CEN 5016: Software Engineering

Spring 2024

University of Central Florida

Dr. Kevin Moran

Week 7 - Class II: Introduction to Software Engineering Research

Administrivia

- *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

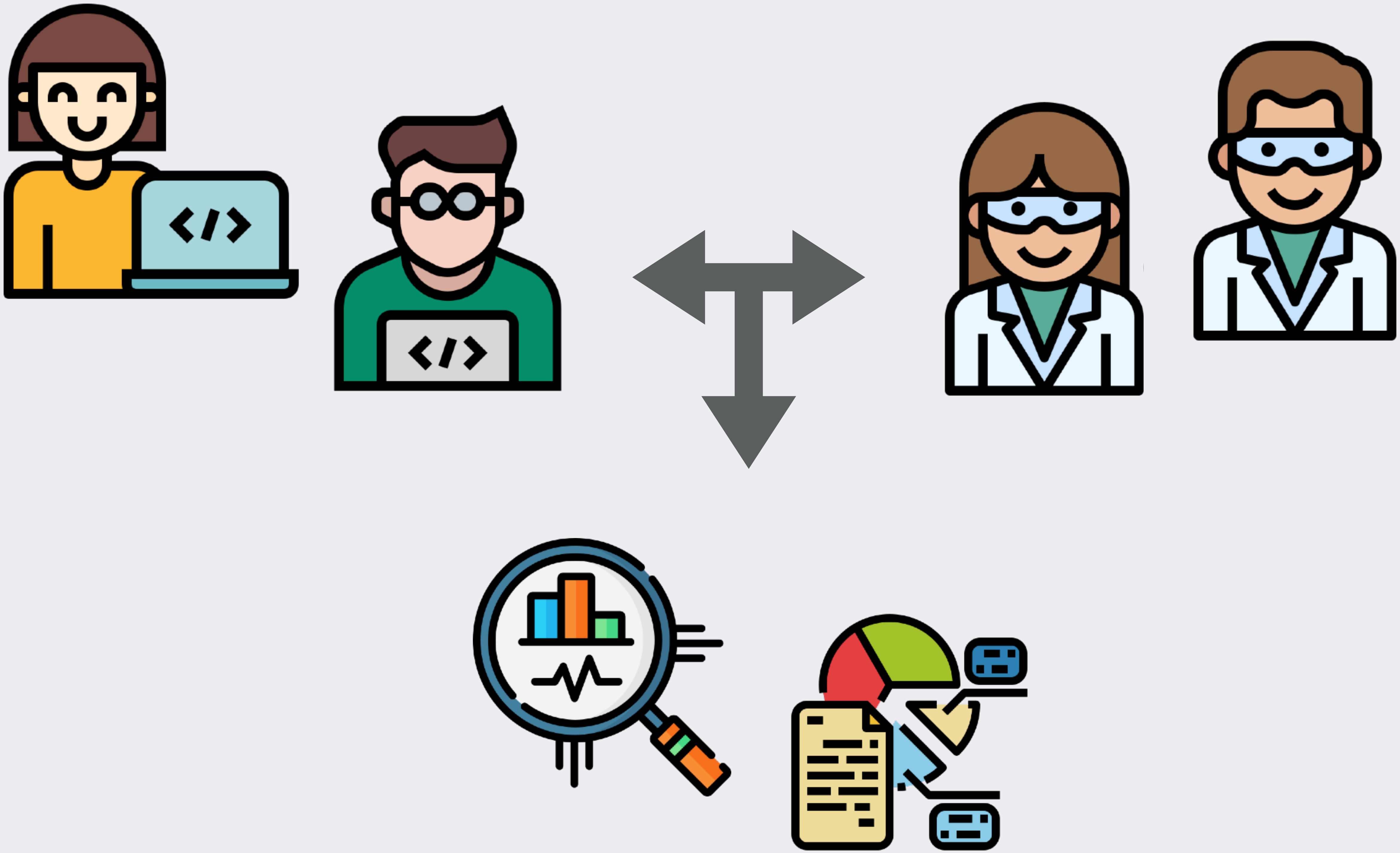
Blend *scientific discovery* with *practical significance*

Practical Significance

Blend *scientific discovery* with *practical significance*

How Can We Design Practical Automation?

Understanding Developer Needs



Mining Software Repositories

GitHub

Bitbucket

Google Play

SOURCEFORGE

GitLab

Mining Software Repositories

Mining Software Repositories

Source Code Files

Software Documentation

Screenshots

Screen Recordings

Bug Reports

Design Documents

Learning Patterns from Software Data

Source Code

Bug Reports

Screenshots

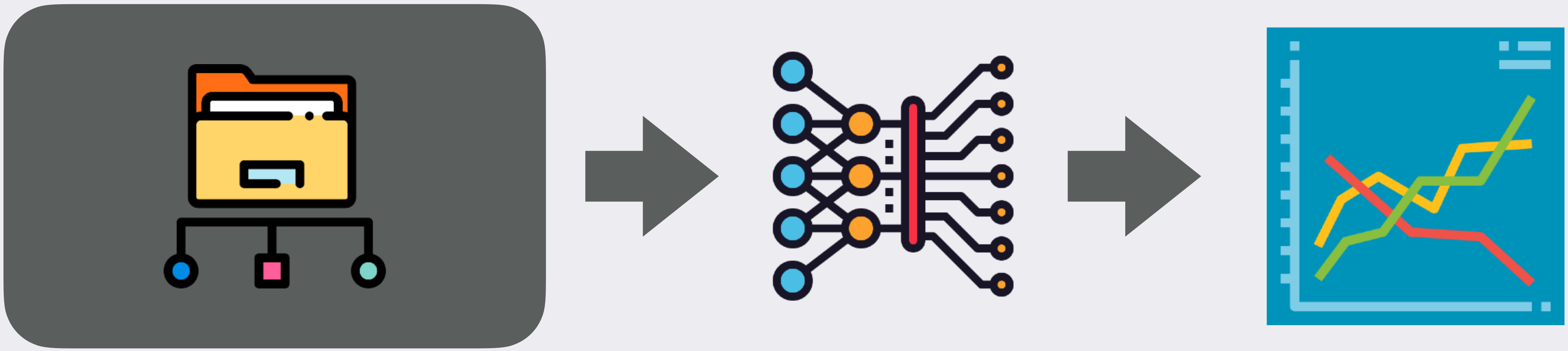
Recordings

Documentation

Learning Patterns from Software Data

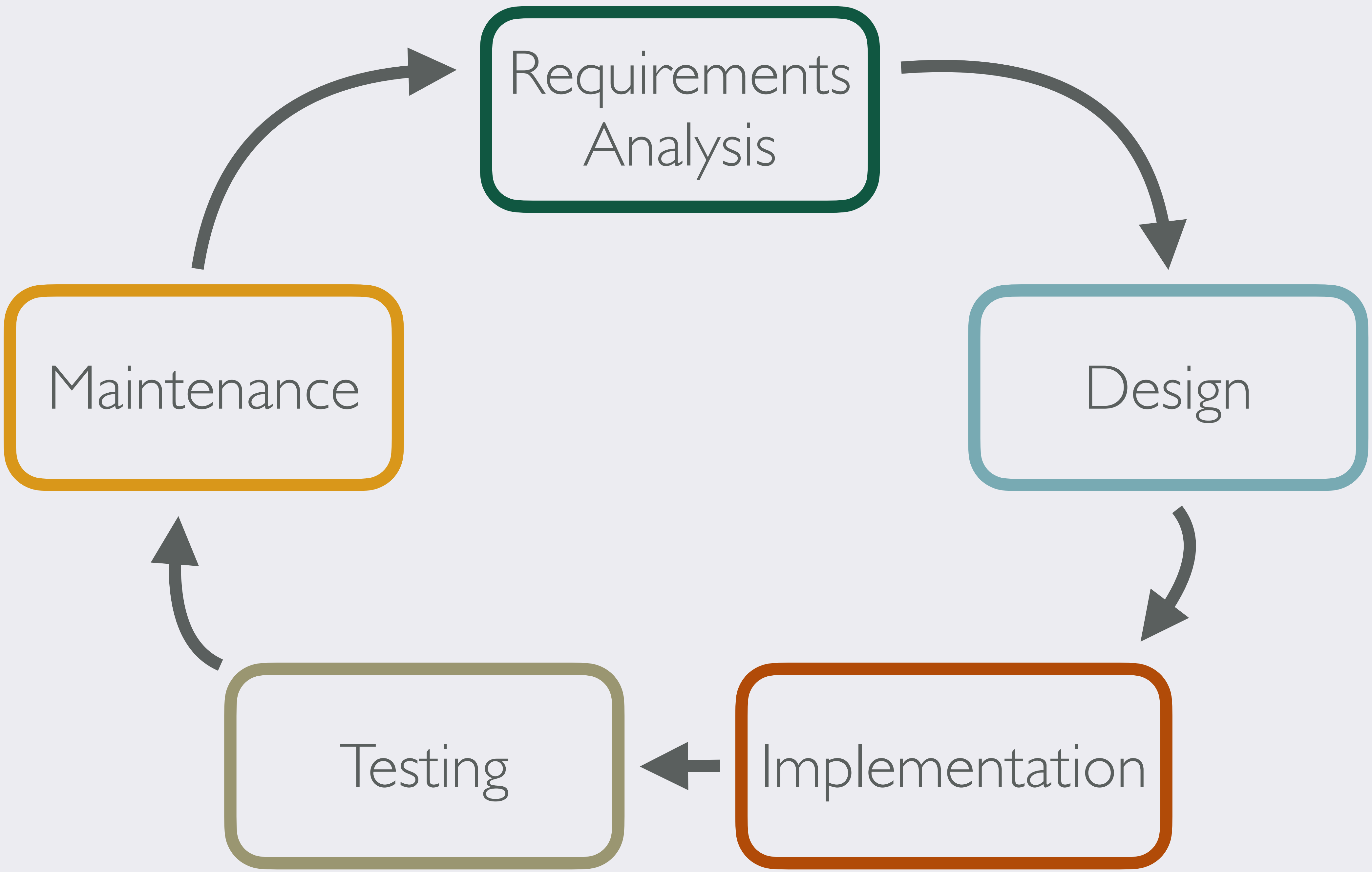
Software Repository Data

Salient Patterns



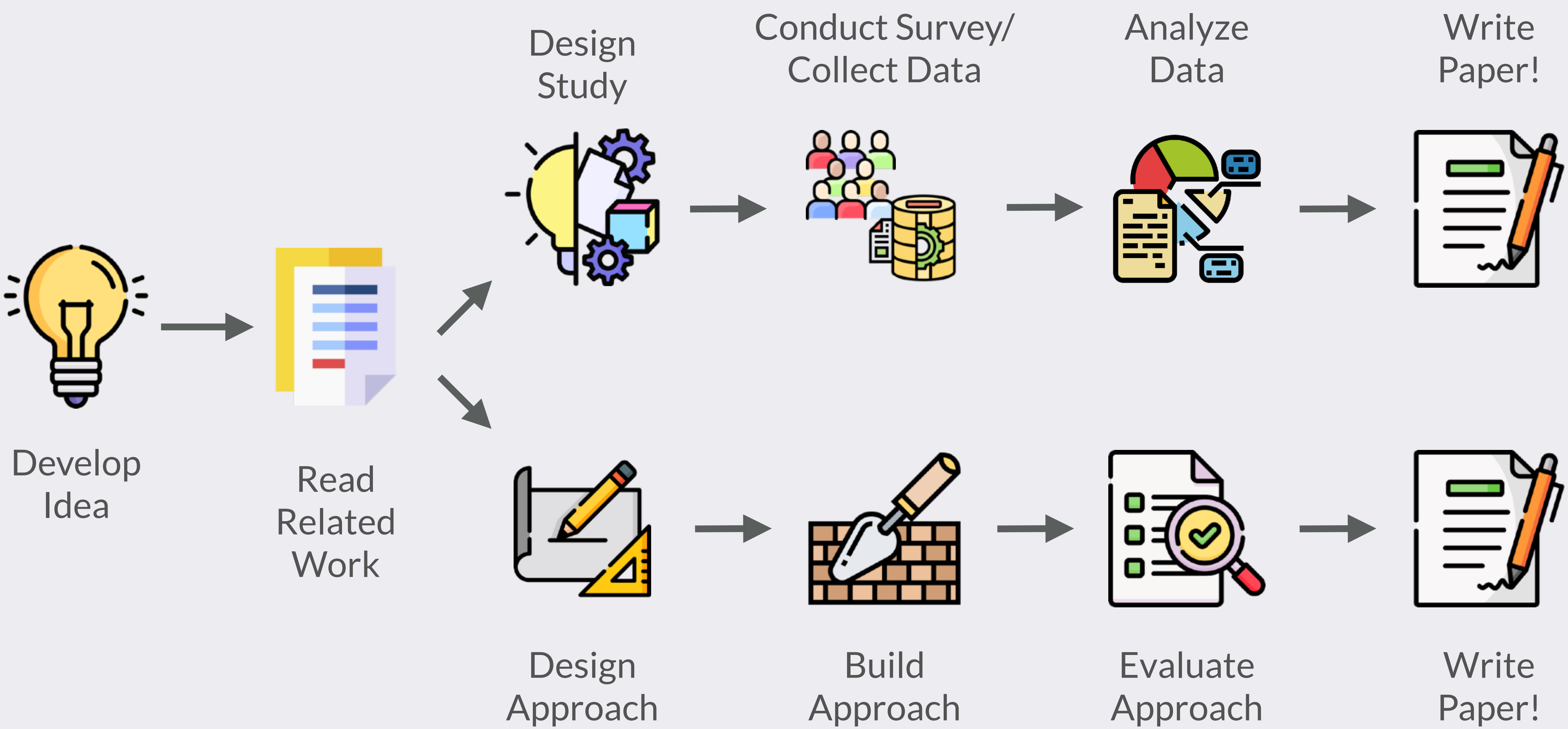
Machine Learning

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

UCF

SAGE

GitHub Copilot

Sign up >

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

```
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";
```

Talk Outline

- **Topic 1 - Background:** The Evolution of Machine Learning (ML) to Deep Learning (DL)

- **Topic 2- DL4SE:** The Current State of Research

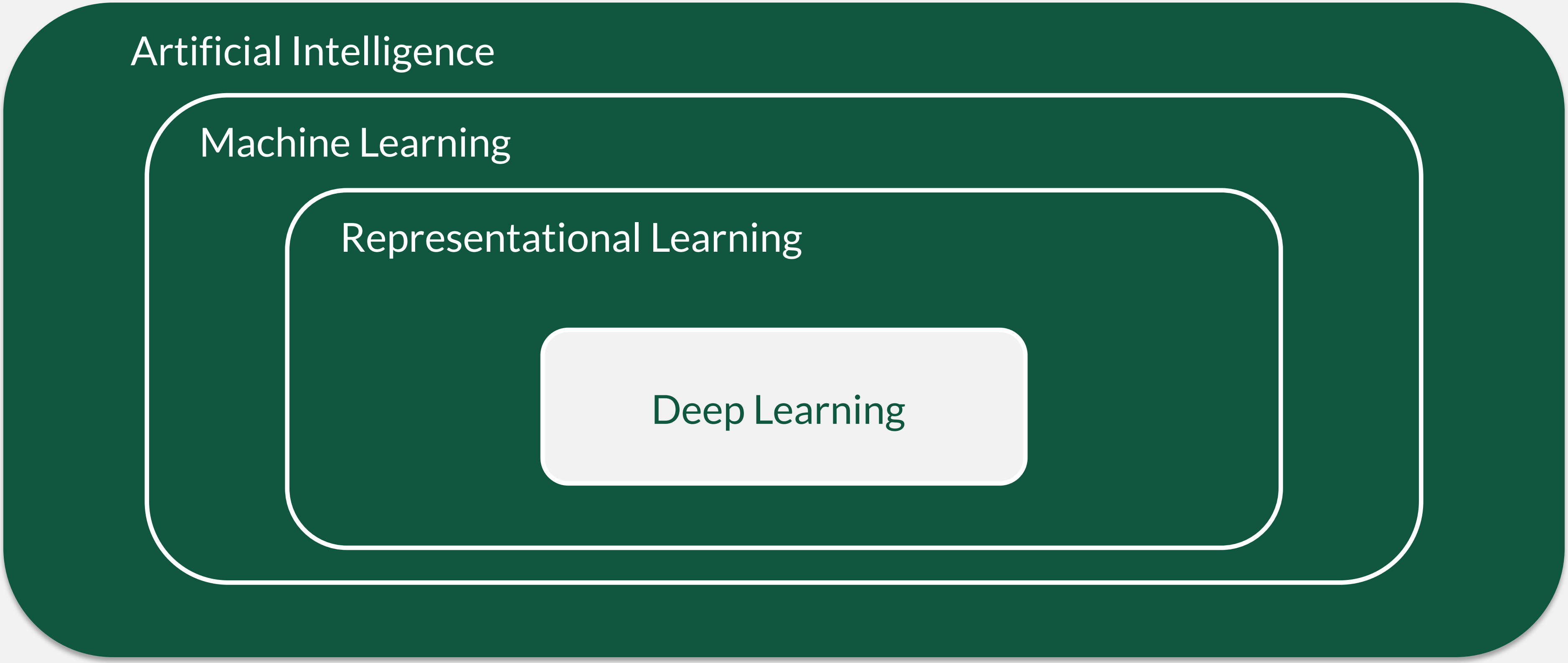
- **Topic 3 – Looking Forward:** Future Directions and Paths Forward

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

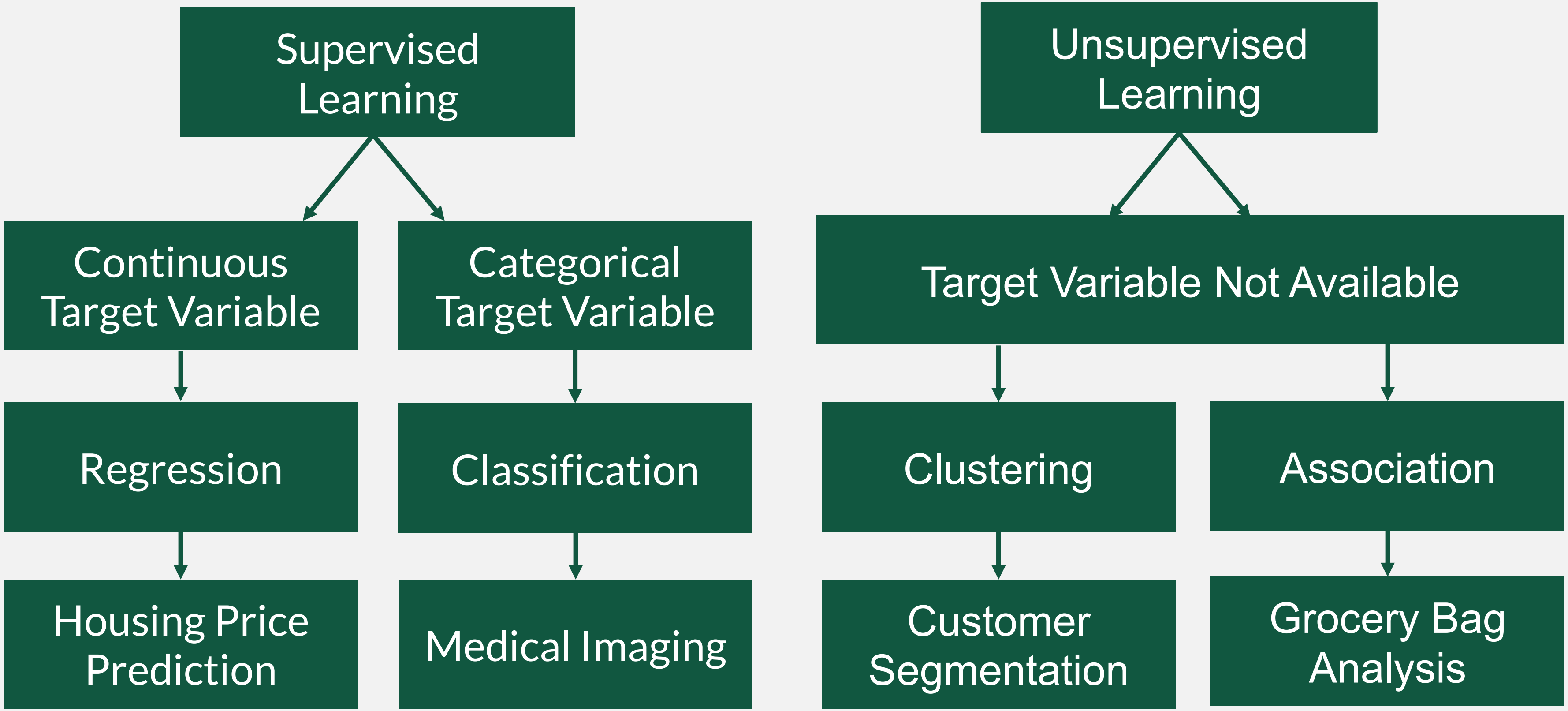
What is Machine Learning?

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

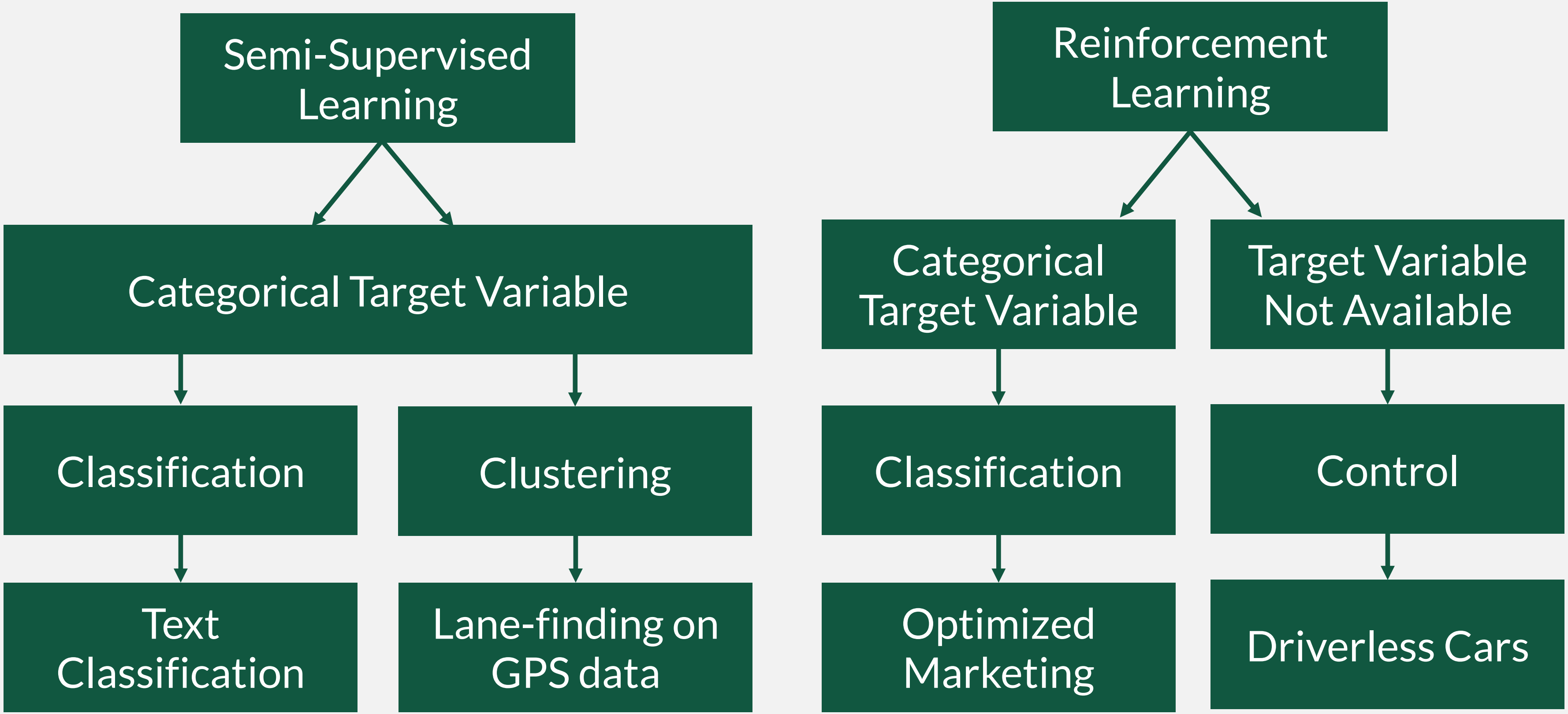
The Hierarchy of Artificial Intelligence



Machine Learning Taxonomy



Machine Learning Taxonomy

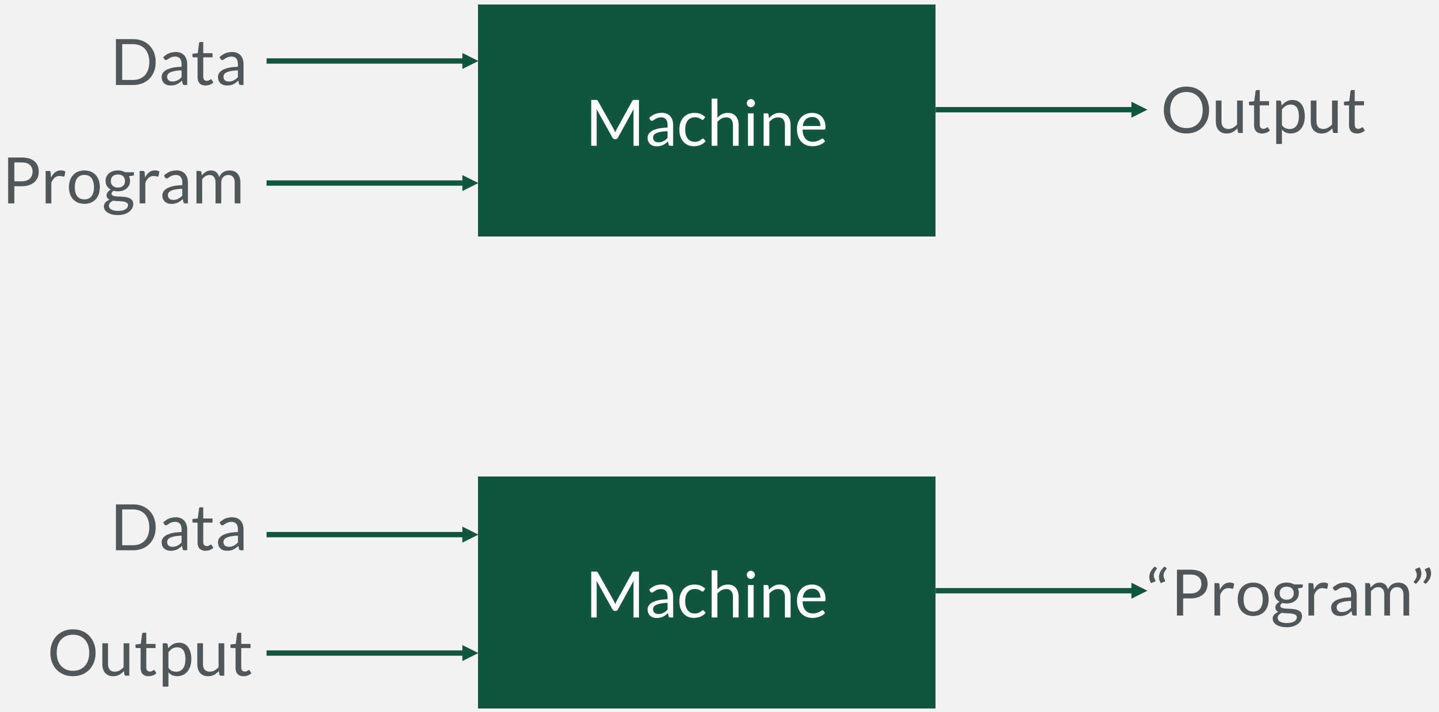


ML Representations

Supervised Learning	Unsupervised Learning	Semi-Supervised Learning	Reinforcement Learning
K Nearest Neighbor	K-means clustering	Self-Training of Existing Classifiers	Q-Learning
Naïve Bayes	Association rule learning	Hidden Markov Models	Temporal Difference
Decision Trees		Multiple Gaussian Distributions	
Linear Regression		Semi-supervised support vector machines	
Support Vector Machine			

Canonical Representation

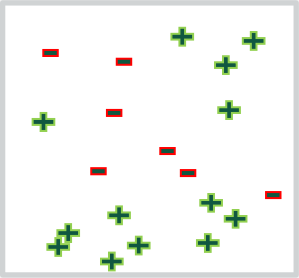
Machine Learning vs. Traditional Programming



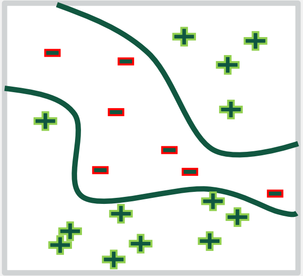
When do We Need Machine Learning?

Three Conditions:

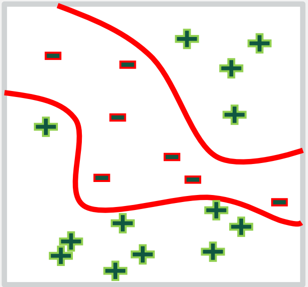
1. We have an Existing Dataset
2. A pattern exists in the data
3. The pattern is not *easily* defined by an equation



The Data

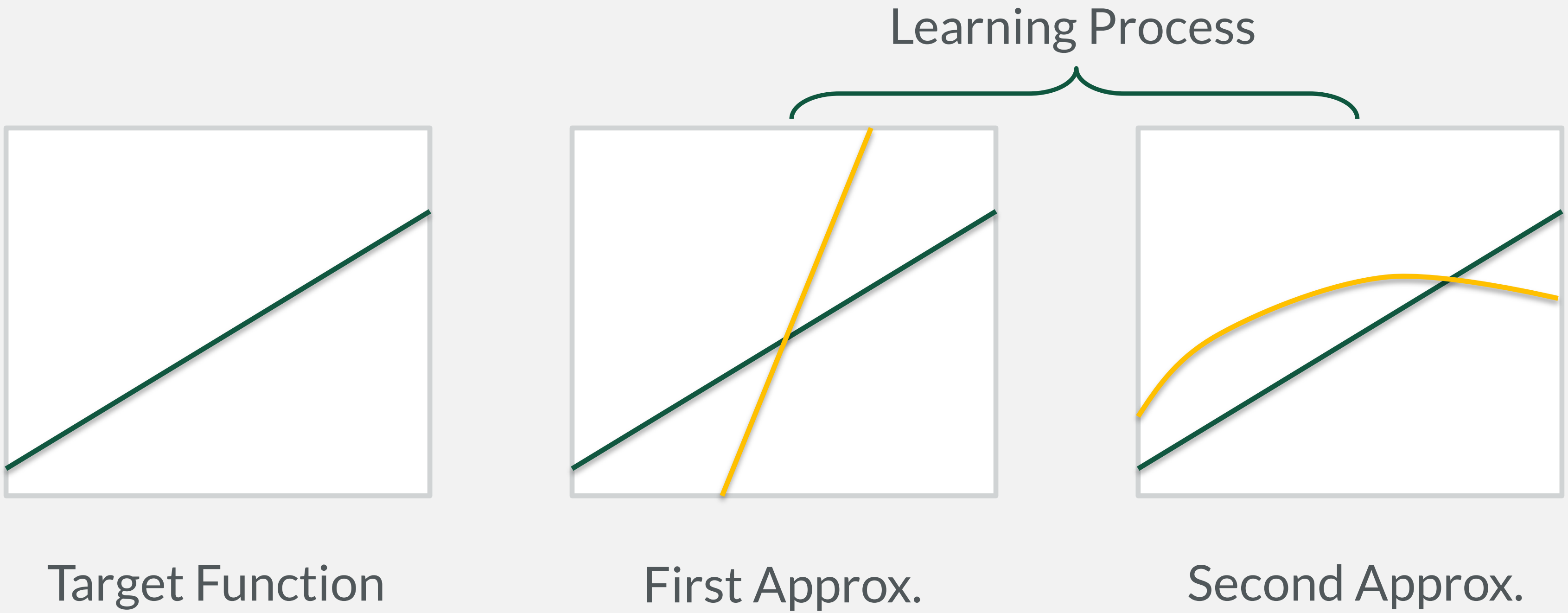


The Pattern



No Possible Equation

The Computational Learning Process



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*).

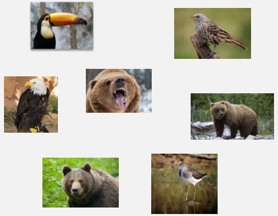
The Five Elements of the Learning Process

Input

Output

`{Bird, Bear}`

Data



Target Function

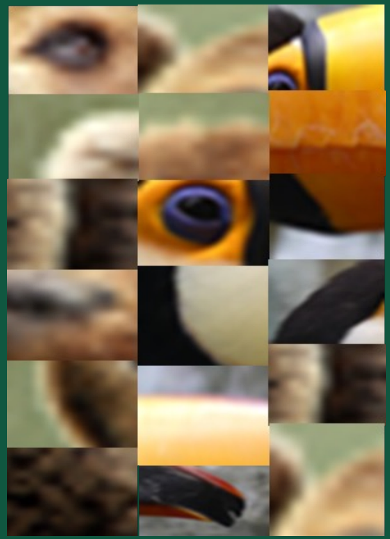
$F(x)$

Hypothesis

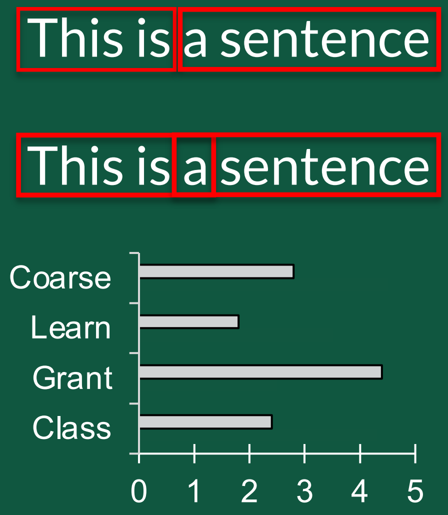
$G(x) \approx F(x)$

Feature Engineering for "Canonical" Machine Learning

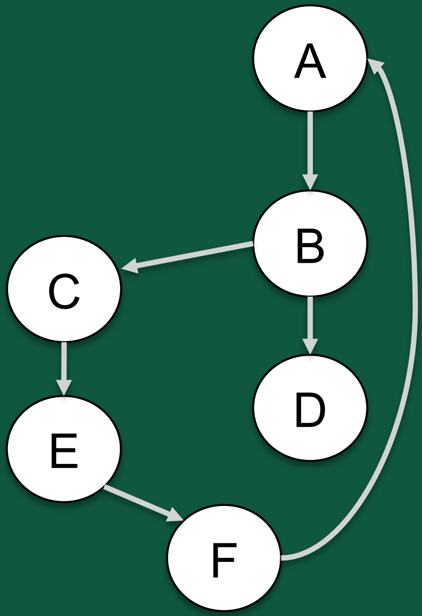
Images



Text



Source Code



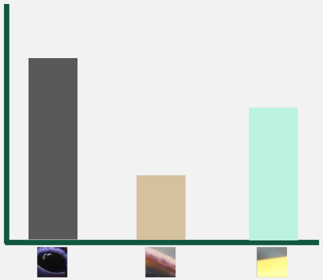
“Canonical” ML Image Classification

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

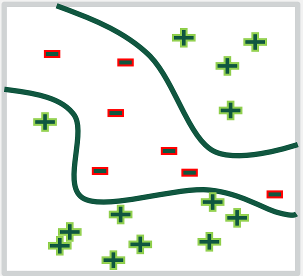
Canonical ML techniques have only been able to achieve ***~ 60% accuracy***

Shortcomings of Traditional ML Techniques

Manually Derived Features



Complex Kernels



Shallow Representation

1	0	1	1	0	1	1	0	1

The Advent of Deep Learning

Input

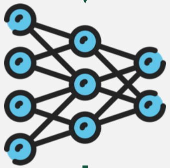
1	0	1	1	0	1	1	0	1

Feature Mapping

Output → Bird

“Canonical” Machine Learning

Input



→ Rich Hierarchical Feature “Embedding”

Feature Mapping

Output → Bear

Deep Learning

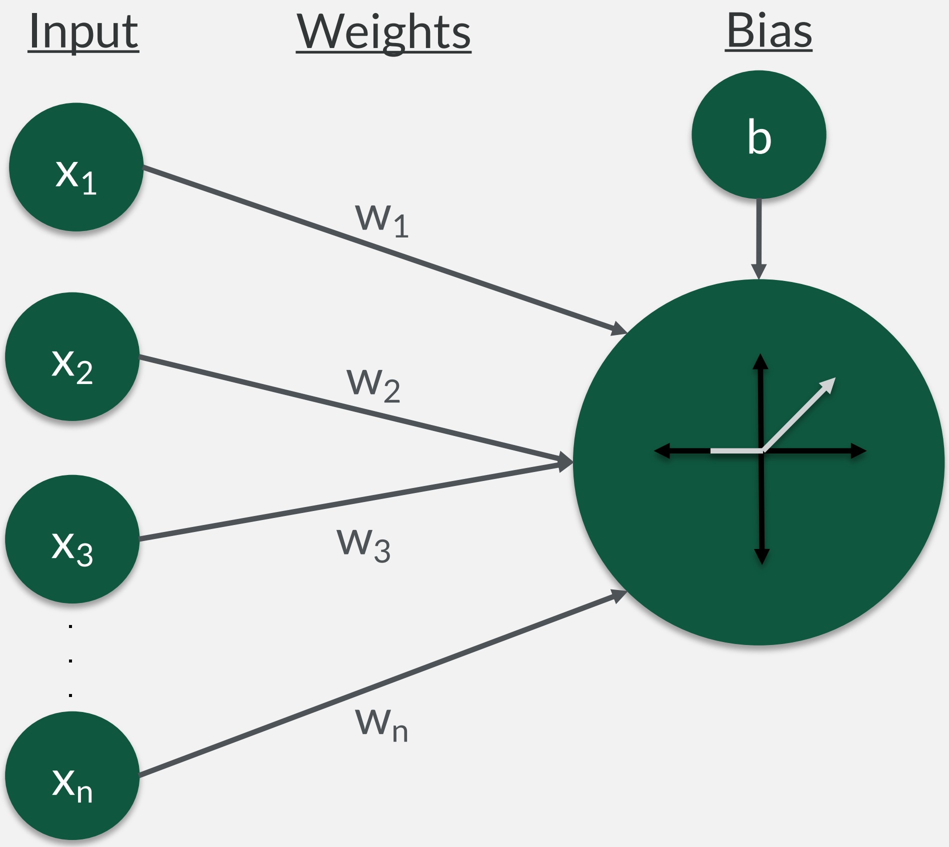
ML Representations

Supervised Learning	Unsupervised Learning	Semi-Supervised Learning	Reinforcement Learning
K Nearest Neighbor	K-means clustering	Self-Training of Existing Classifiers	Q-Learning
Naïve Bayes	Association rule learning	Hidden Markov Models	Temporal Difference
Decision Trees	Autoencoders	Multiple Gaussian Distributions	Deep Adversarial Networks
Linear Regression	Deep Belief Networks	Semi-supervised support vector machines	Deep Q-Learning
Support Vector Machine	Generative Adversarial Networks (GANs)	Neural networks	
Neural Networks (Convolutional, Recurrent, Feed-forward, etc.		Autoencoders	

Canonical Representation

Deep Representation

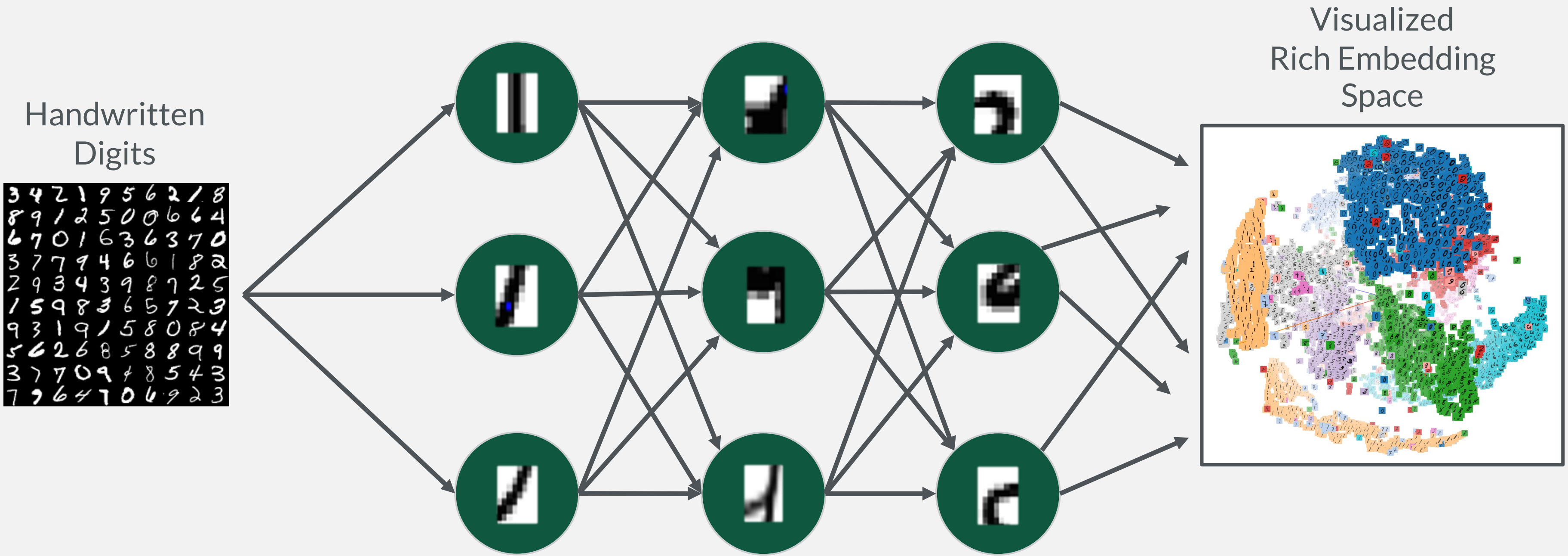
Neurons: The Building Blocks of Rich Features



Additional Activation Functions

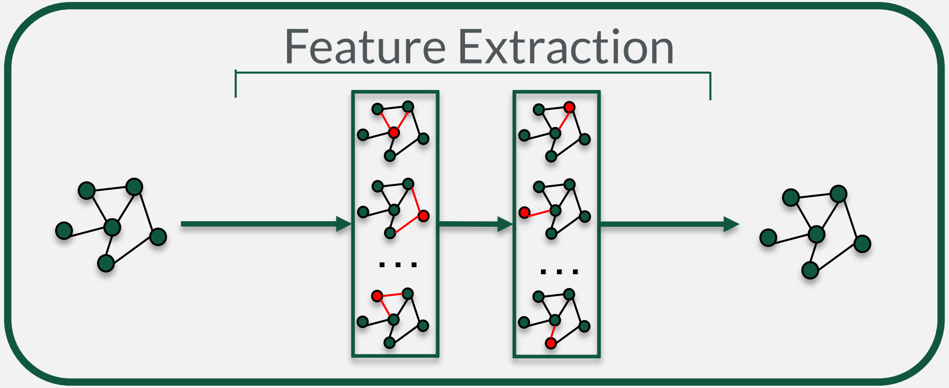
- Identity
- Binary Step
- Sigmoid
- Tanh
- Leaky ReLU
- Softmax

Neural “Networks” for Rich Embeddings

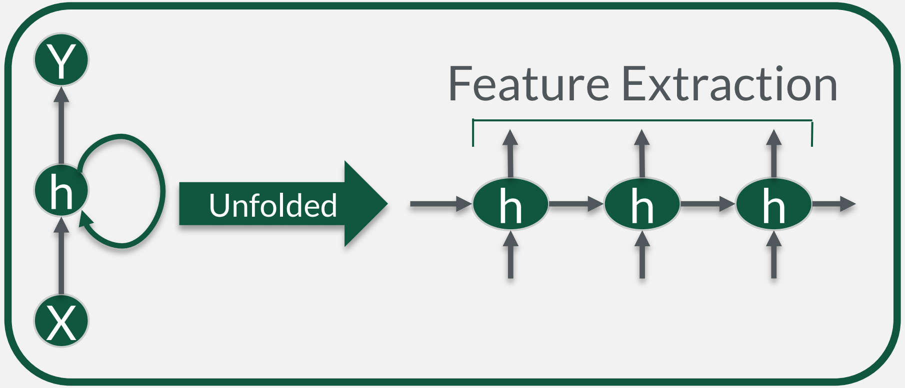


Automated Feature Discovery

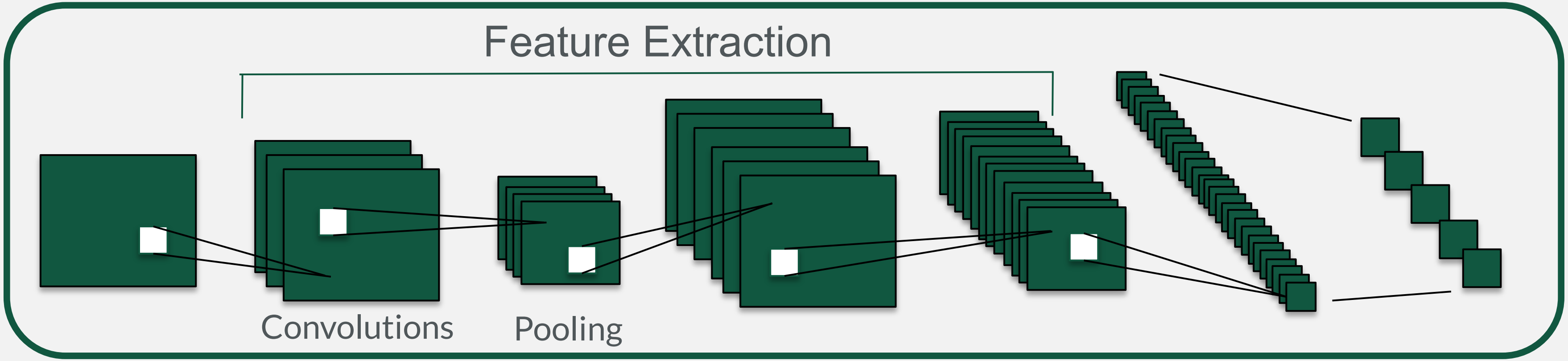
Code



Text (Natural Language)



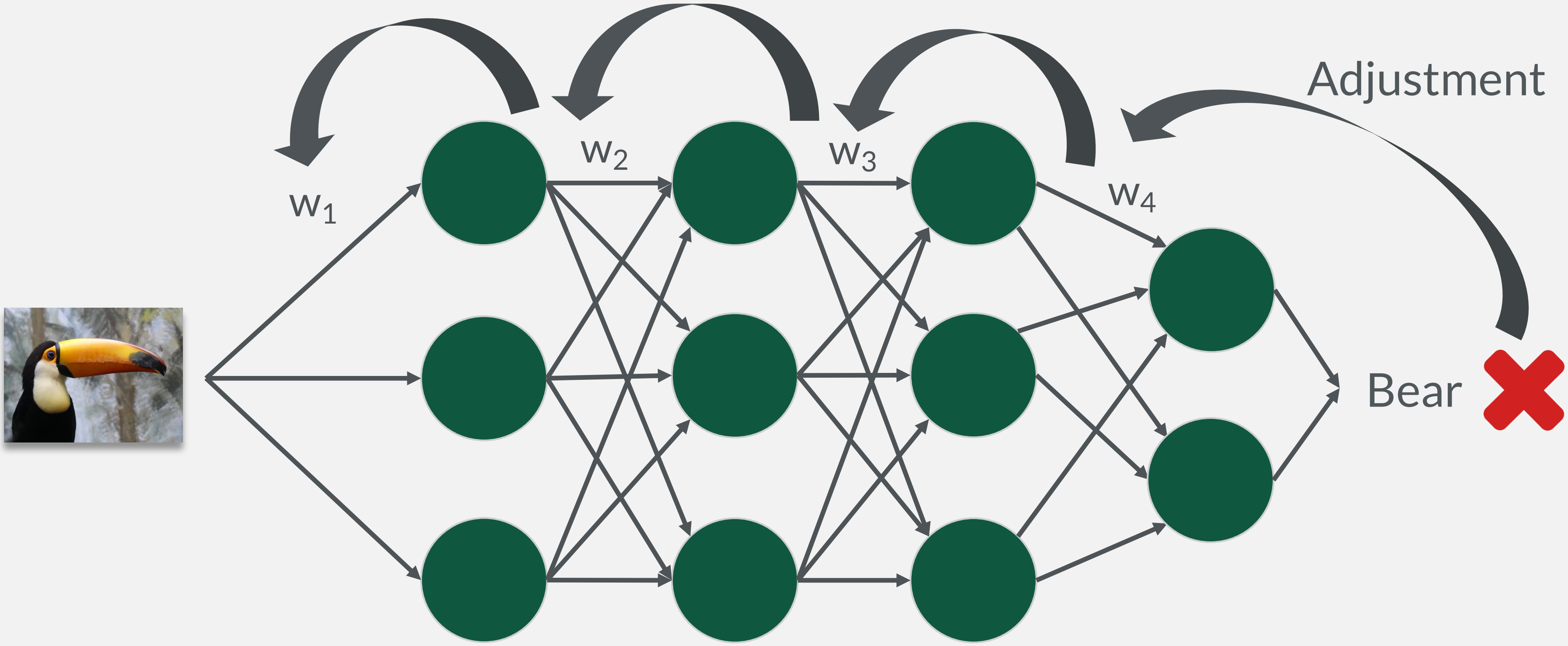
Images



How Can a Model Learn from Deep Embeddings?

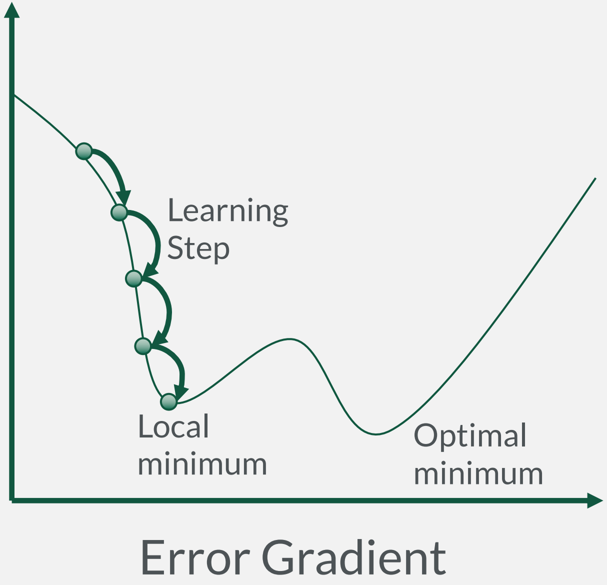
Adjust the Neuron “***weights***” according to ***errors*** made on a given task.

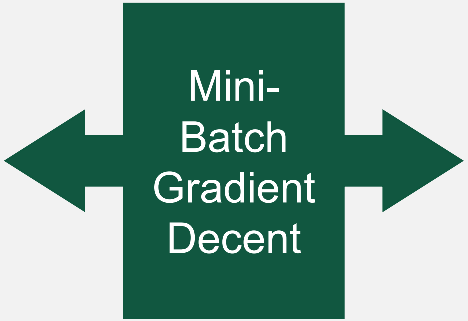
How Can a Model Learn from Deep Embeddings?



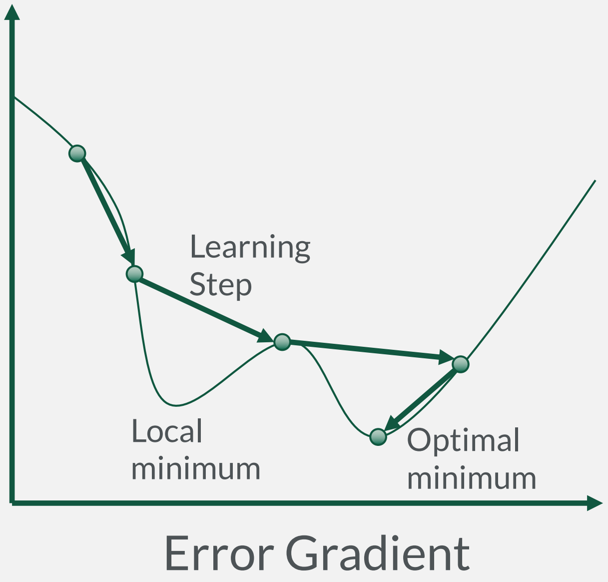
How Should the Weights be Updated?

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, Hierarchal 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 difficult to interpret (Black Box)

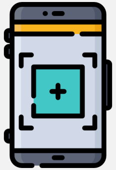
Topic 2 – DL4SE: The Current State of Research

Mining Software Repositories

Automation in Software Engineering Research


Source Code Files


Software Documentation


Screenshots

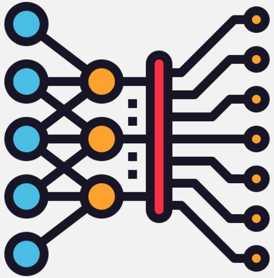

Screen Recordings


Bug Reports


Design Documents

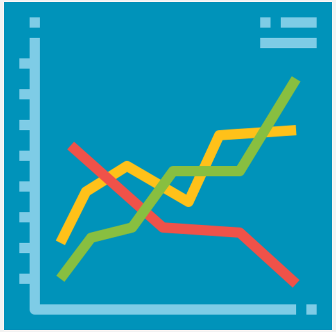
Automation in Software Engineering Research

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*;

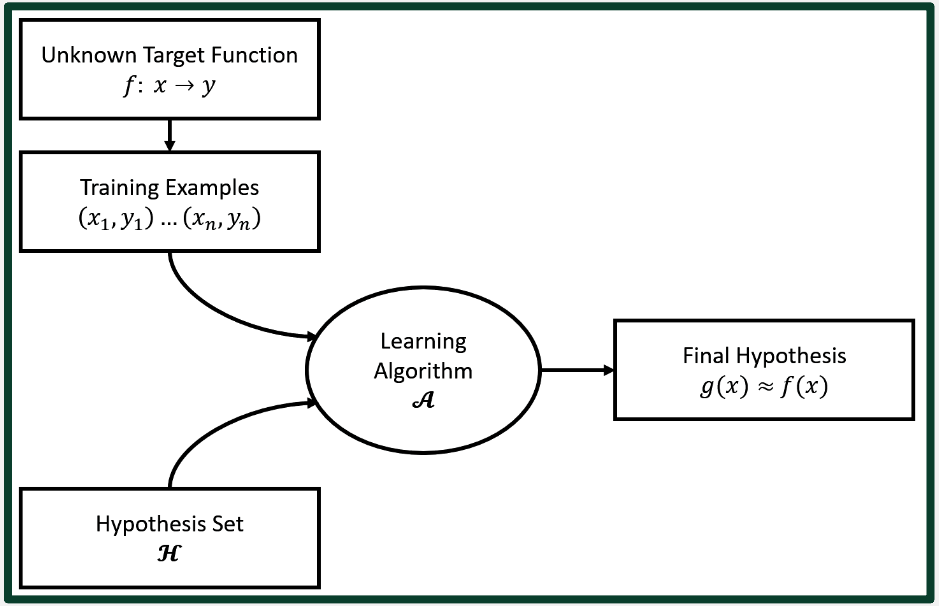
Additional Key Words and Phrases: deep learning, neural networks, literature review, software engineering, machine learning

Systematic Literature Review

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

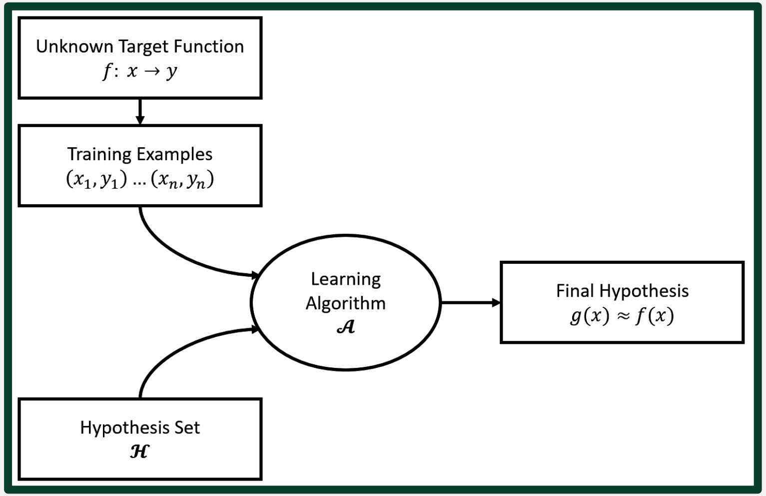
Systematic Literature Review

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



Systematic Literature Review

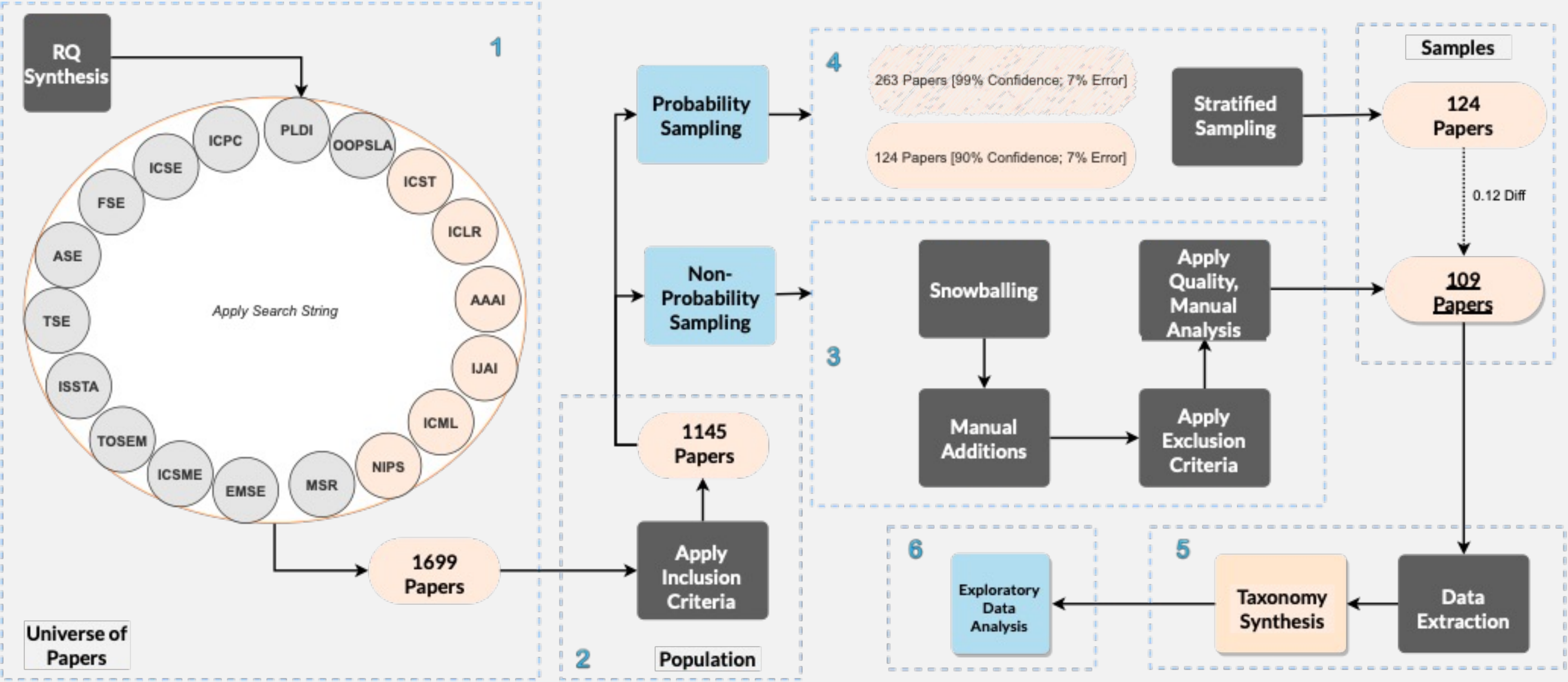
- **RQ_1**: Target Function (*SE Task*)
- **RQ_2**: Data (*Training/Testing Data*)
- **RQ_3**: Learning Model (*Algorithm + Hypothesis Set*)
- **RQ_4**: Final Hypothesis (*Results*)
- **RQ_5**: Reproducibility and Replicability



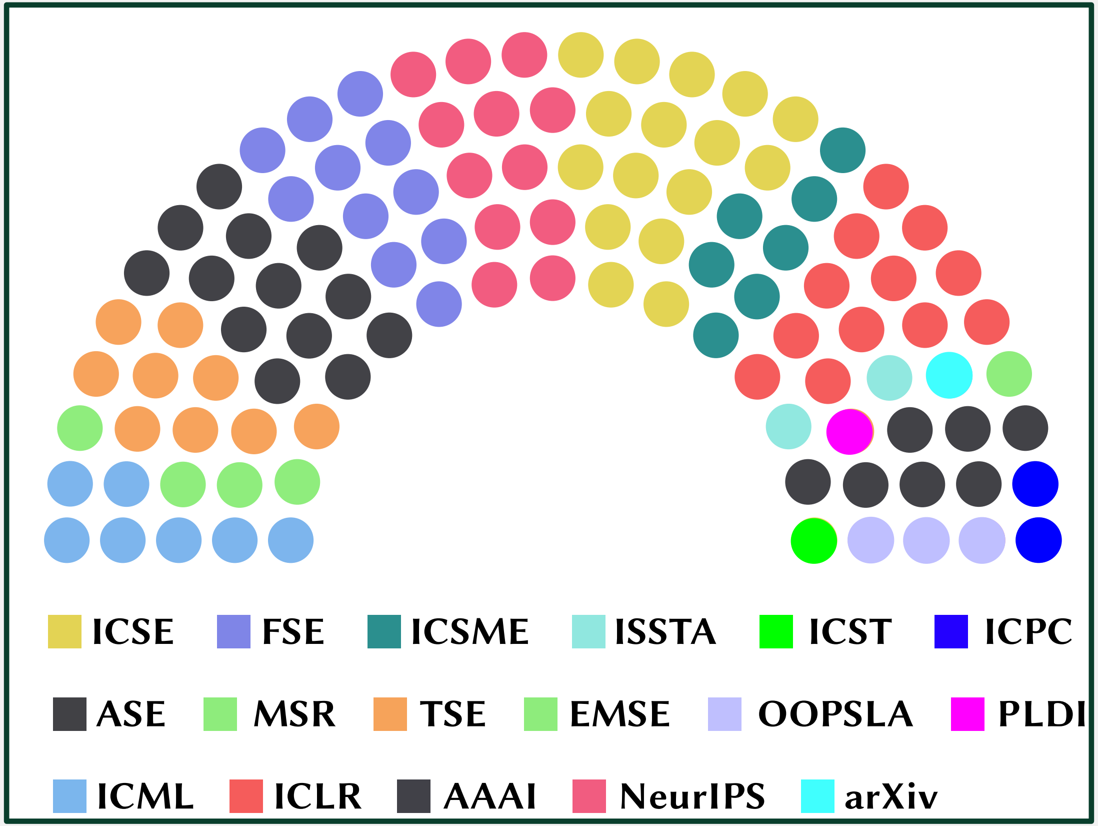
Systematic Literature Review

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

SLR Search Process

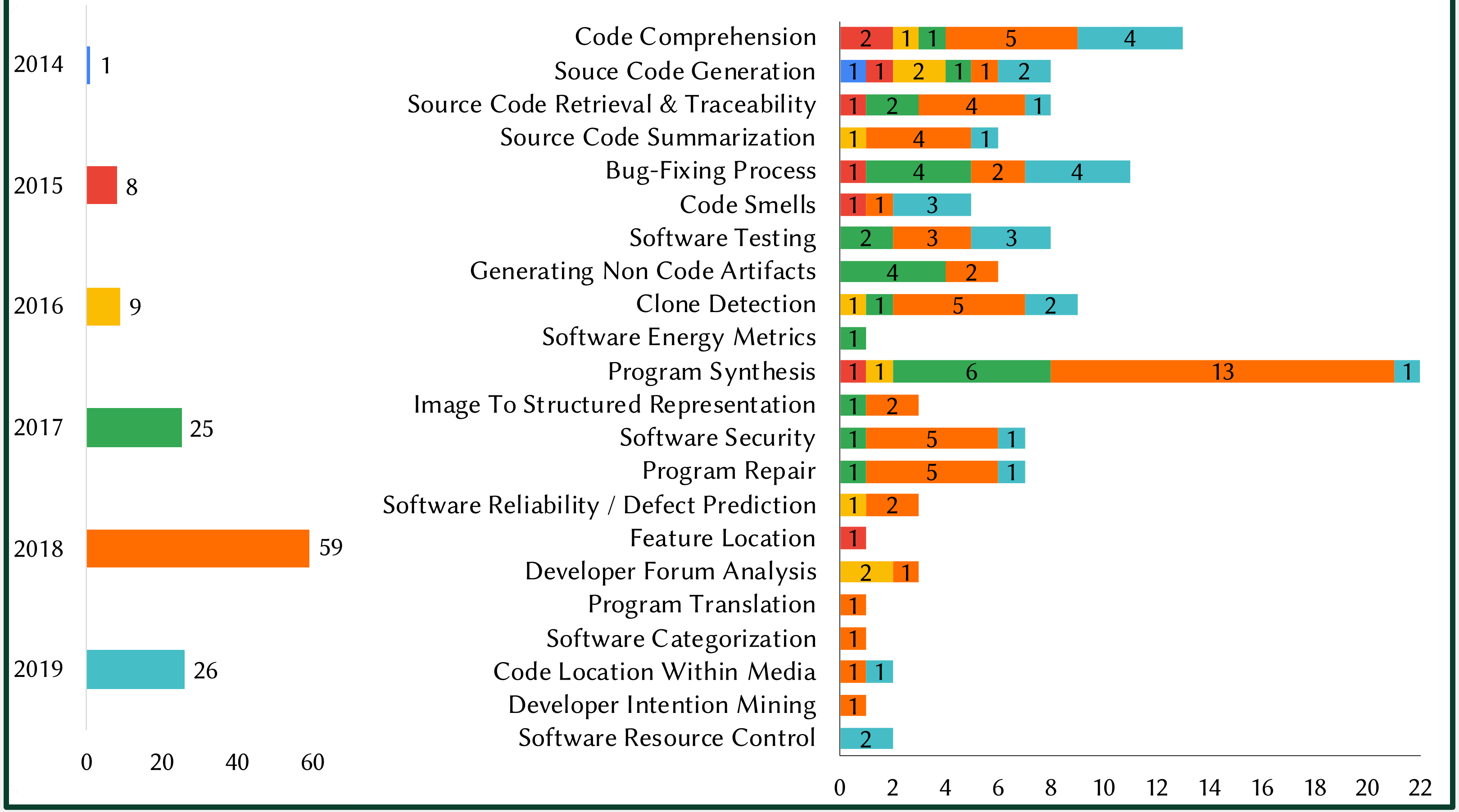


Publication Distribution By Venue



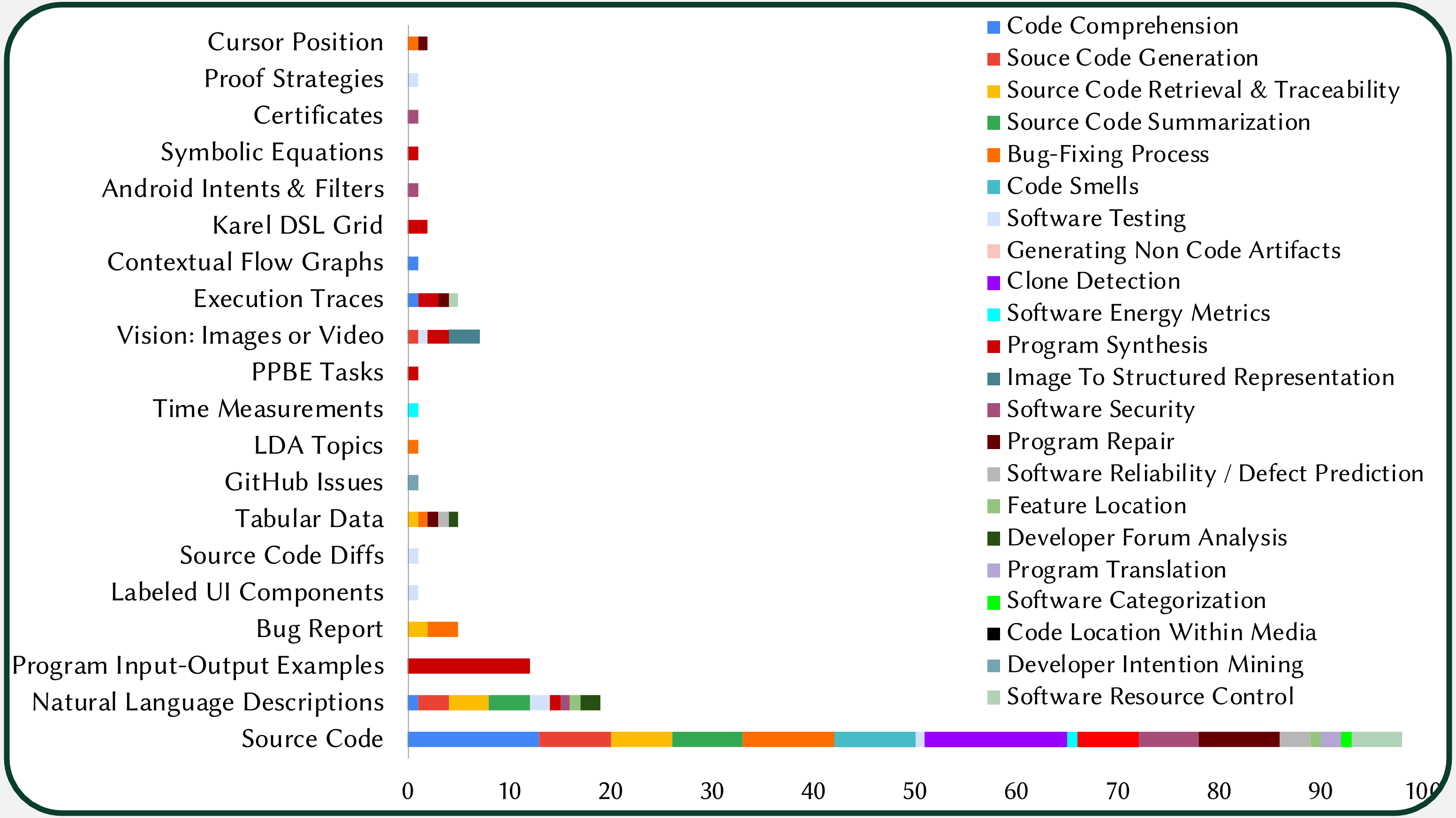
RQ_1: Target Function (SE Task)

DL4SE Publications Over Time and SE Tasks

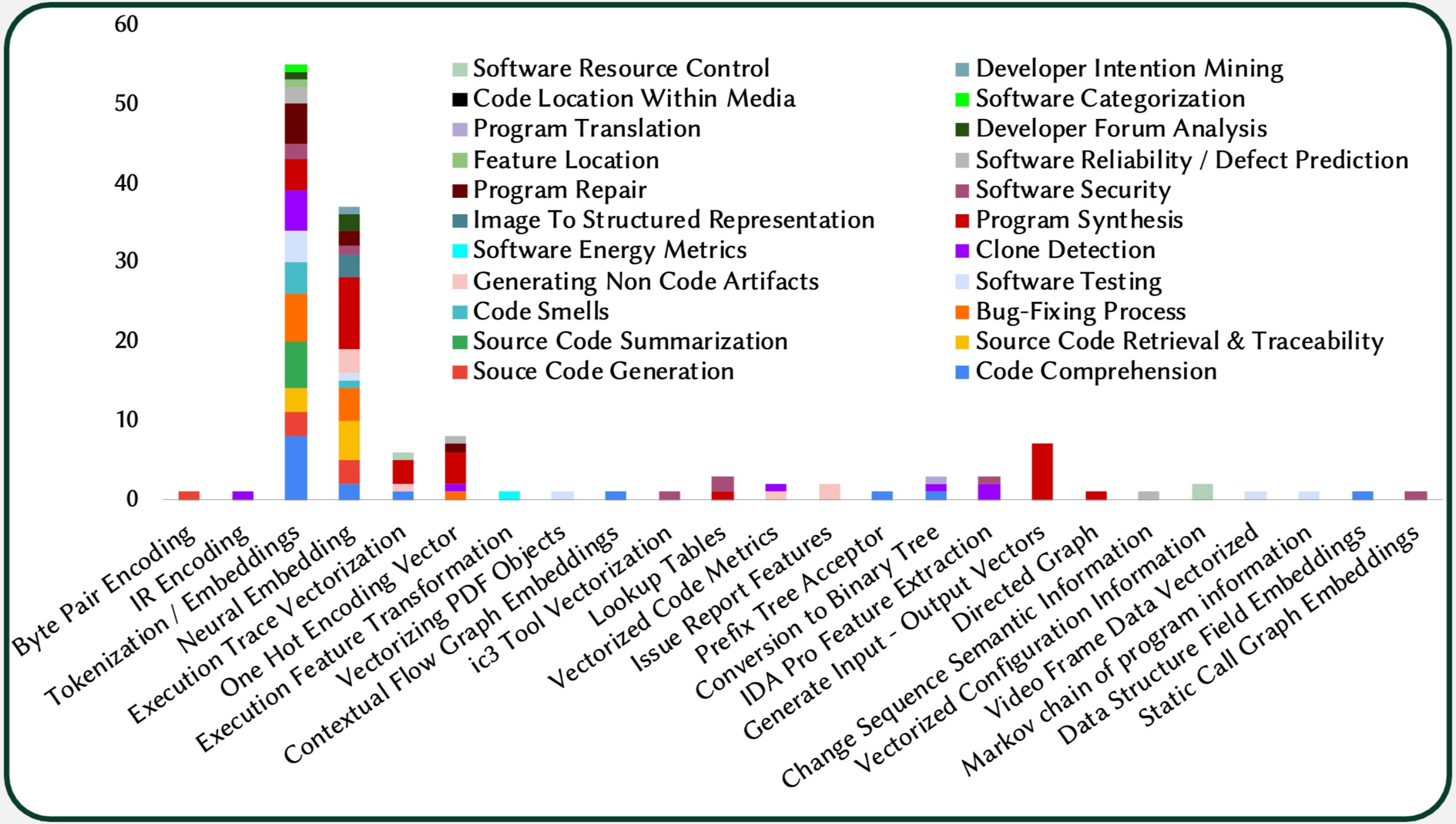


RQ_2: Data (Training/Testing Data)

Data Used in DL4SE Approaches by SE Task

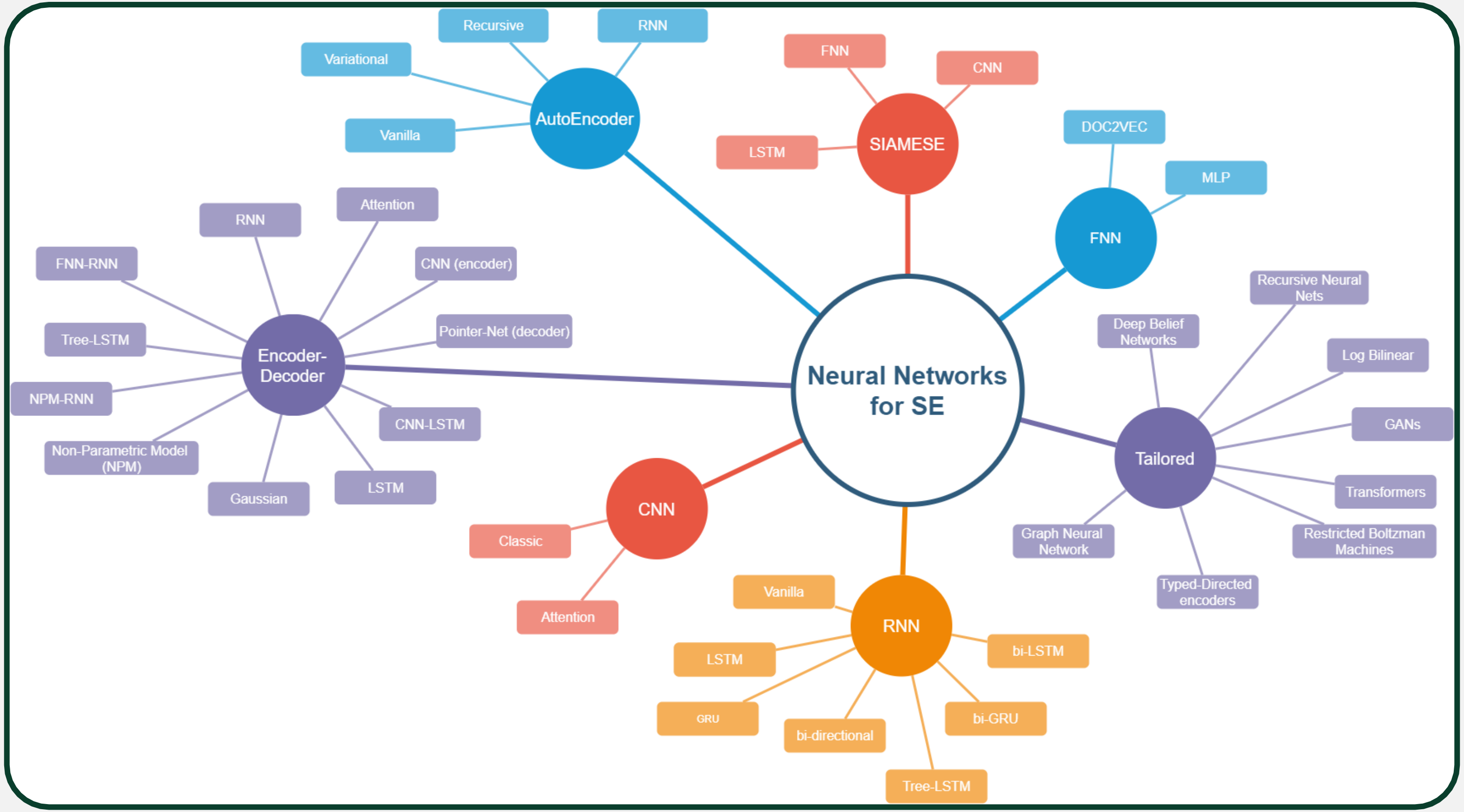


Data Processing Techniques by SE Task

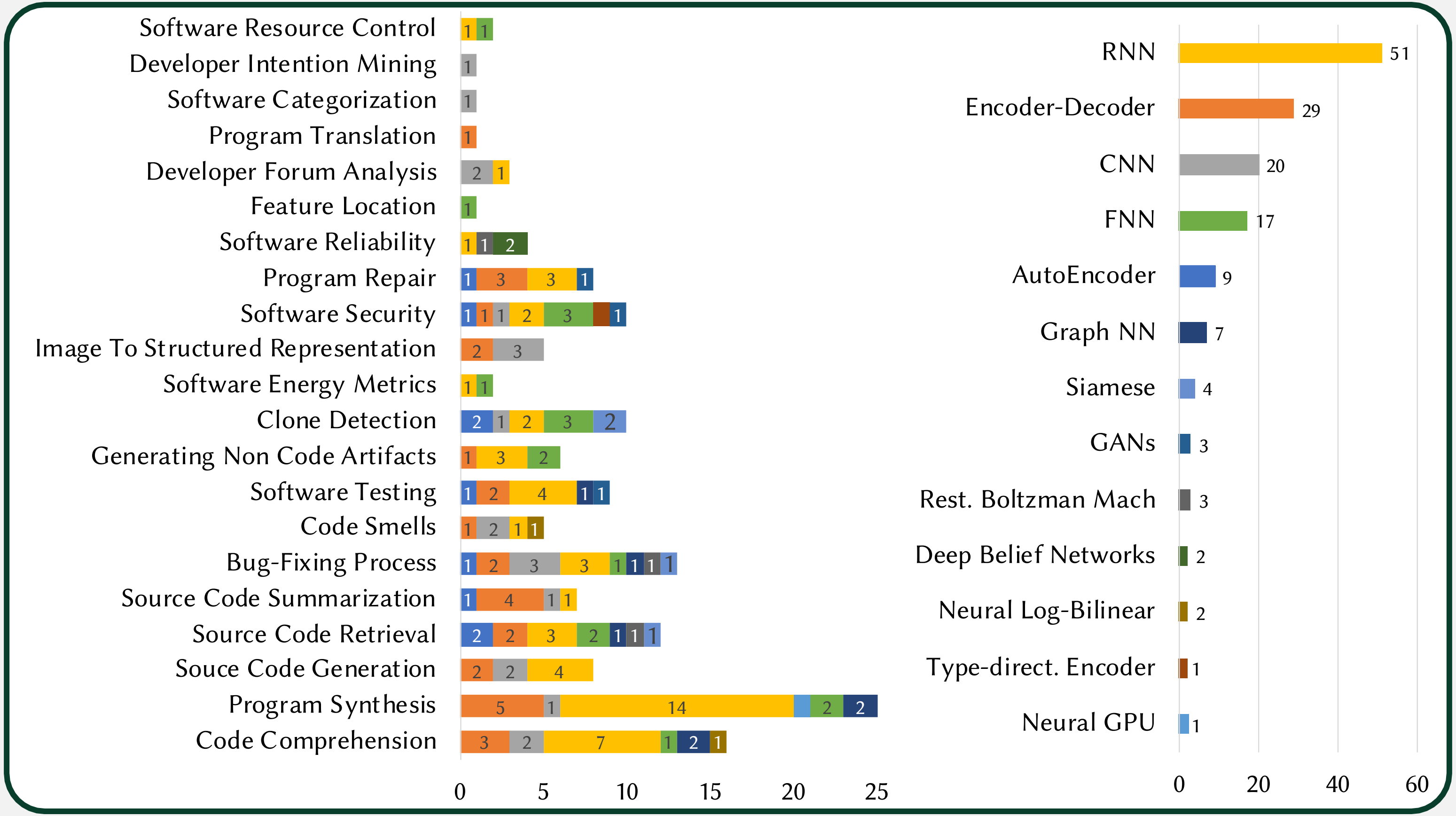


RQ_2: Learning Model (Algorithm + Hypothesis Set)

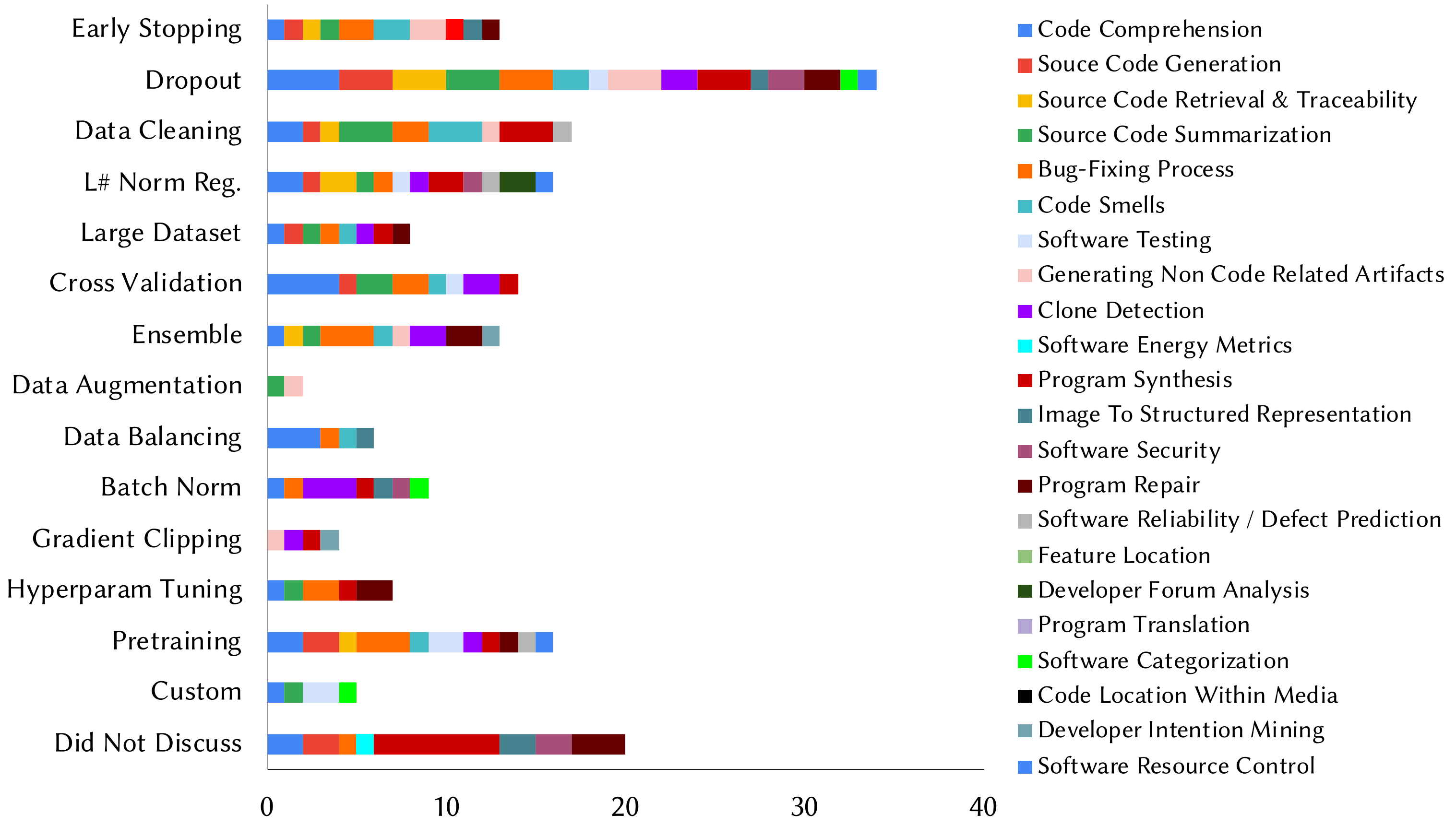
DL4SE Neural Network Architectures



DL4SE Neural Network Architectures

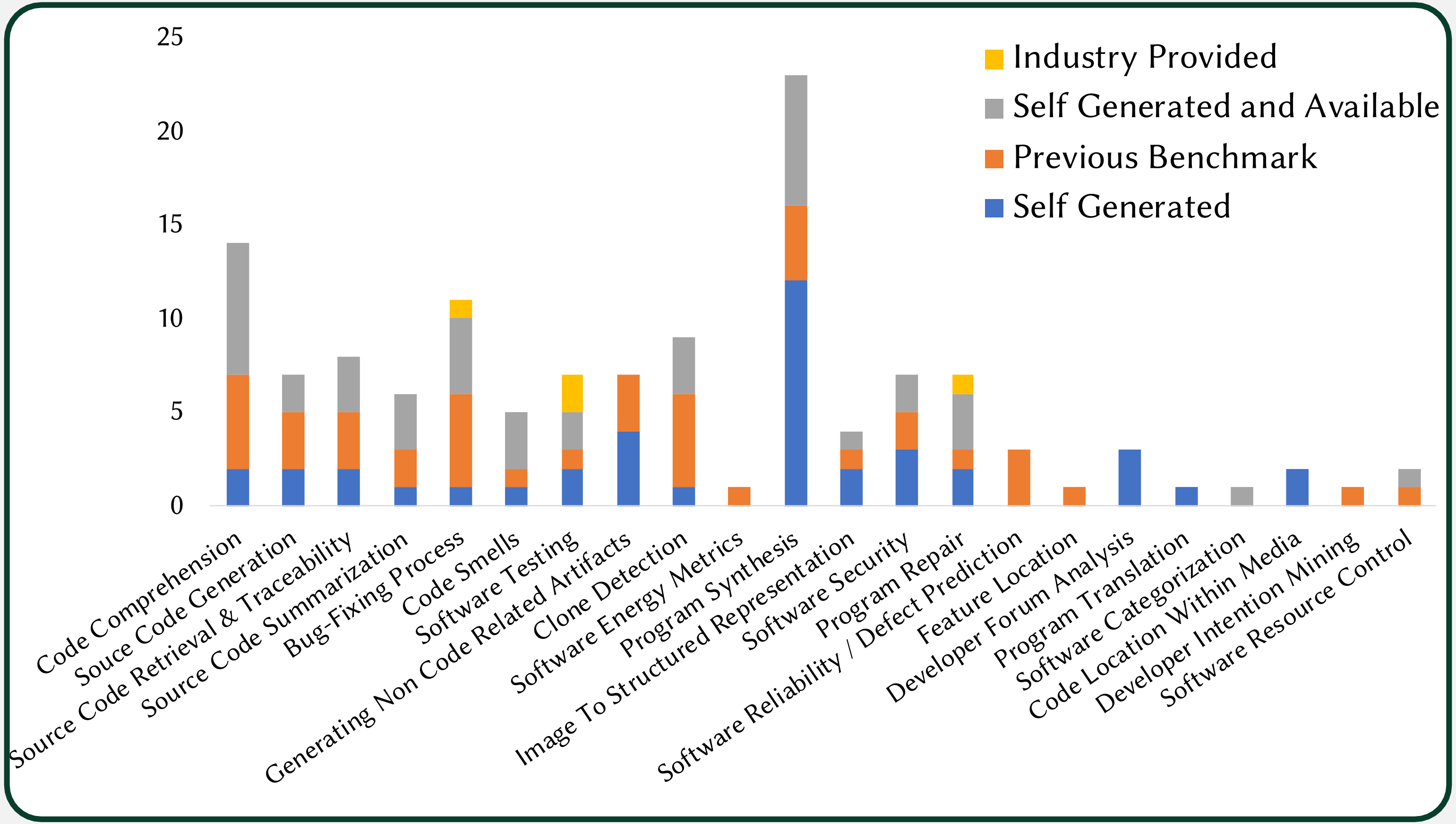


DL4SE Techniques to Combat Overfitting

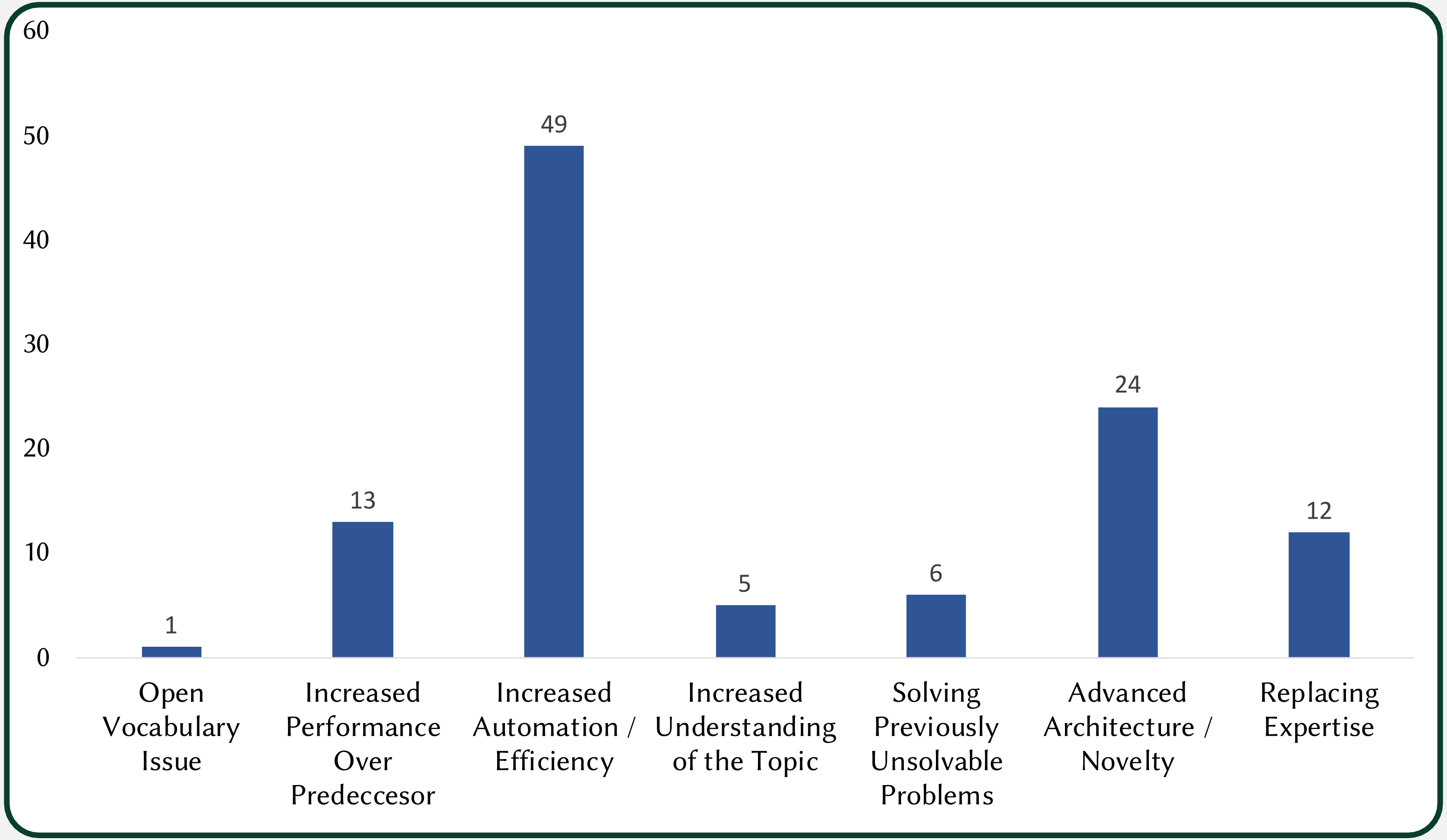


RQ_4: Final Hypothesis (Results)

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:

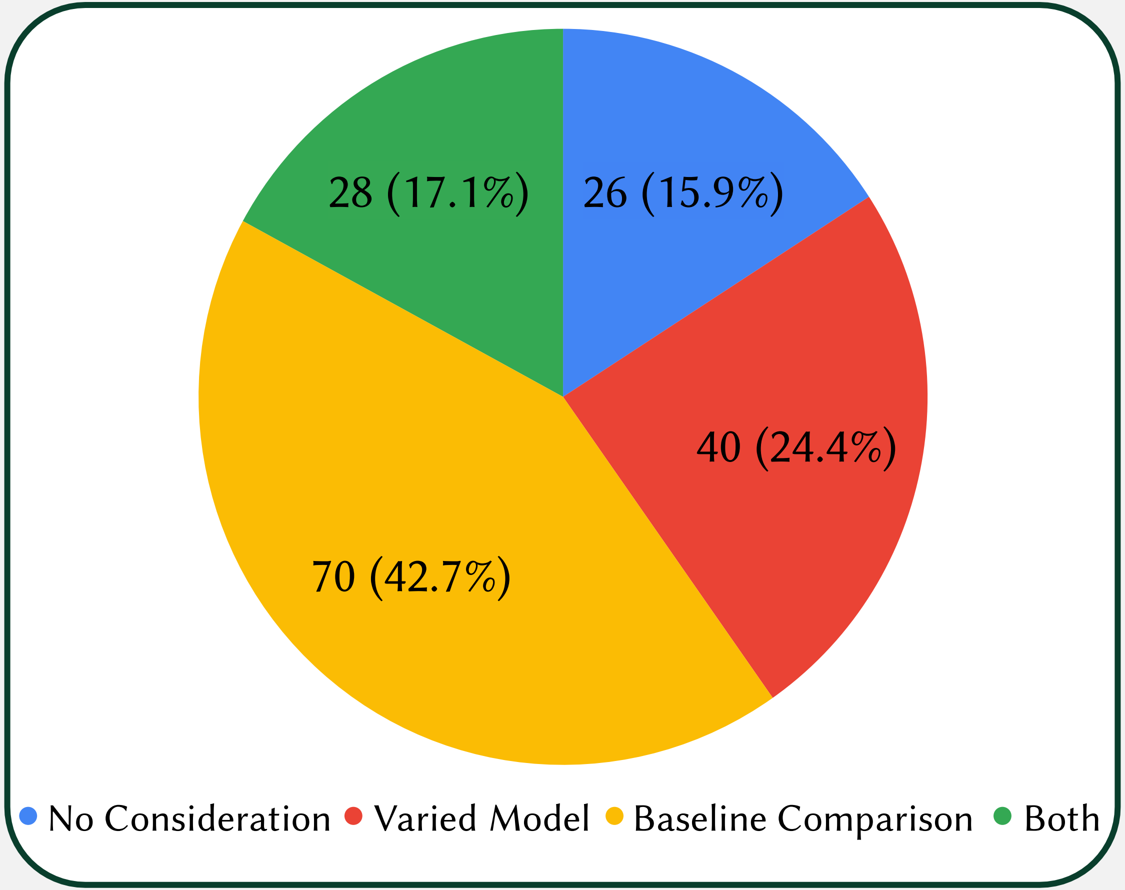
- XU report that their analysis required 14 hours of CPU.
- Le [40] used a cluster with 1,000 machines (16,000 cores) for three days to train a deep learner.

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.

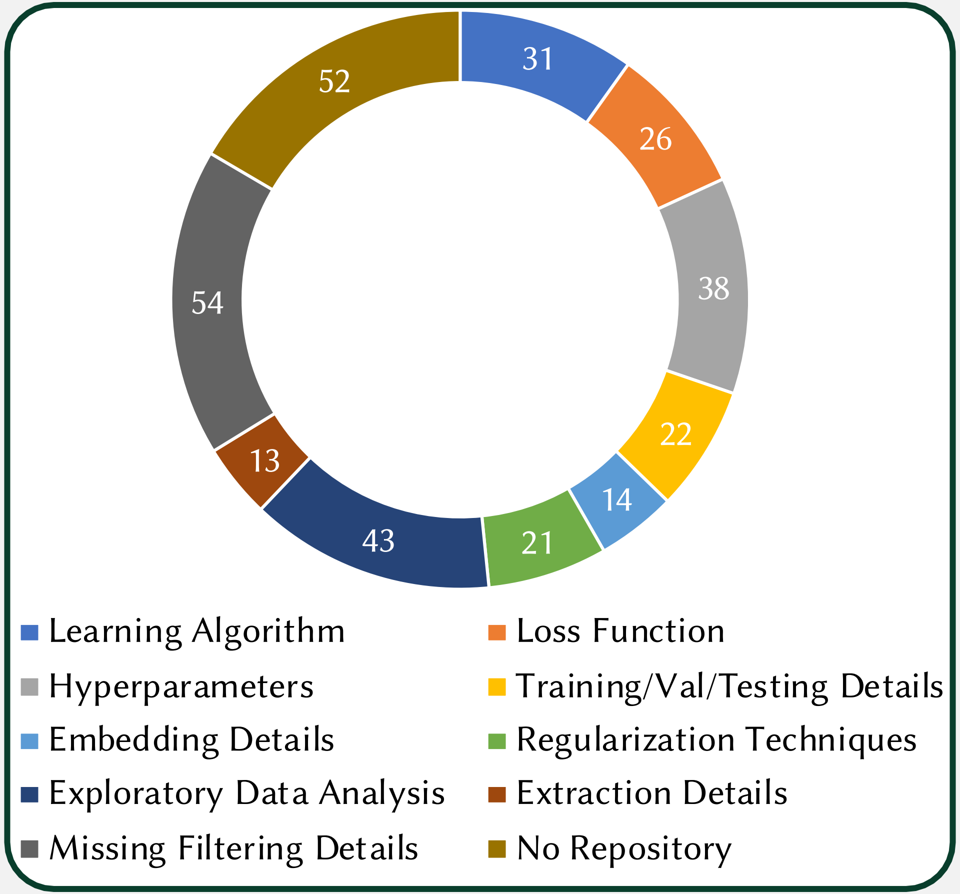
RQ2*: Can DE tune a standard learner such that it outperforms XU’s deep learning method?* We apply differential evolution to tune

Consideration of Occam's Razor

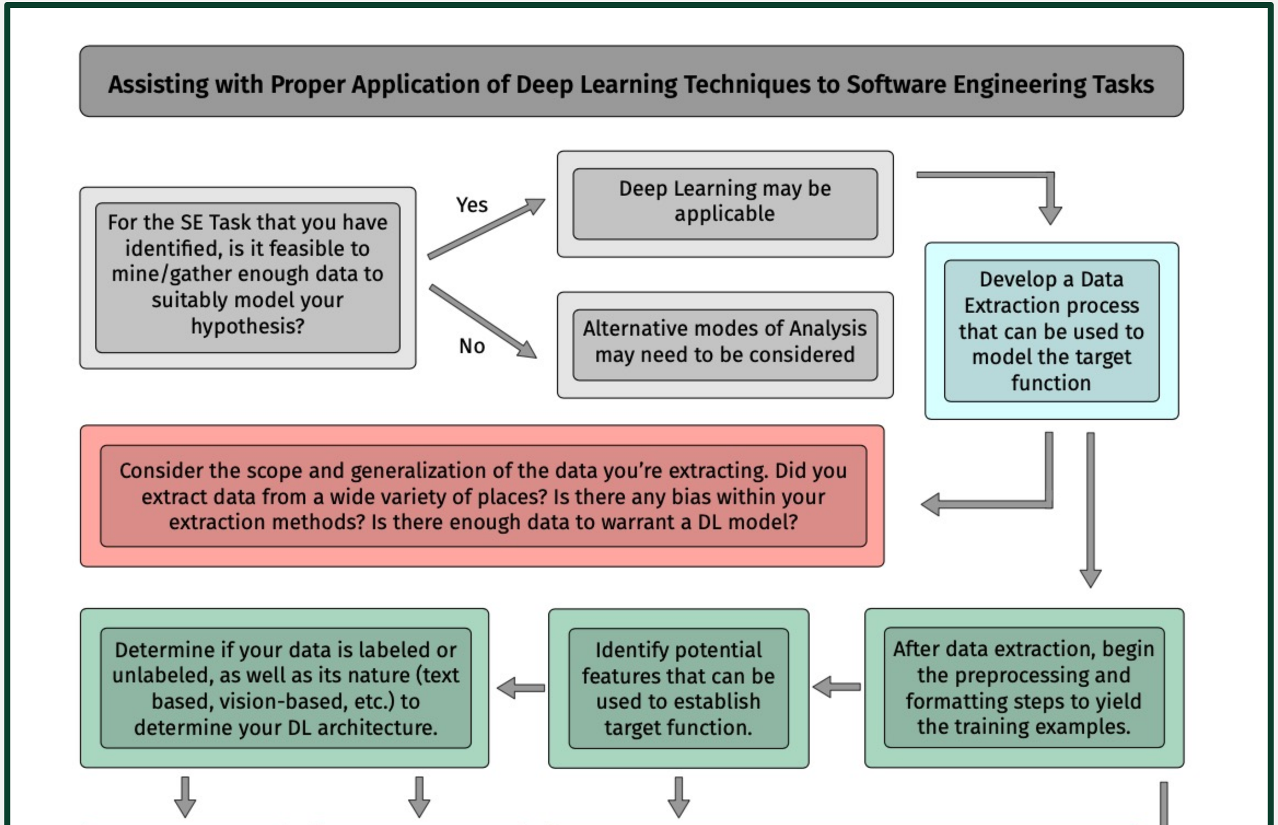


RQ_5: Reproducibility & Replicability

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

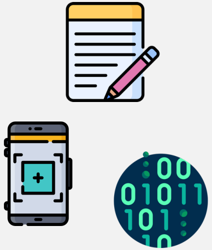
SE4DL: Where Deep Learning Techniques are viewed as a new form of software development that needs tool and process support

Future Work on DL4SE

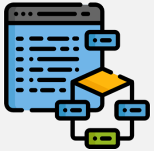
Future Research Directions in DL4SE

Combining Features Learned via DL with Existing Empirical Knowledge

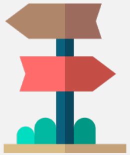
Leveraging & Combining Heterogenous Sources of SE Data



Developing Architectures tailored for SE Data



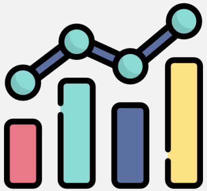
Systematic & Reproducible Research Methodology



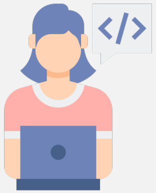
Ethical & Social Considerations of DL4SE

Future Research Directions in DL4SE (cont'd)

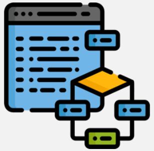
Designing new Effectiveness Metrics for SE-specific Tasks



HCI Aspects of AI-assisted Developer Tools



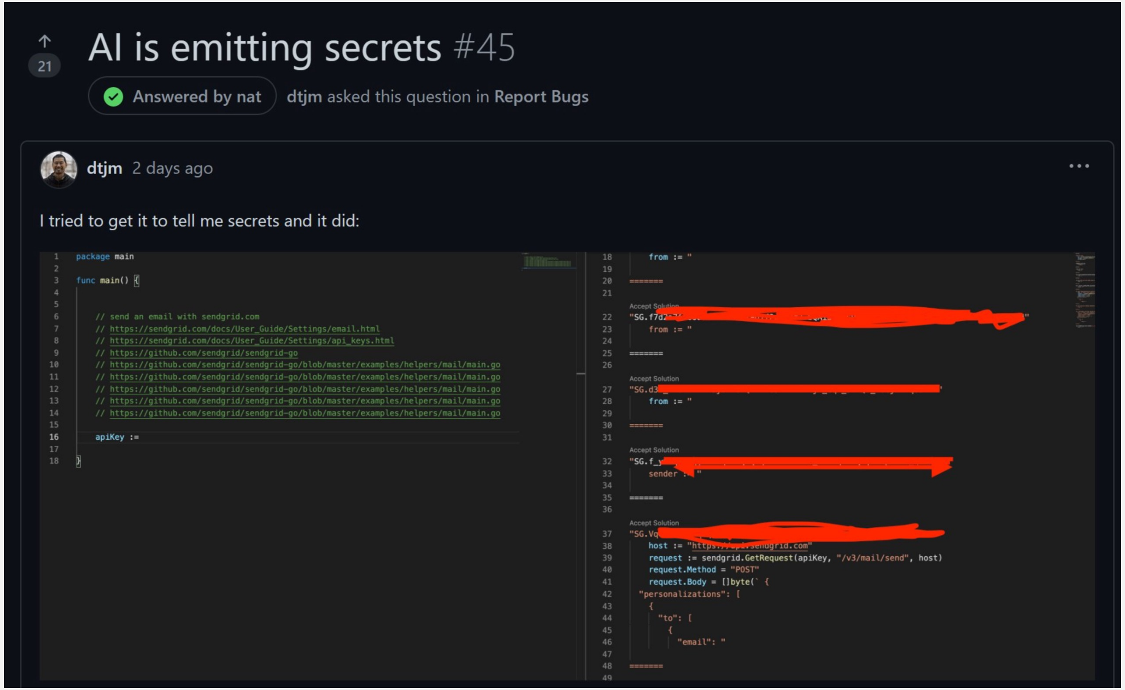
Development of Tailored "Clean" Community Datasets



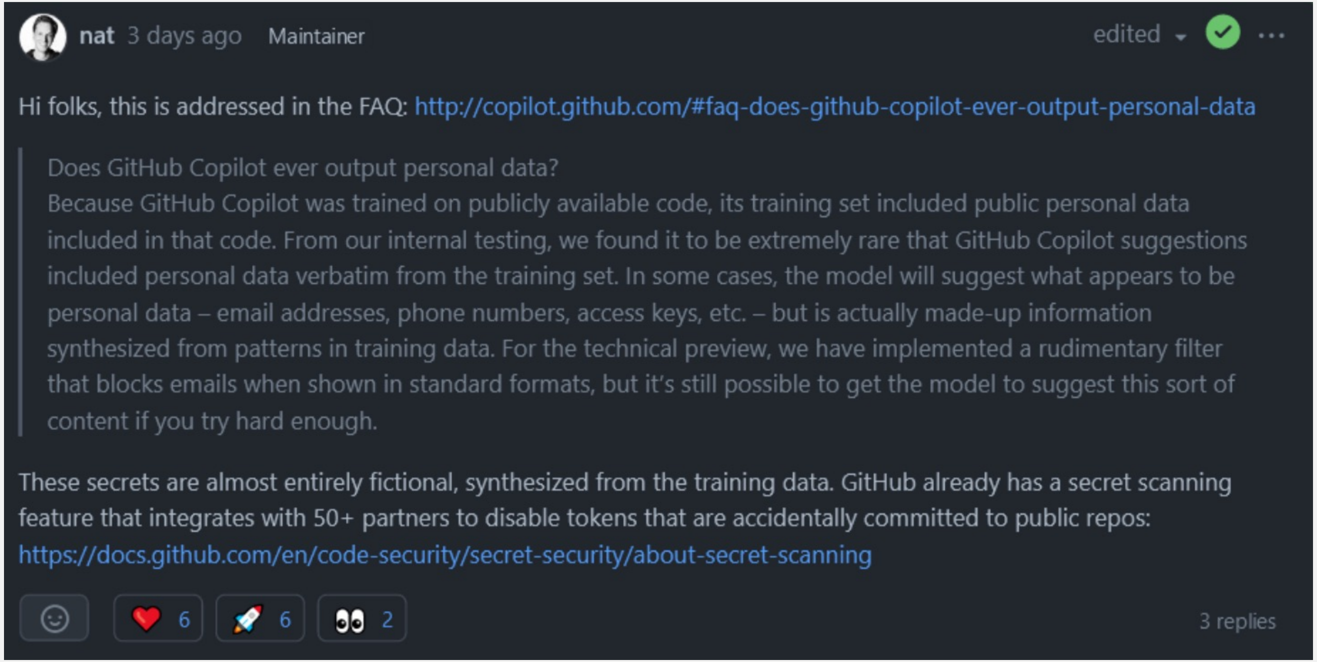
New Application Areas

New Data Sources

Ethical and Social Considerations of DL4SE



Ethical and Social Considerations of DL4SE



nat 3 days ago Maintainer edited

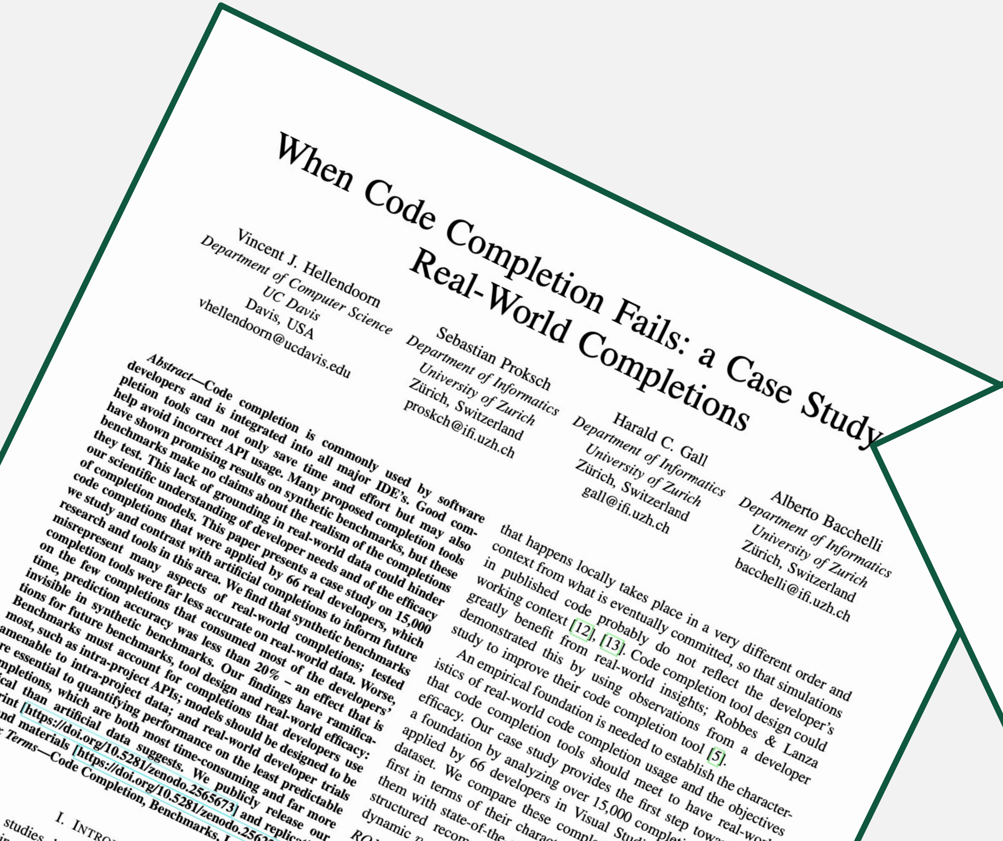
Hi folks, this is addressed in the FAQ: http://copilot.github.com/#faq-does-github-copilot-ever-output-personal-data

> 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 on 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

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

A PREPRINT

Frank F. Xu, Carnegie Mellon University, fangzhex@cs.cmu.edu

Bogdan Vasilescu, Carnegie Mellon University, vasilescu@cs.cmu.edu

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

February 1, 2021

ABSTRACT

New Application Areas and Data-Sources

Potential SE Tasks

Software Testing

Code Review

Troubleshooting Tasks

Bug Triaging

Requirements Engineering

Potential Data Sets

Tailored for SE Tasks

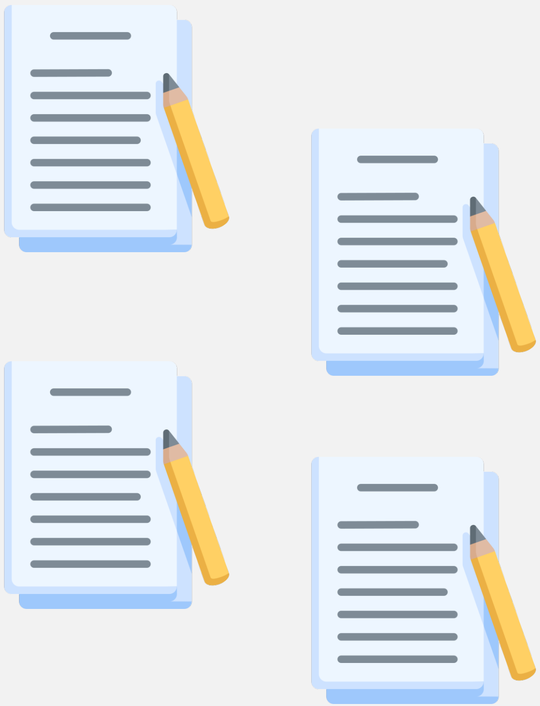
Graphical Software Artifacts

IDE Instrumentation

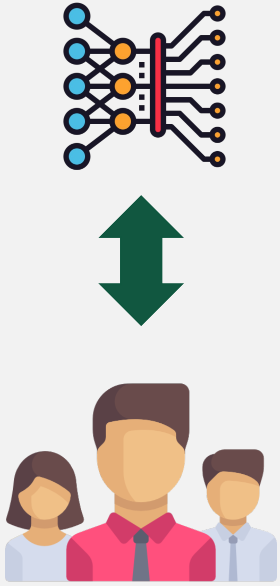
EDA for Datasets

Combining Empirical Knowledge with Deep Learning

Empirical SE Studies



Deep Learning Tools



Future Work on SE4DL

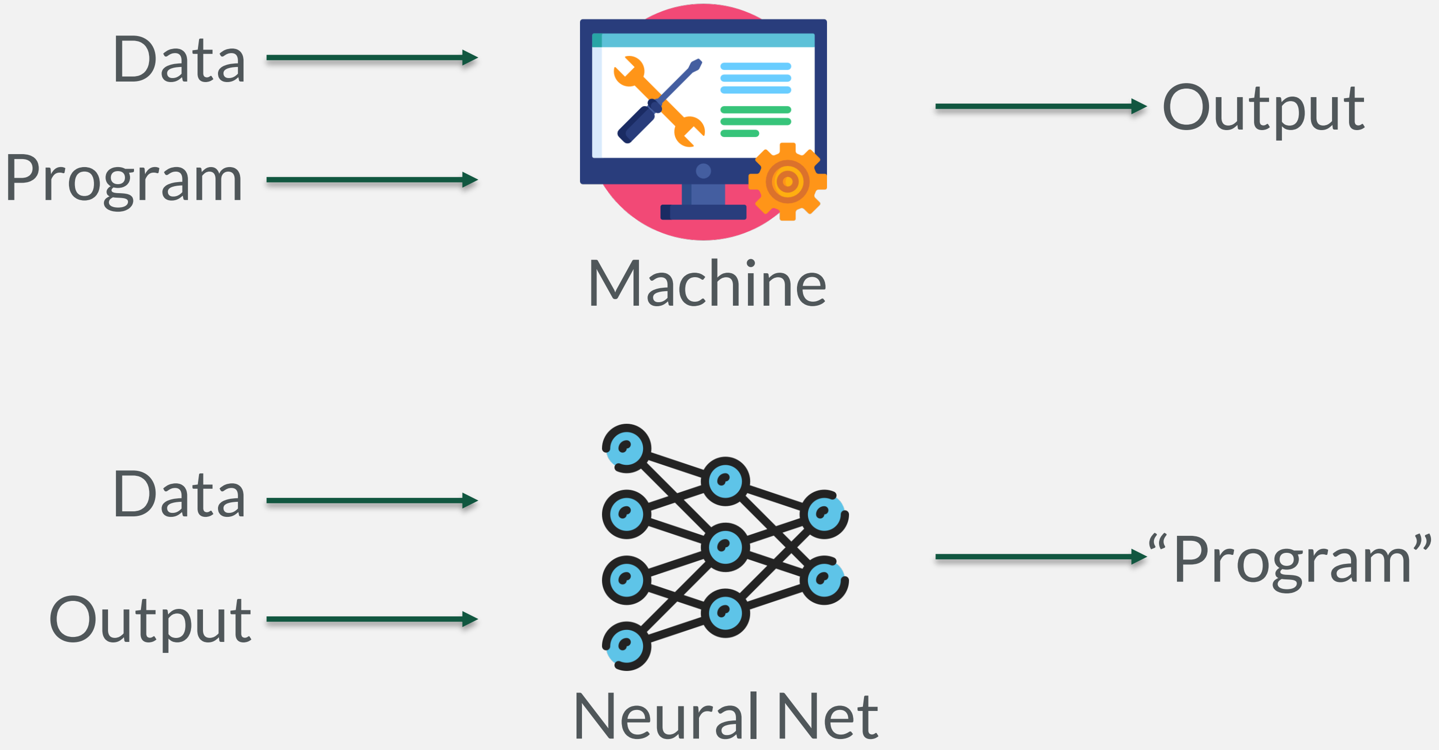
“Gradient descent can write code better than you. I’m sorry”

-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

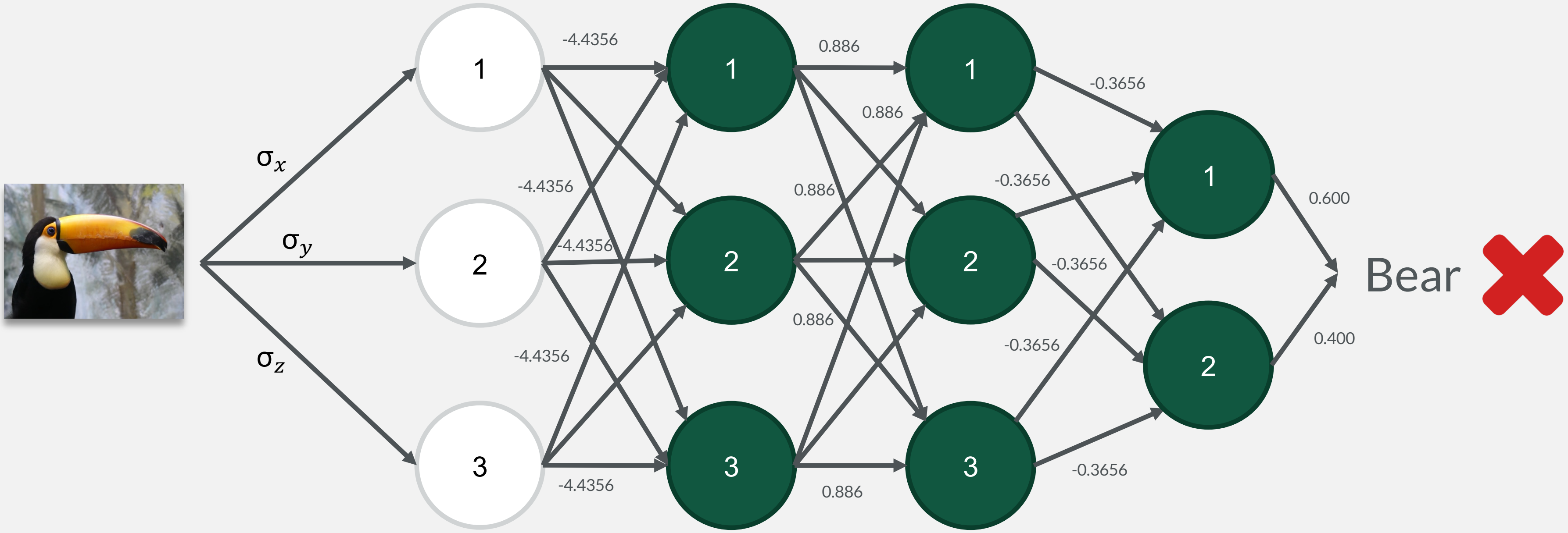
Software 1.0 vs. Software 2.0



Software 1.0

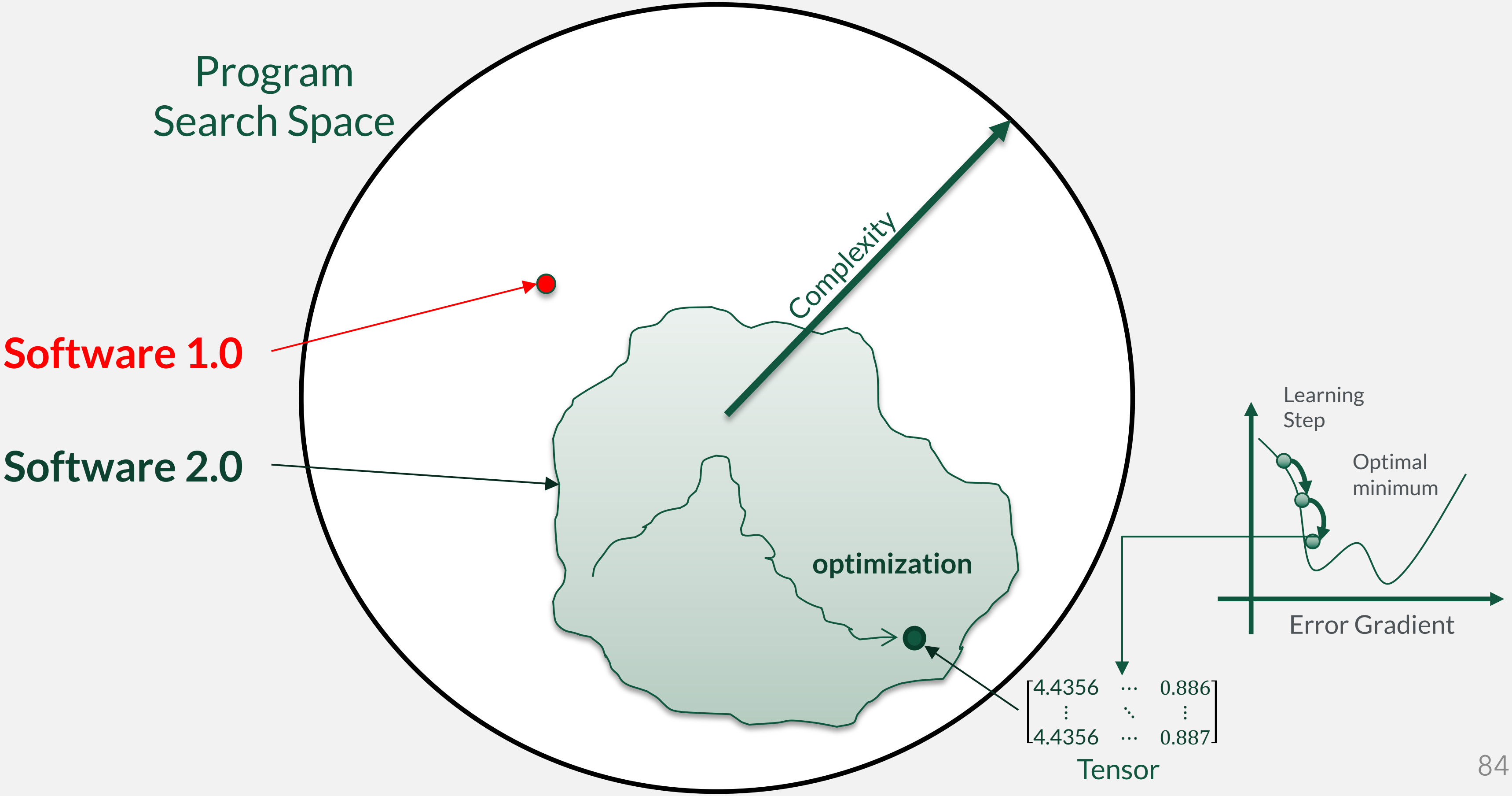
```
/**
* Add element in the list
* @param element to add
* @return true if element added, false otherwise
*/
public boolean addElement (Element elem) {
  if(myList != null){
     myList.add(elem);
    return true;
  }
  return false;
 }
```

Software 2.0 = DL-based systems

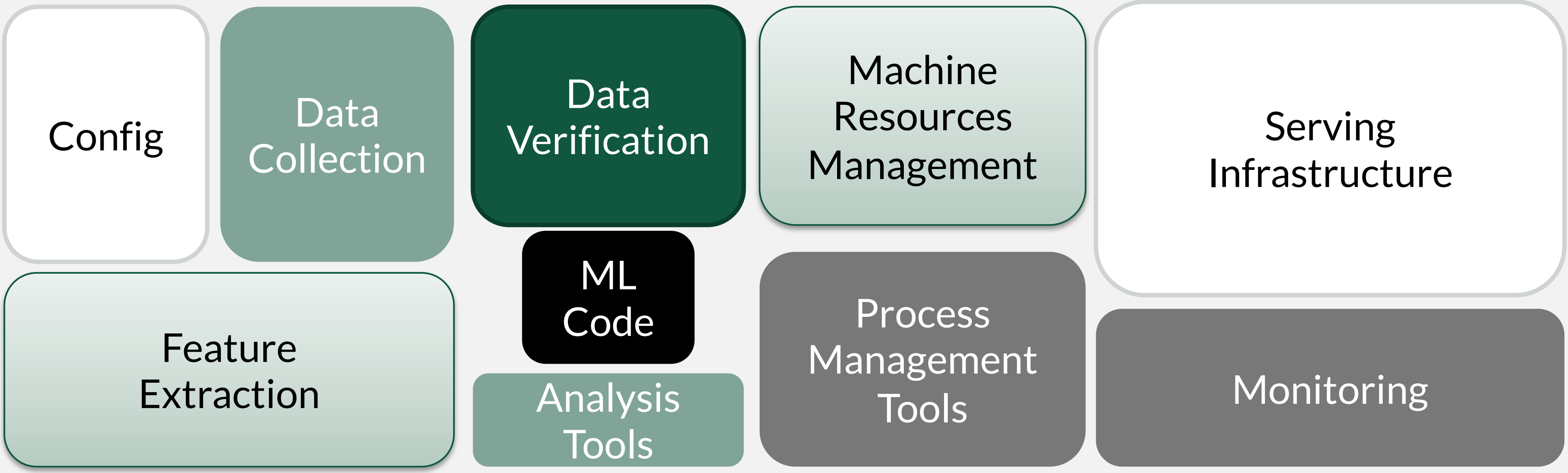


How is Deep Learning Software 2.0?

Optimization by Gradient Descent to Find "The Program"



Real-world DL-based System (Software 2.0)



Yesterday's Devs vs. Tomorrow's Devs

Machine Resources Management

Analysis Tools

Process Management Tools

Serving Infra-structure

ML Code

Monitoring

Configuration

Data Collection

Data Verification

ML Code

Feature Extraction

Will Deep Learning encompass all software?

Will Deep Learning encompass all software?

Not quite …

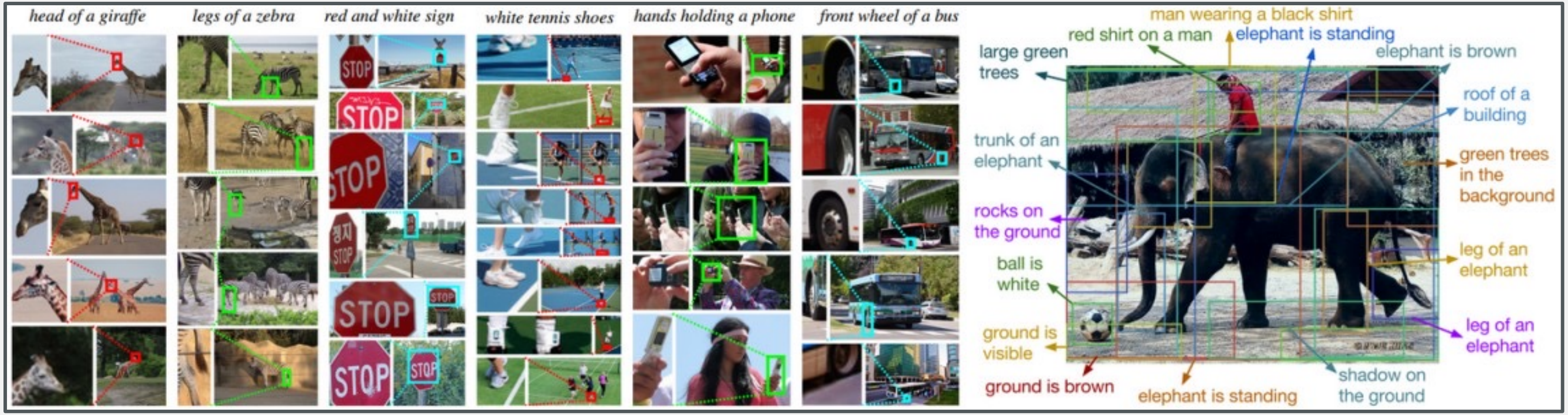
Will Deep Learning encompass all software?

Not quite …

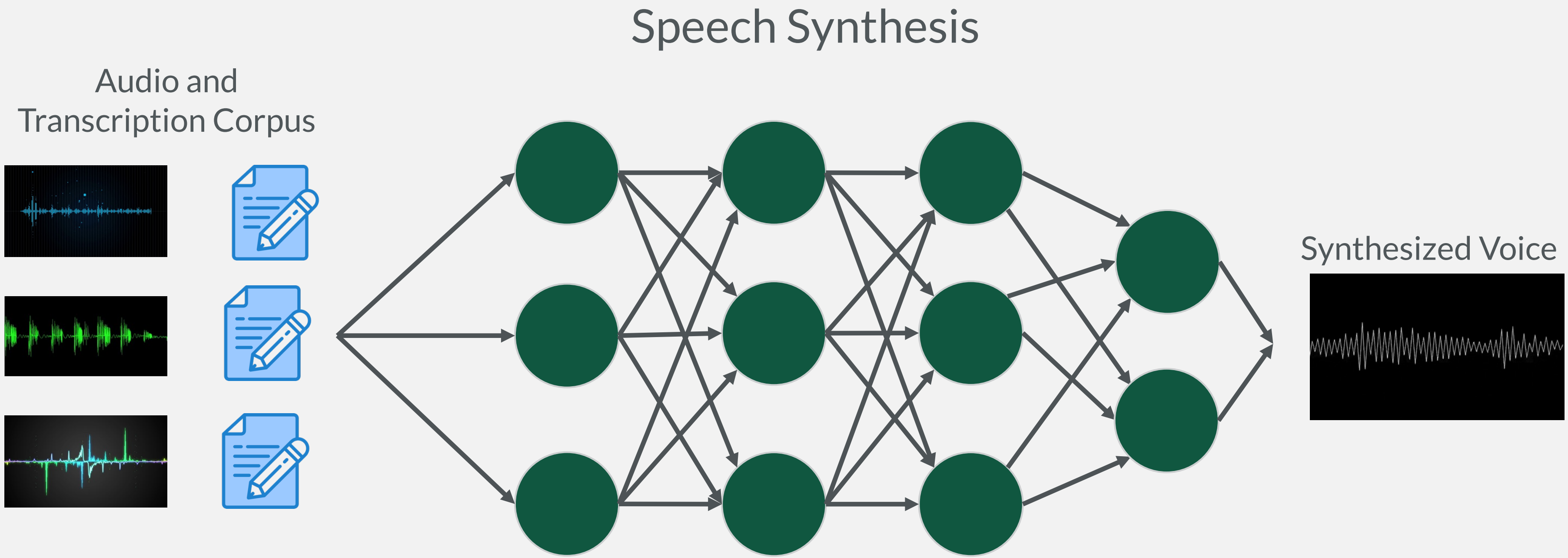
But the applications of DL are numerous and growing!

The Transition to Software 2.0

Image Recognition and Understanding

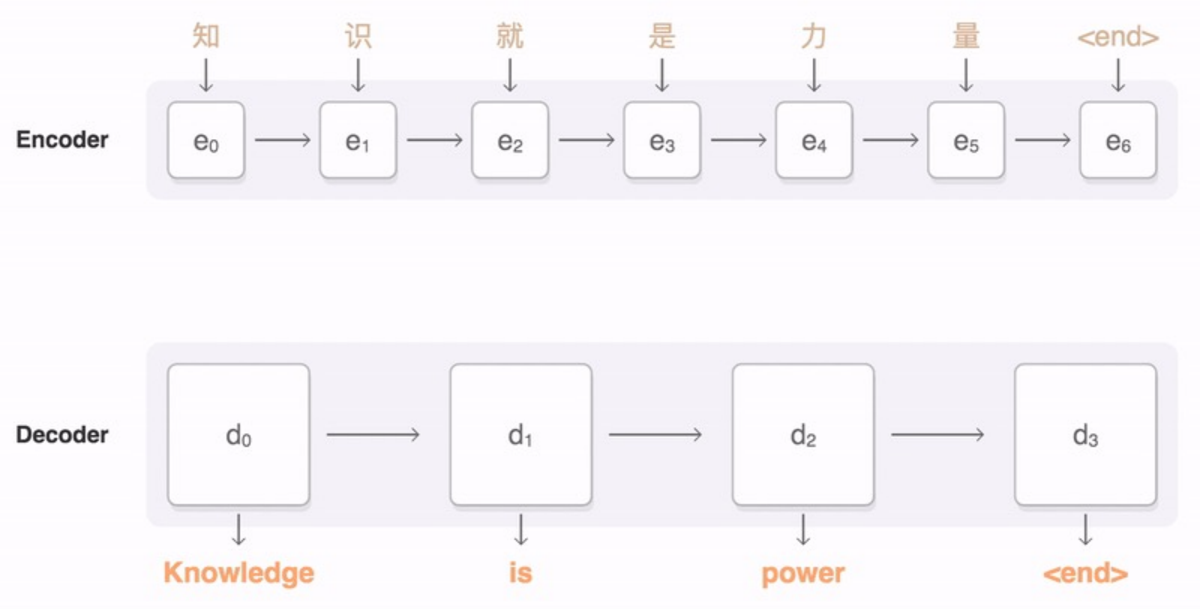


The Transition to Software 2.0



The Transition to Software 2.0

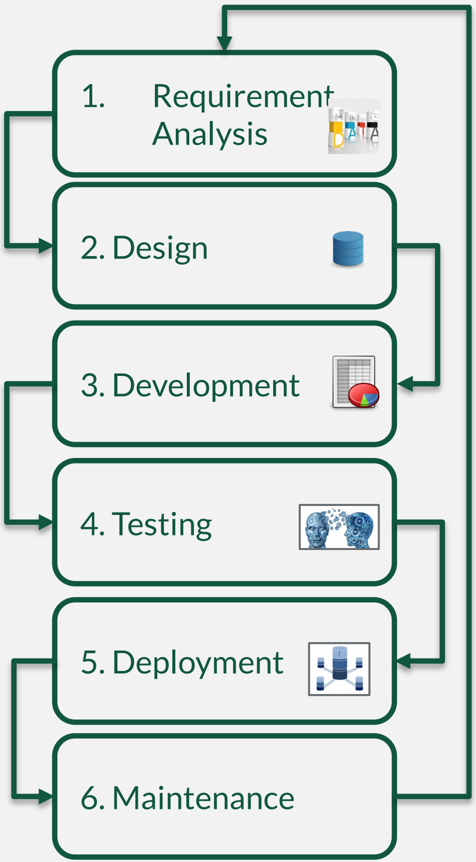
Machine Translation

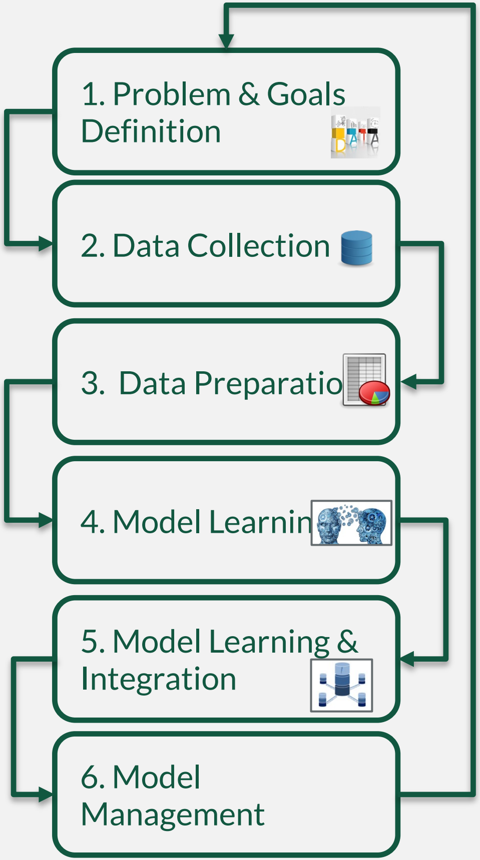


Benefits of Software 2.0

- 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)

Software development challenges

Maintenance challenges

Testing Challenges

Debugging Challenges

Deployment Challenges

Other: security, privacy, explainability, reuse

Challenges: Software Development for DL

Deriving Requirements

Effort Estimation

Experiment Management

Data Labeling

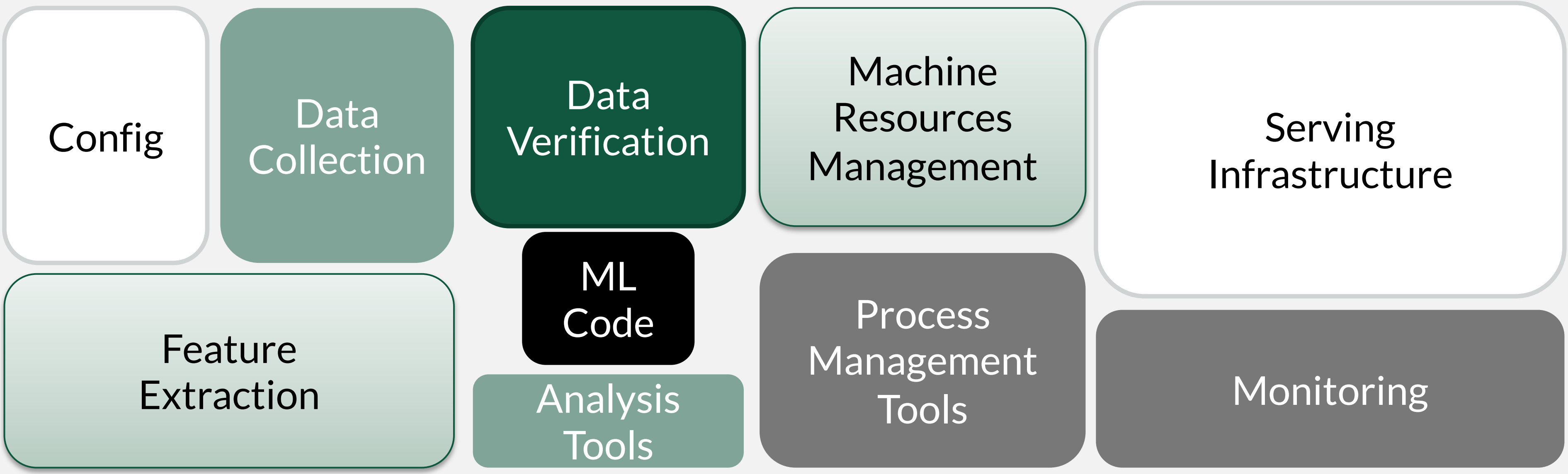
Versioning Models

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

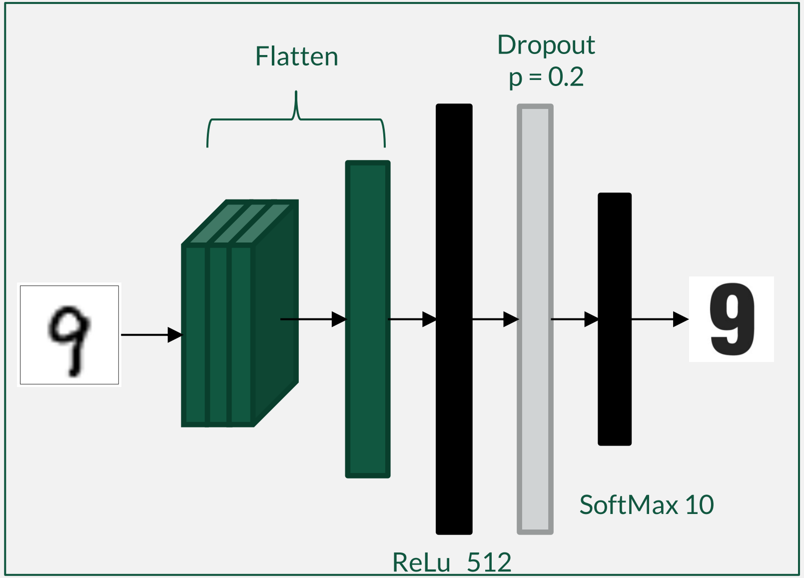
Supervised

Model Definition

Loss Function

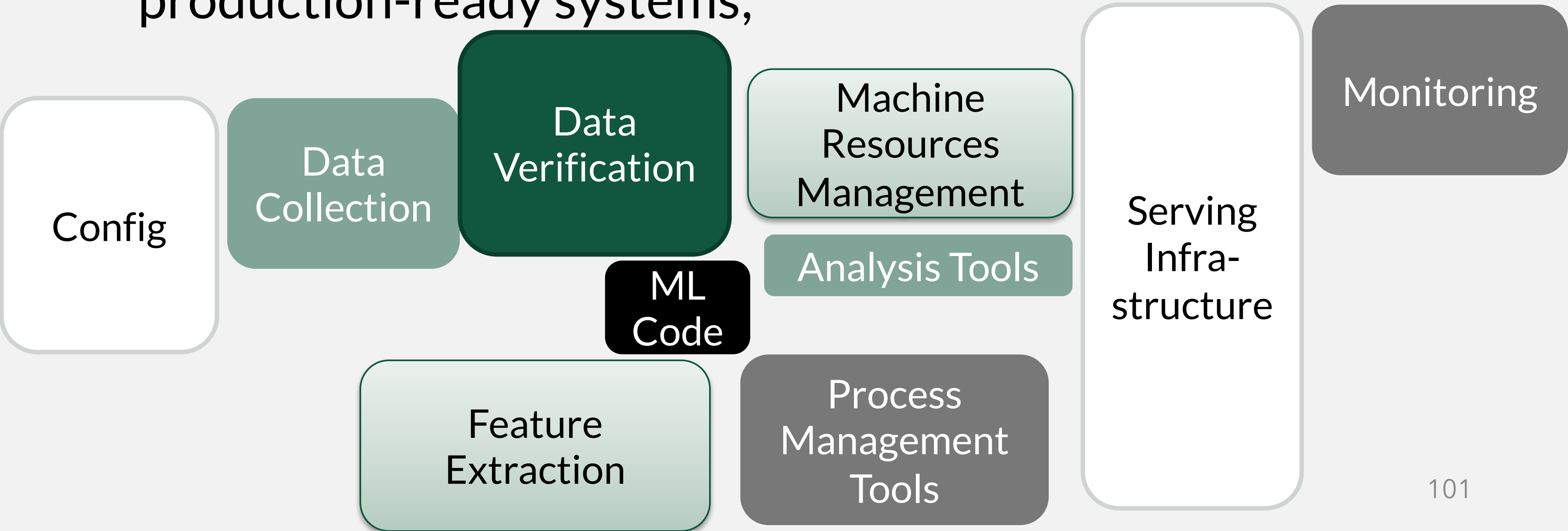
Evaluation

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
 tf.keras.layers.Flatten(),
 tf.keras.layers.Dense(512, activation=tf.nn.relu),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam’,
             loss='sparse_categorical_crossentropy’,
             metrics=['accuracy’])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```



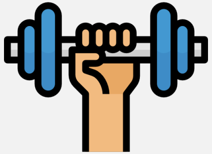
Challenges: Testing for DL

- 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

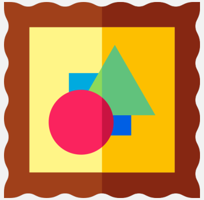


"Traditional" Debuggers Don't Apply

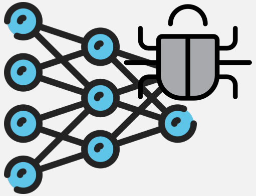
Lazy Execution

Bugs in Dataset

Bugs can be Abstract

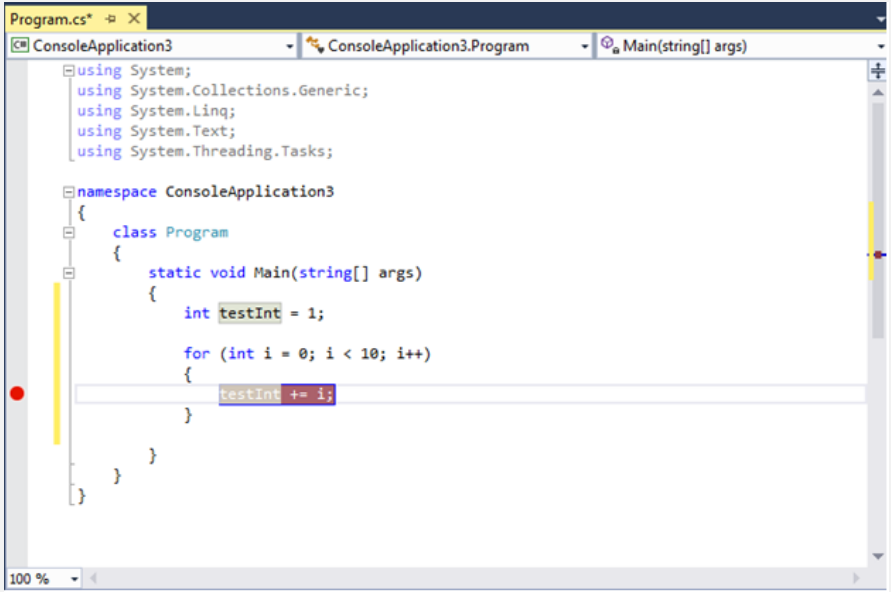


DNN Bugs



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



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

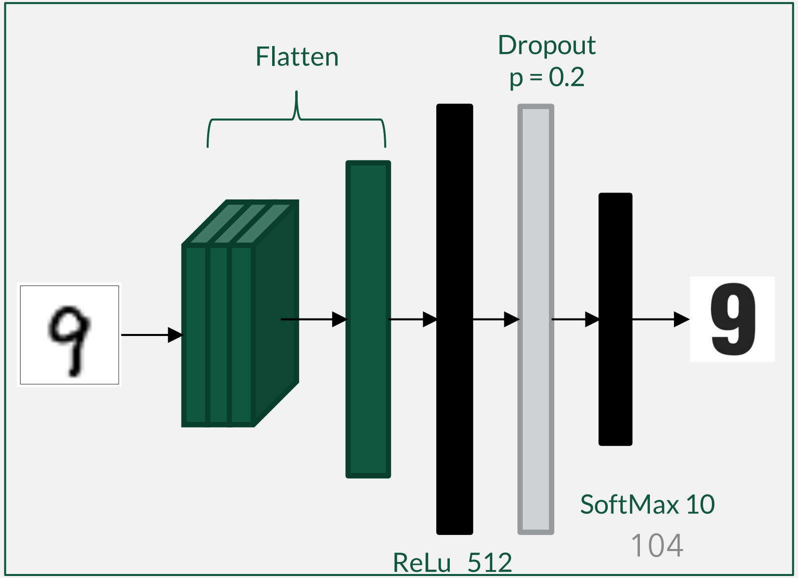
Supervised

Model Definition

Loss Function

Evaluation

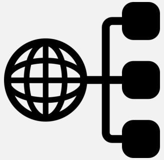
```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
 tf.keras.layers.Flatten(),
 tf.keras.layers.Dense(512, activation=tf.nn.relu),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam’,
              loss='sparse_categorical_crossentropy’,
              metrics=['accuracy’])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```



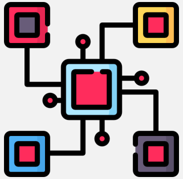
Challenges: DL Deployment

Feedback Loops

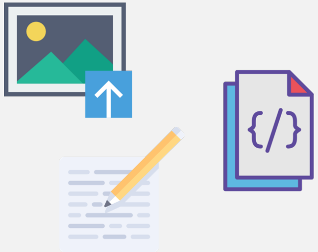
Stream Processing



Distributed DL



Data Modalities

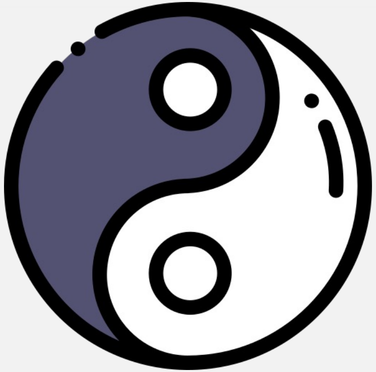


Data Formatting

What are the Next Steps?

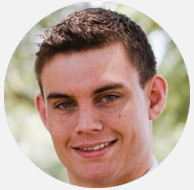
There is still a lot of work to be done!

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