
- “Better than programmers” (at least on anything involving images/video/sound/speech)

Traditional SE Development vs. DL Development



SE Challenges for Software 2.0 (or SE4DL)



Challenges: Software Development for DL

Deriving
Requirements



Effort
Estimation



Experiment
Management



Data
Labeling

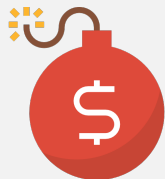


Versioning
Models



Challenges: Software Maintenance for DL

Technical
Debt



Data
Dependencies



Reliance on
Pre-Trained
Models



Experimental
Code Paths



Configuration
Management

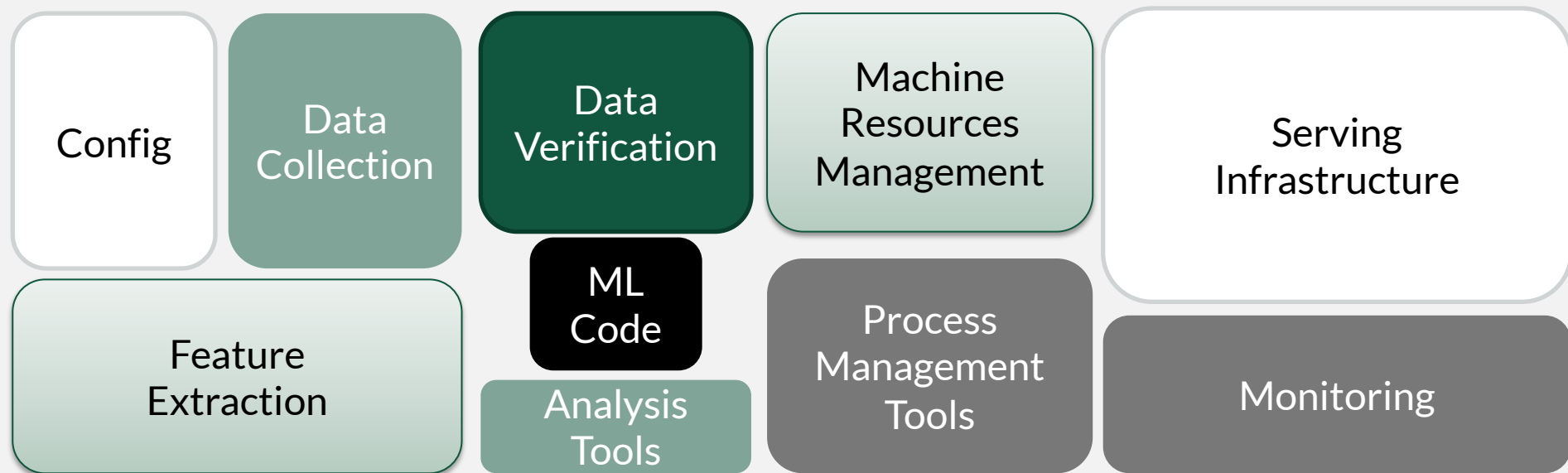


Evolving
Hardware +
Software



Challenges: Software Maintenance for DL

- Code and data technical debt (~95% is glue code)



Challenges: Testing for DL

Testing
Data



Deployment
Testing



Edge
Case
Discovery



Non
Determinism



Performance
Testing



Challenges: Testing for DL

- Data replaces code and should be tested rigorously

Dataset

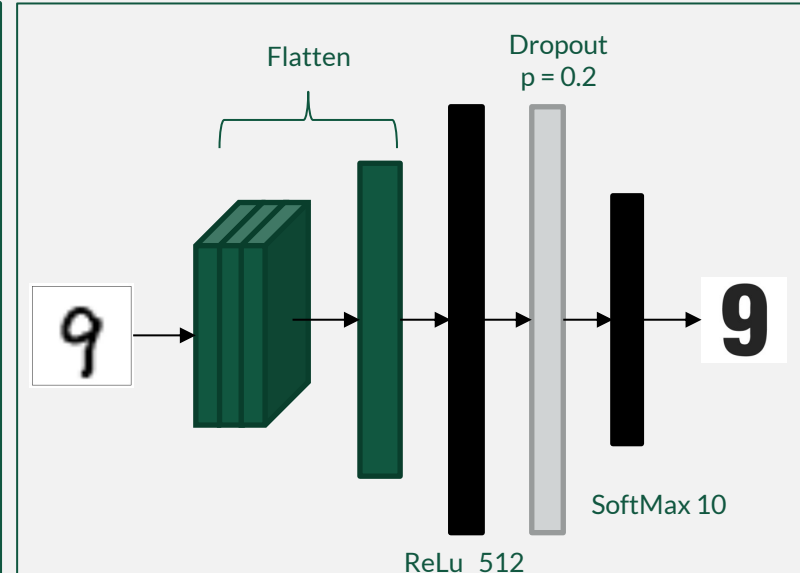
Supervised

Model
Definition

Loss
Function

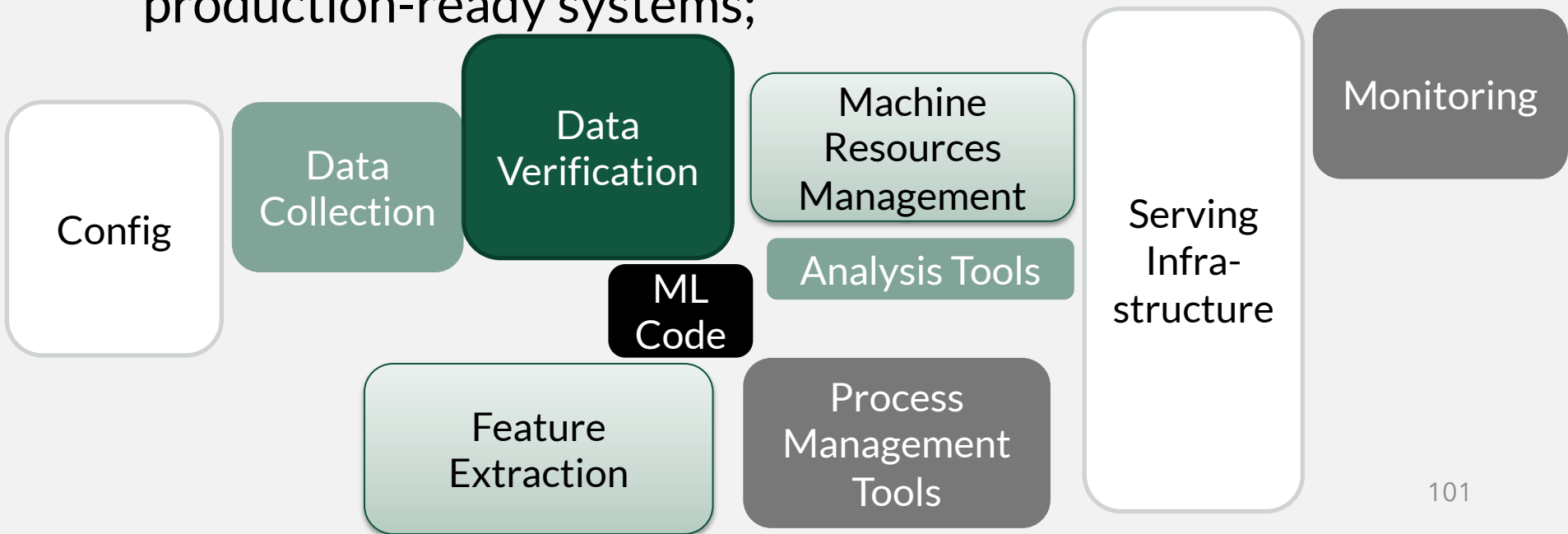
Evaluation

```
1. import tensorflow as tf
2. mnist = tf.keras.datasets.mnist
3.
4. (x_train, y_train),(x_test, y_test) = mnist.load_data()
5. x_train, x_test = x_train / 255.0, x_test / 255.0
6.
7. model = tf.keras.models.Sequential([
8.     tf.keras.layers.Flatten(),
9.     tf.keras.layers.Dense(512, activation=tf.nn.relu),
10.    tf.keras.layers.Dropout(0.2),
11.    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
12. ])
13.
14. model.compile(optimizer='adam',
15.               loss='sparse_categorical_crossentropy',
16.               metrics=['accuracy'])
17.
18. model.fit(x_train, y_train, epochs=5)
19. model.evaluate(x_test, y_test)
```



Challenges: Testing for DL

- Data replaces code and should be tested rigorously;
- We need to test not only the models, but also production-ready systems;



Challenges: Debugging for DL

Requires
Trained
Model



“Traditional”
Debuggers
Don’t Apply



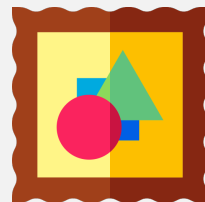
Lazy
Execution



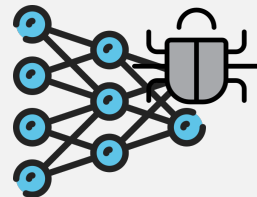
Bugs in
Dataset



Bugs can
be Abstract

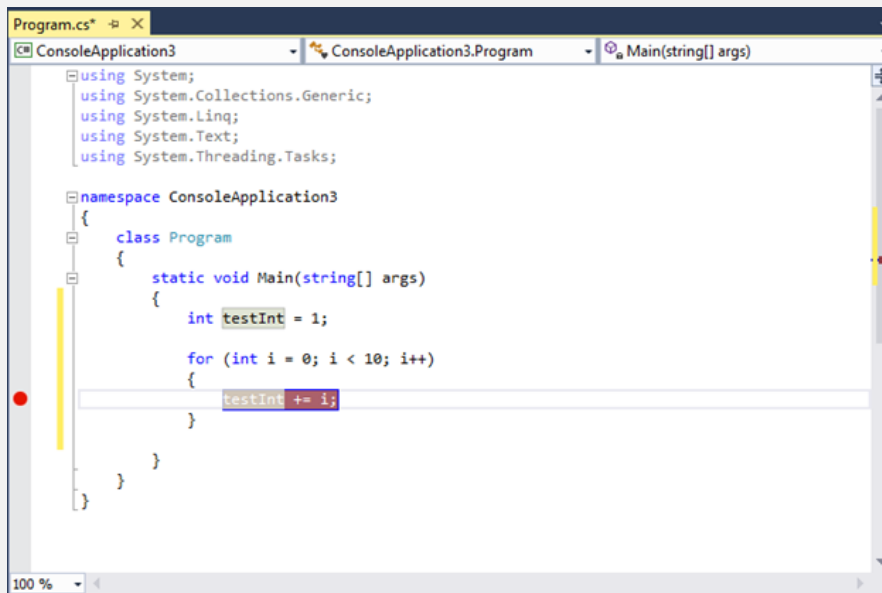


DNN
Bugs



Challenges: Debugging for DL

- We can not estimate the results (and debug the model) until the model is trained
- Traditional debugging works in software 1.0



The image shows a screenshot of a Visual Studio code editor window. The title bar indicates the file is 'Program.cs'. The editor displays the following C# code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
            int testInt = 1;

            for (int i = 0; i < 10; i++)
            {
                testInt += i;
            }
        }
    }
}
```

A red dot on the left margin indicates a debugger breakpoint is set on the line `testInt += i;`. The code is otherwise standard C# for a simple loop that calculates the sum of integers from 0 to 9.

Challenges: Debugging for DL

- We can not estimate the results (and debug the model) until the model is trained
- Traditional debugging does not work in software 2.0

Dataset

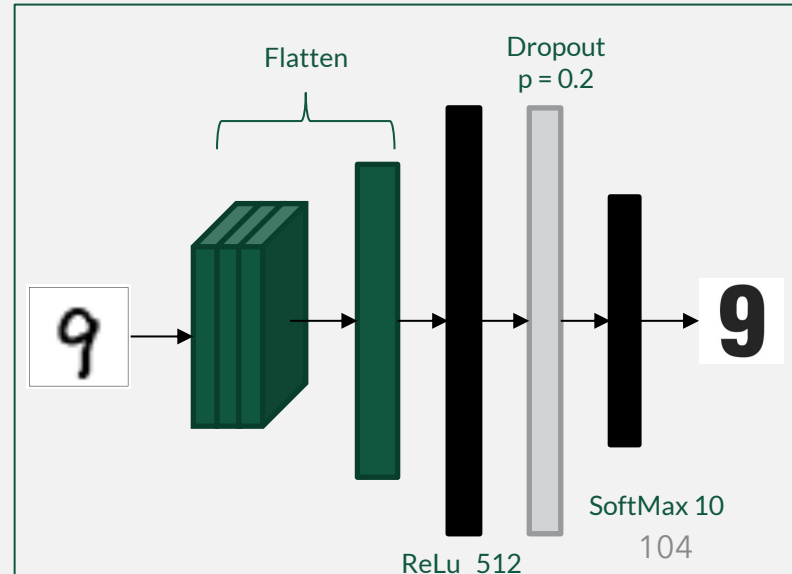
```
1. import tensorflow as tf
2. mnist = tf.keras.datasets.mnist
3.
4. (x_train, y_train),(x_test, y_test) = mnist.load_data()
5. x_train, x_test = x_train / 255.0, x_test / 255.0
6.
7. model = tf.keras.models.Sequential([
8.     tf.keras.layers.Flatten(),
9.     tf.keras.layers.Dense(512, activation=tf.nn.relu),
10.    tf.keras.layers.Dropout(0.2),
11.    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
12. ])
13.
14. model.compile(optimizer='adam',
15.               loss='sparse_categorical_crossentropy',
16.               metrics=['accuracy'])
17.
18. model.fit(x_train, y_train, epochs=5)
19. model.evaluate(x_test, y_test)
```

Supervised

Model
Definition

Loss
Function

Evaluation

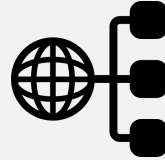


Challenges: DL Deployment

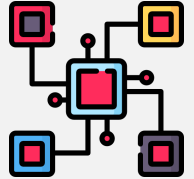
Feedback
Loops



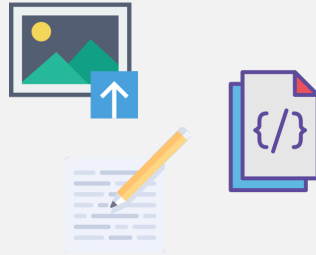
Stream
Processing



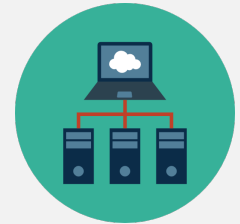
Distributed
DL



Data
Modalities



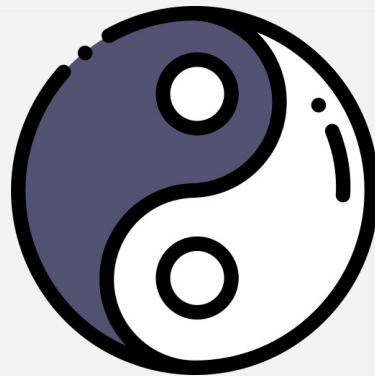
Data
Formatting



What are the Next Steps?

There is still a lot of work to be done!

SE Research



ML/DL Research

Acknowledgements – DL4SE Survey



Cody Watson



David Nader Palacio



Nathan Cooper



Denys Poshyvanyk

Acknowledgements – DLSE Workshop

Co-Chairs



Denys Poshyvanyk



Baishakhi Ray

Steering Committee



Prem Devanbu



Matthew Dwyer



Michael Lowry



Xiangyu Zhang



Rishabh Singh



Sebastian Elbaum