
Toward an Interpretable Science of Deep Learning for Software Engineering

A Causal Inference View

Alejandro Velasco
William & Mary

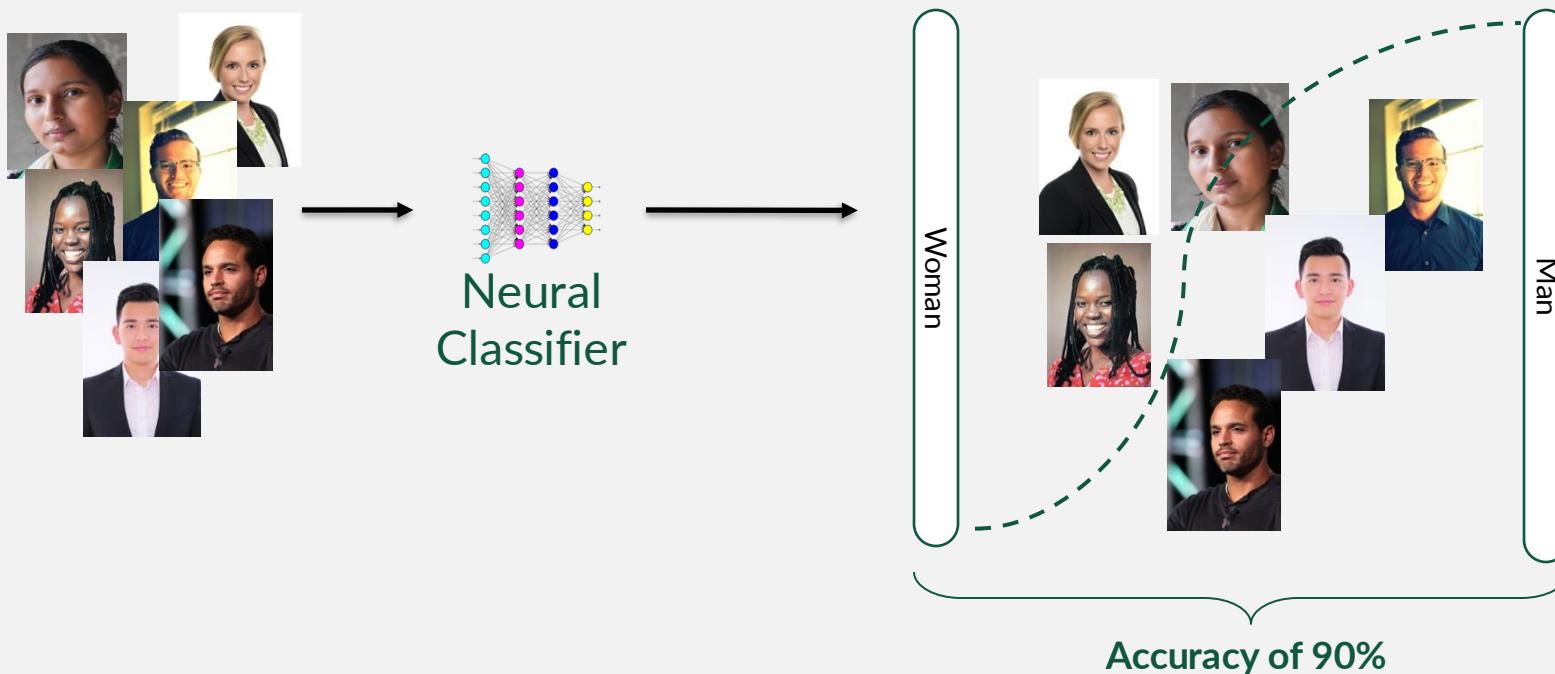
Tuesday, Nov 4th, 2025



WILLIAM & MARY
CHARTERED 1693

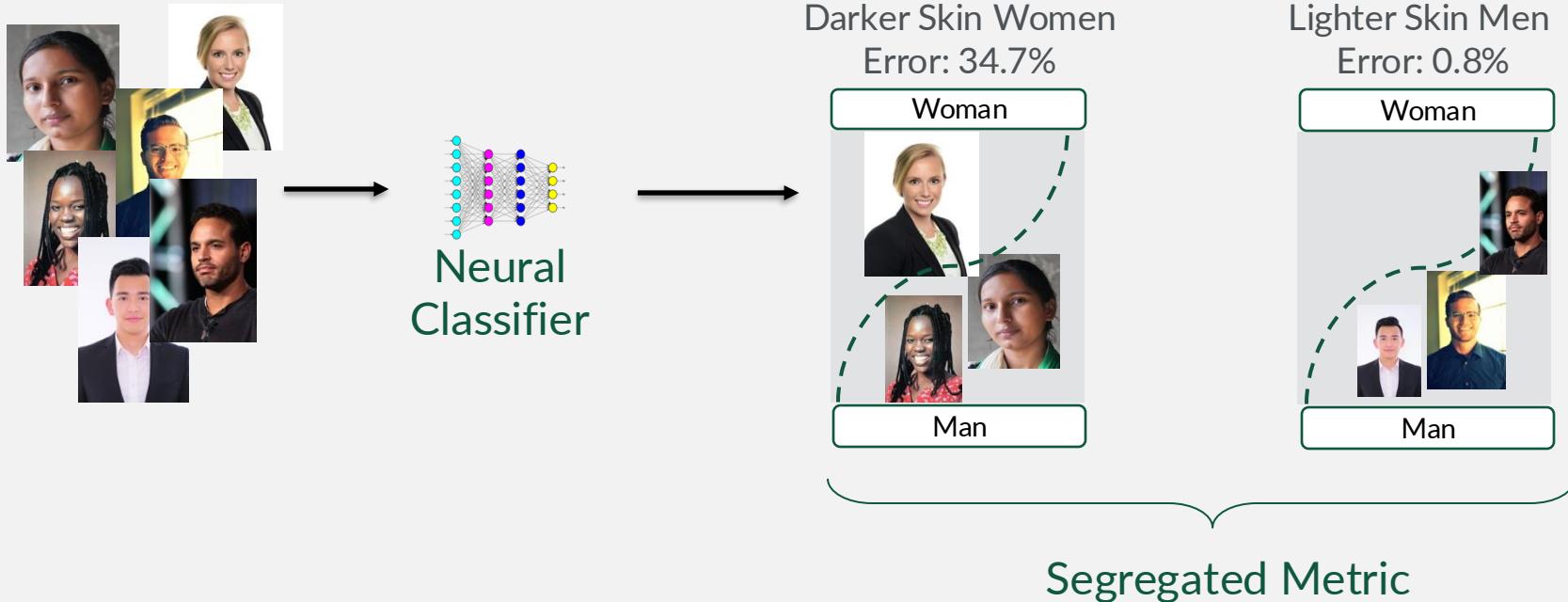
Disparities in gender classification

Aggregated Metrics obfuscate key information about where system tends to succeed or fail



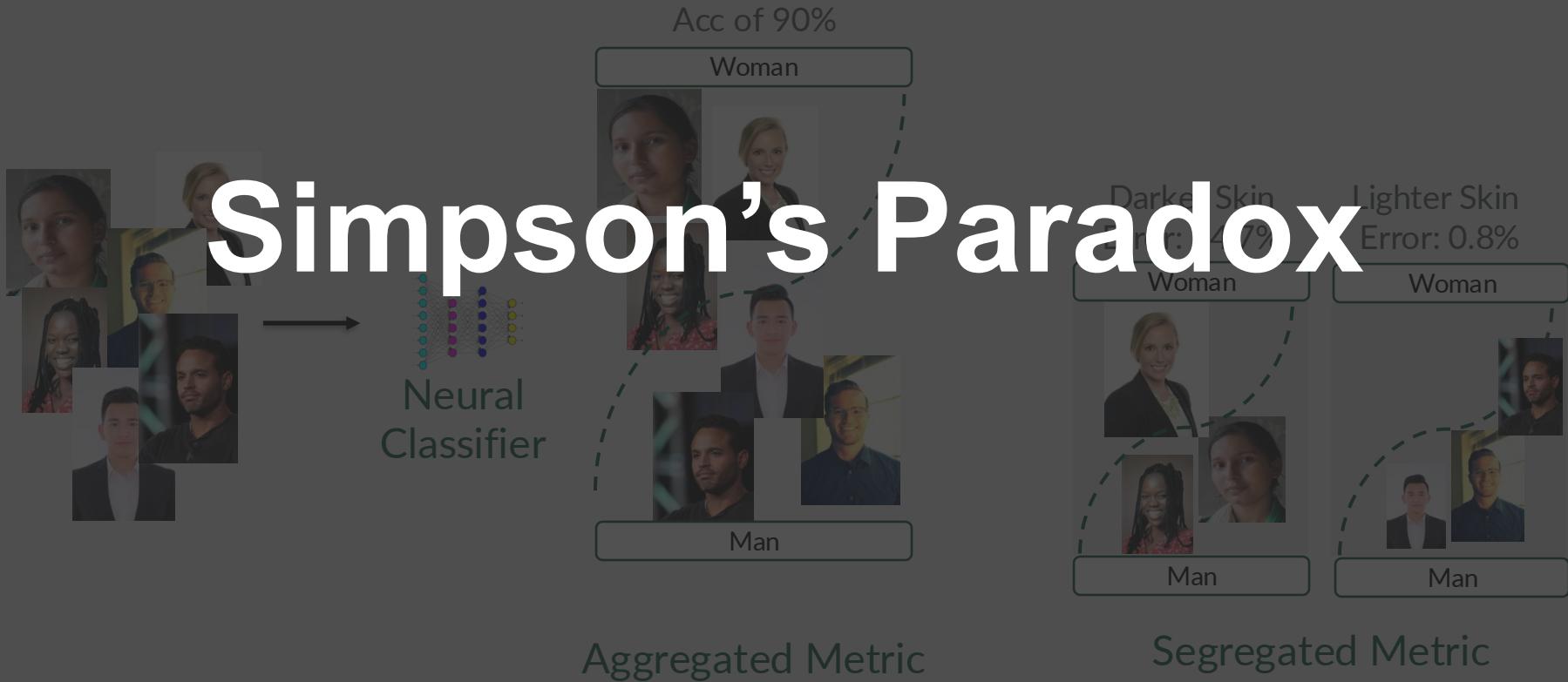
Disparities in gender classification

Segregated Metrics enhance the explanation of prediction performance



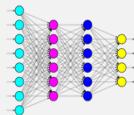
Aggregated Metrics make difficult for policymakers to determine the fairness and safety of a given AI system

Simpson's Paradox



Disparities in Code Generation

$$p(w_t | \mathbf{d}_t)$$



Sampling Snippets

```
1. def countChars(string, character):  
2.     count = 0  
3.     for letter in string:  
4.         if letter == character:  
5.             count = count + 1  
6.     return count
```

(Un)conditioned generated code

Aggregated Accuracy ~80%

Getters & Setters

Functional Programming



[For Statement]
or Cycles

Segregated Accuracy (varies)

70%

20%

55%



High Dimensional
Manifold Space of Code
Features

Acc. Segregated
by Code Feature

Confounding Variables

This **confounders** allow us to decompose the performance into meaningful clusters

$z = \{Skin\ Types\}$

Darker Skin Women
Error: 34.7%



Lighter Skin Men
Error: 0.8%

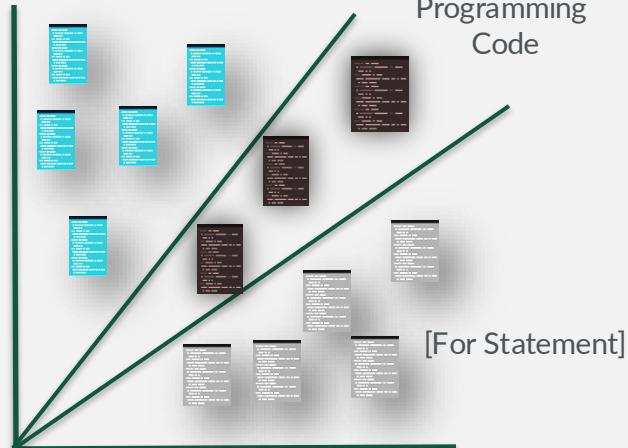


Gender Classification

$z = \{Code\ Features\}$

Getters & Setters

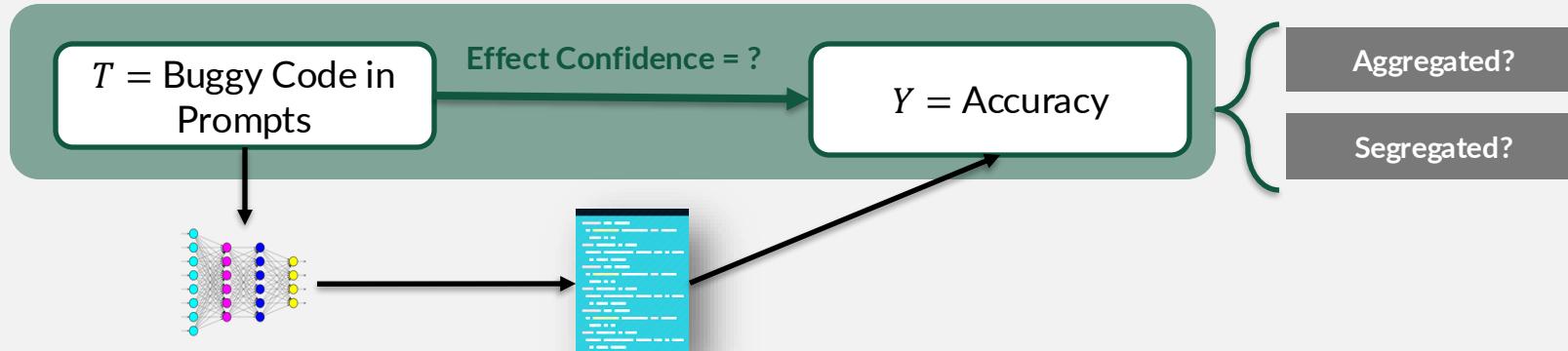
Functional Programming Code



Code Generation

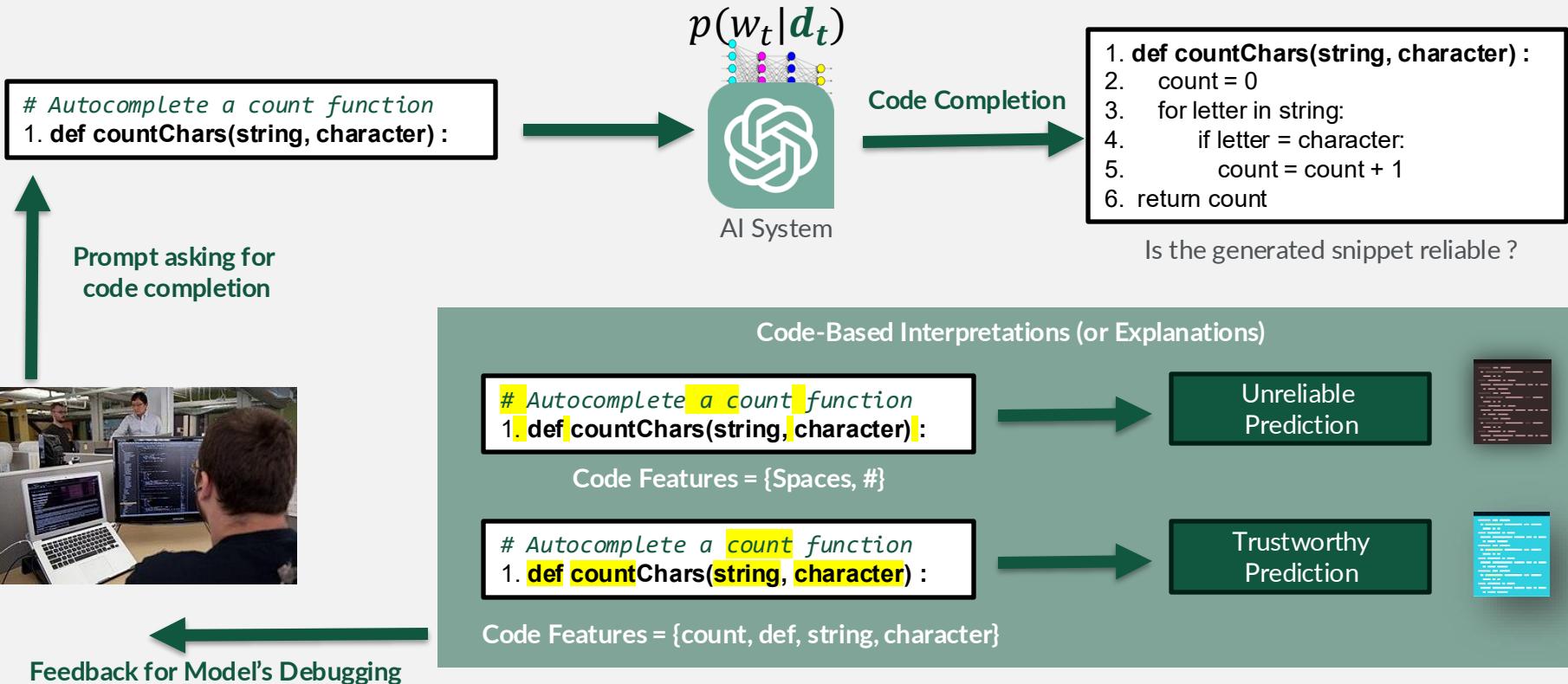
First Observation: Confounding Bias

Aggregated measures offers a **partial understanding** of the inference process of Neural Models. These measures and associated *confounders* can lead to a *misinterpretation* of an apparent relationship between two variables of interest (i.e., buggy prompt and model's accuracy).



Motivation: Why model understanding?

Can practitioners **trust** the code generated by AI Systems? How?



Motivation: Why model understanding?

Can practitioners trust the code generated by AI Systems? How?

```
# Autocomplete a count function  
1. def countChars(string, character):
```

$$p(w_t | d_t)$$



Code Completion

```
1. def countChars(string, character):  
2.     count = 0  
3.     for letter in string:  
4.         if letter == character:  
5.             count = count + 1  
6.     return count
```

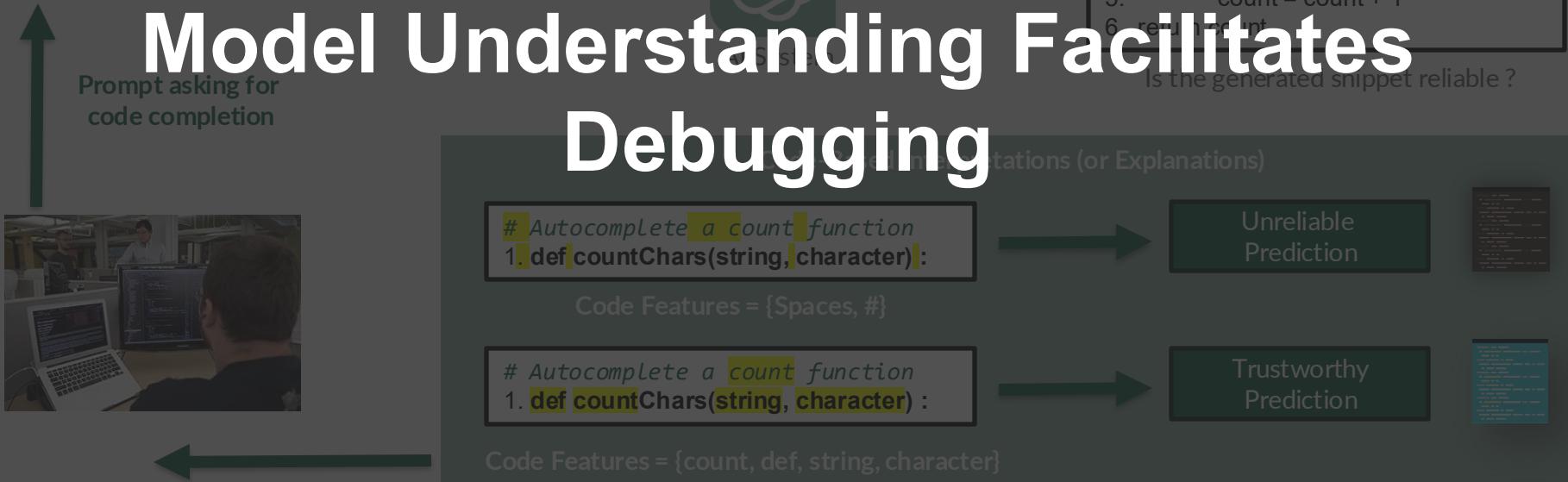
Is the generated snippet reliable ?

Model Understanding Facilitates Debugging

Prompt asking for code completion



Feedback for Model's Debugging



Interpretability Utility

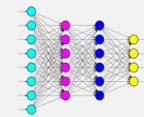
Model Understanding is critical in high stakes domains such as code generation since AI systems can impact lives of millions of individuals (e.g., health care, criminal justice)

Facilitating Debugging and Detecting Bias

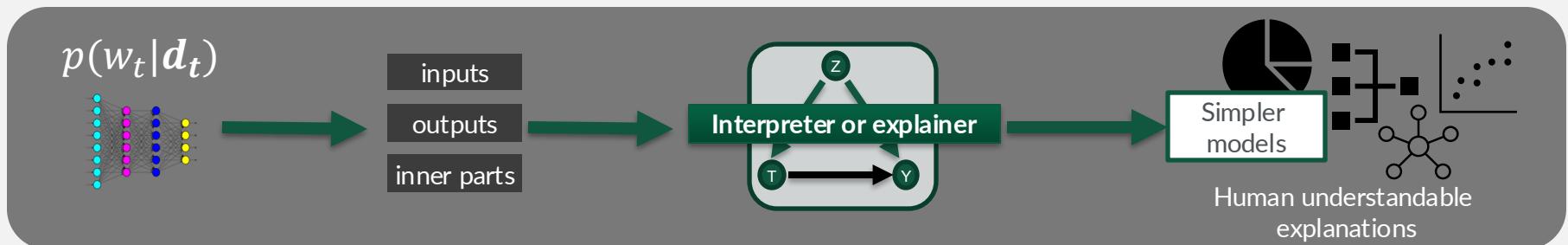
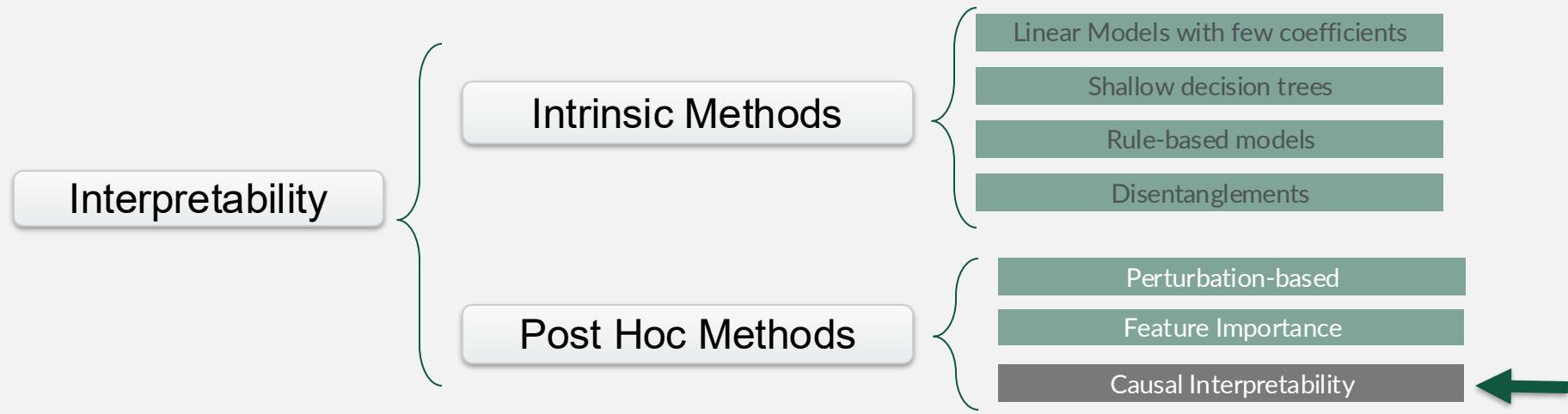
Providing recourse to practitioners who are negatively affected by predictions

Vetting models to determine if they are suitable for deployment in real scenarios

Assessing if and when to trust model predictions when making decisions

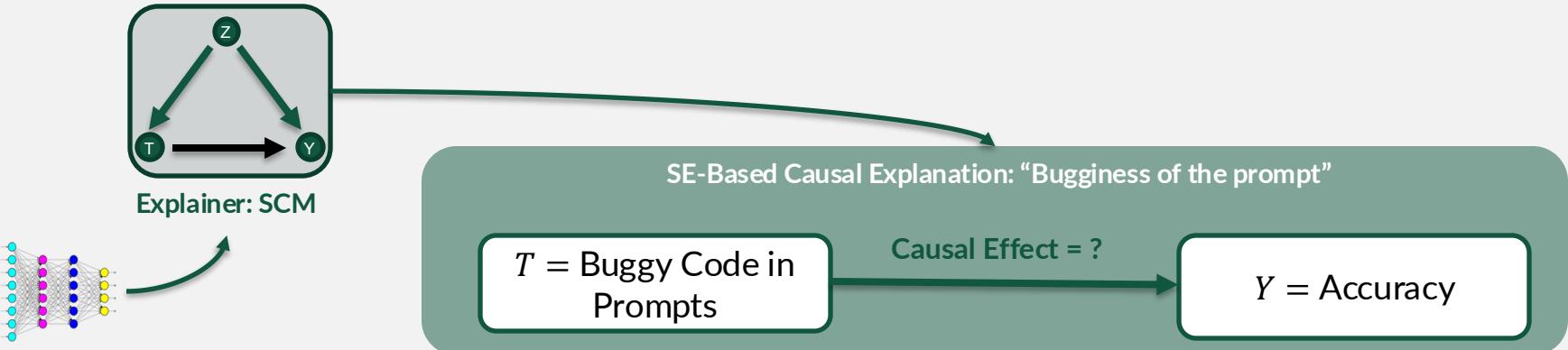


How do we attain model understanding or interpretability?



What is Causal Interpretability?

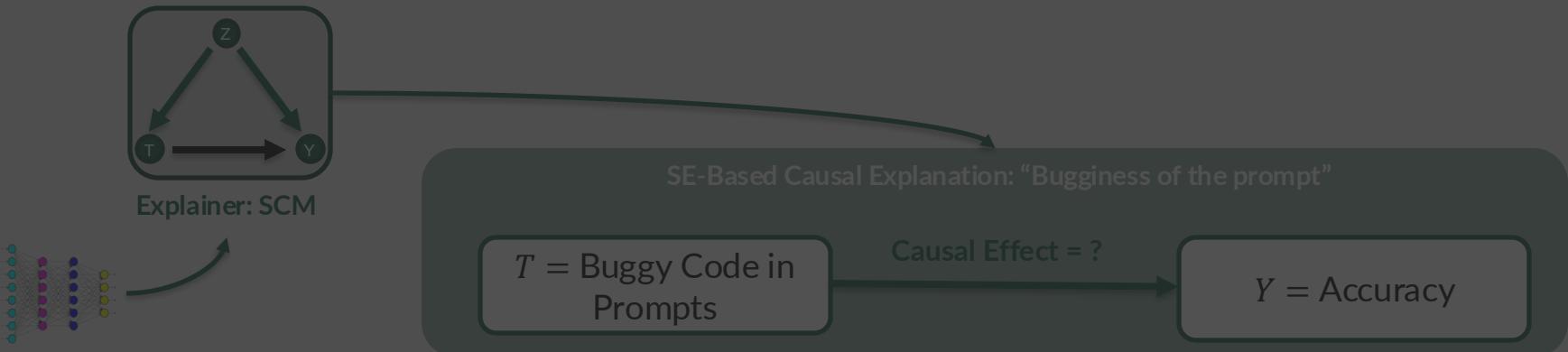
A post hoc global approach by which *Neural Code Models* are interpreted or explained from a causal assumption encoded in a Structural Causal Graph (SCM)



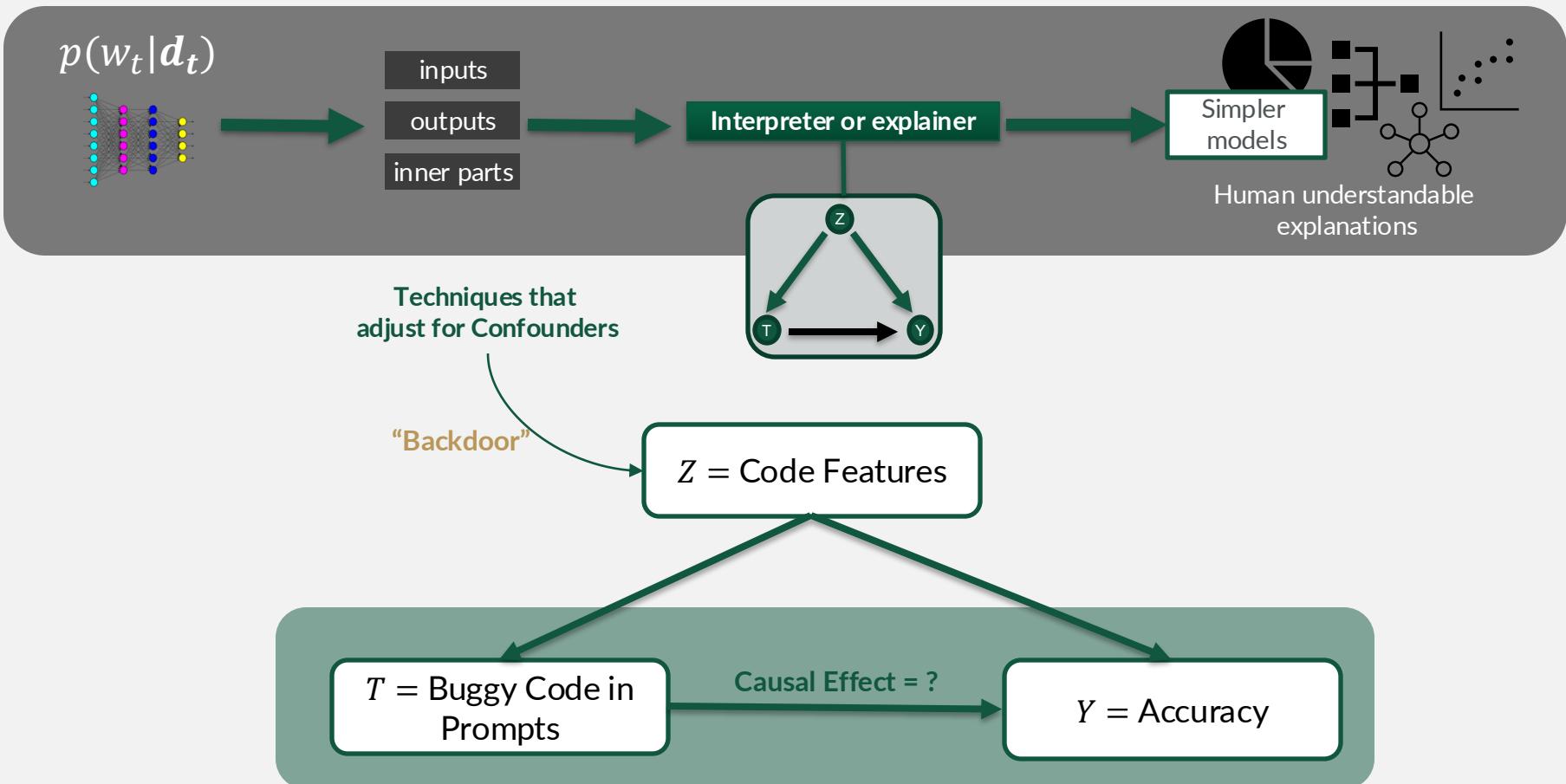
What is Causal Interpretability?

A post hoc global approach by which *Neural Code Models* are interpreted or explained from a causal assumption encoded in a Structural Causal Graph (SCM)

Why Causal Inference for Interpretability?

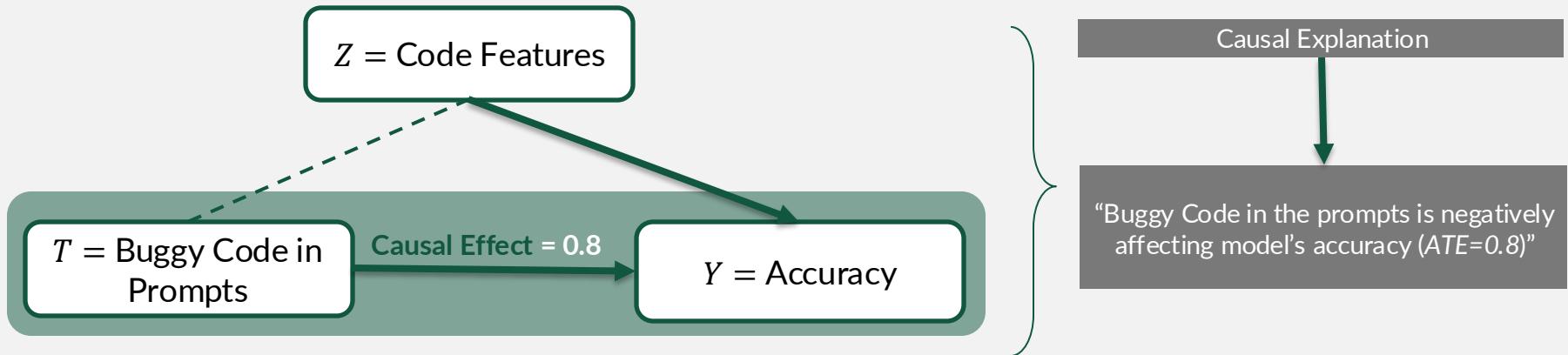


Causal Inference introduces techniques for reducing **confounding bias**

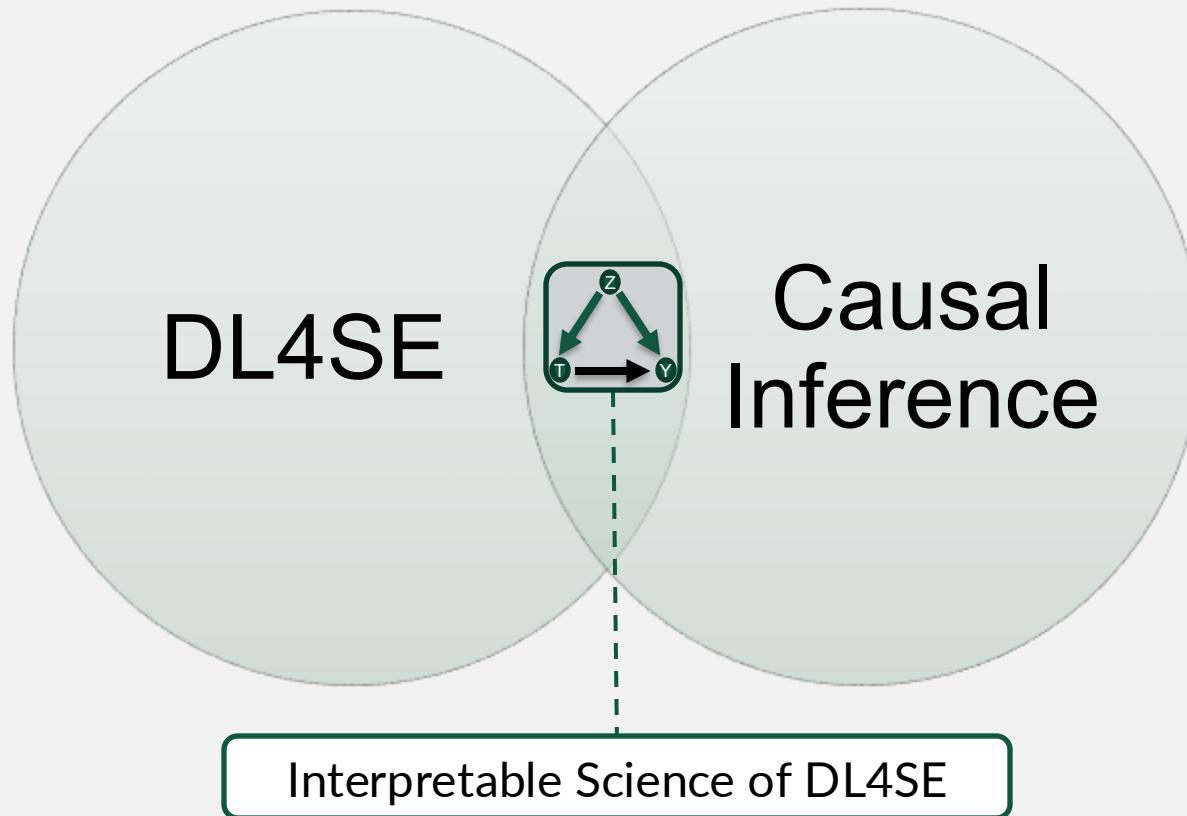


Second Observation: Causal Inference as a Tool for Interpretability

Causal Inference helps us to control for *confounding bias* by offering an explanation or interpretation to models' output.



In summary, this causal interpretability explores the **intersection** between



Talk Outline

Topic 1. Background

- At the intersection of DL4SE and Causal Inference

Topic 2. *do_{code}*

- A Causal Interpretability Approach

Topic 3. Applicability

- Case Study on Deep Code Generation

Topic 4. Code-Based Explanations

- Rationales and AST-based explanations for Foundation Models for Code

Topic 5. Discussion and Implications

- Challenges
- On the causal nature of Software Information
- Summary of Contributions

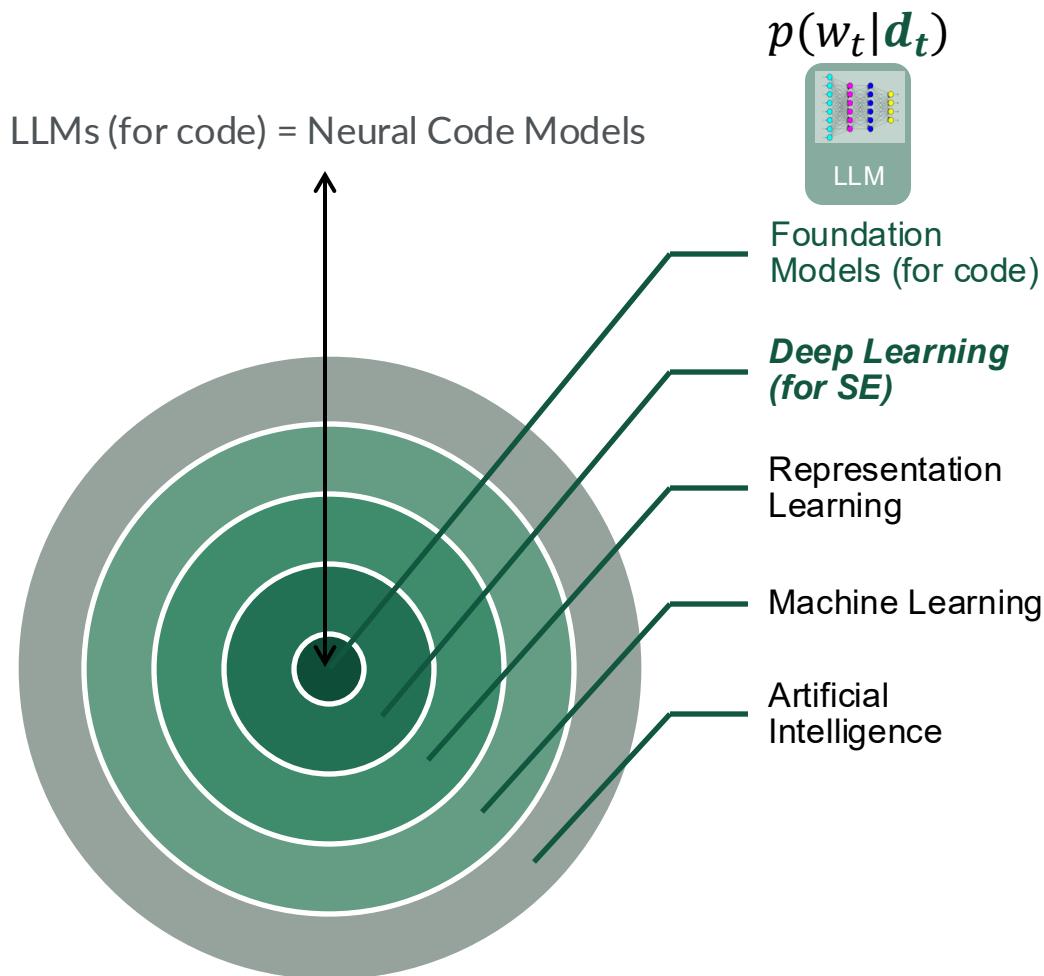
Topic 1 – Background

Deep Learning (4 SE) \cap Causal Inference

Deep Learning for Software Engineering

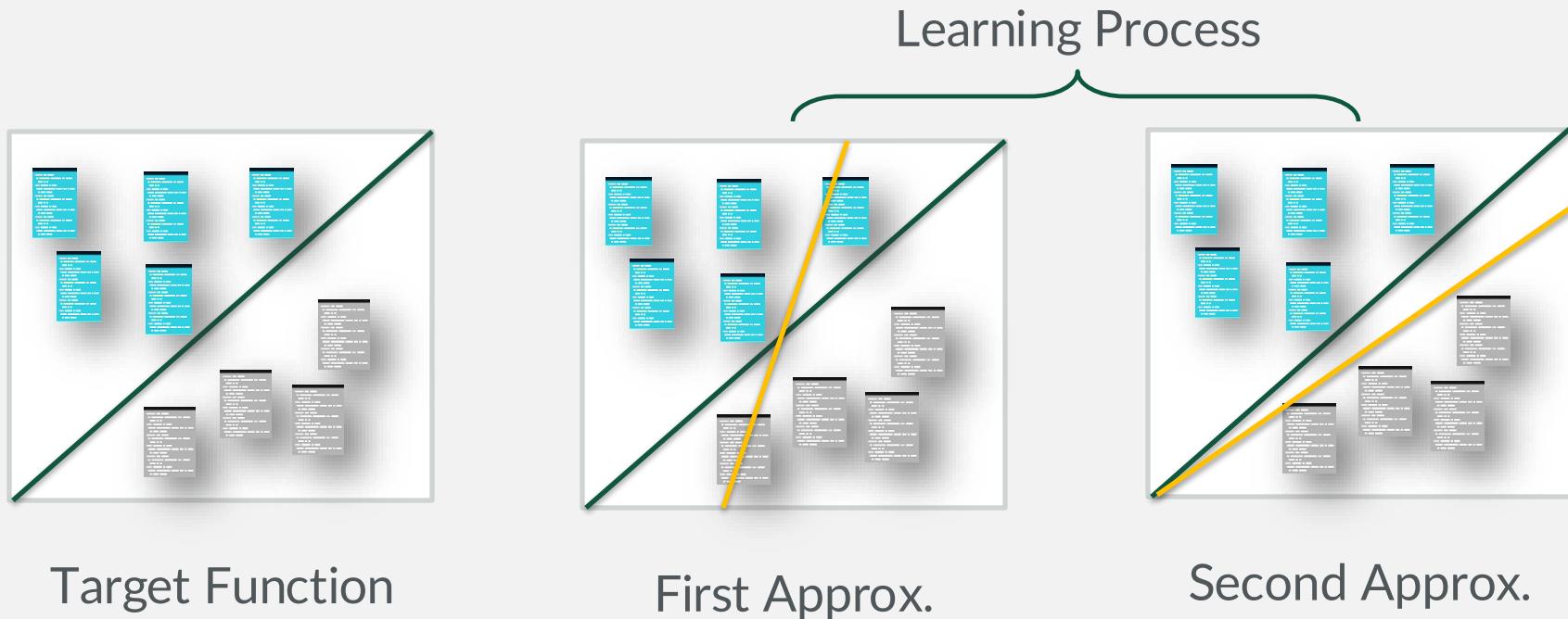


Hierarchy of Artificial Intelligence



The Computational Statistical Learning Process

Induction principle → empirical risk minimization (e.g., minimizing training error)



Target Function

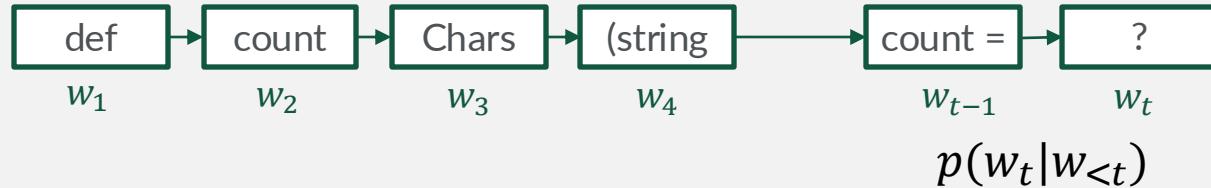
First Approx.

Second Approx.

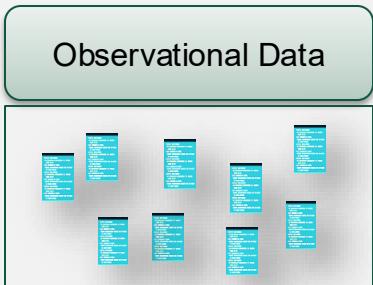
Deep Code Learning:

Neural Code Models Statistically Learn Patterns From Code

```
1. def countChars(string, character) :  
2.     count = 0  
3.     for letter in string:  
4.         if letter == character:  
5.             count = count + 1  
6.     return count
```

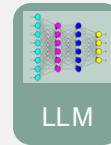


Code Data



Statistical Learning

Code Data
(inputs, outputs)



Probability Model
(e.g., Transformer or RNN)

$p(w_t | d_t)$
Observational
Distribution

$$p(w_t | w_{<t}) \approx p(w_t | d_t)$$

Some Applications of **Neural Code Models**

**Assert Statements
Generation**
(Watson, et al., ICSE'20)

Automatic Bug Fixing
(Tufano, et al., TOSEM'19)

**Learning Code
Changes**
(Tufano, et al; ICSE'19)

Clone Detection
(White, et al., ASE'16)

**Learning to Identify
Security
Requirements**
(Palacio, et al., ICSME'19)

Some Applications of **Neural Code Models**

Deep Code
Generation

Automatic Bug Fixing
(Tufano, et al., TOSEM'19)

Learning Code
Changes
(Tufano, et al; ICSE'19)

Assert Statements
Generation
(Watson, et al., ICSE'20)

Clone Detection
(White, et al., ASE'16)

Learning to Identify
Security
Requirements
(Palacio, et al., ICSME'19)

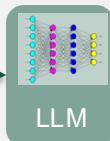
Deep Code Generation

Prompt

```
1. import os  
2. import requests  
3.  
4. URL = "https://www.organization.com/api/fac/1"  
5.  
6. # Generate a Request of Data Organization
```

Generated Code

```
6. # Generate a Request of Data Organization  
7. API_KEY = 'https://thisisaprivatekey.sss.org'  
8.  
9. headers = {"Authorization": "Api-Key " + API_KEY}  
10. response = requests.get(URL, headers=headers)  
11. data = response.json()["data"][0]  
12. print(data)
```



Neural Code Model



Why?

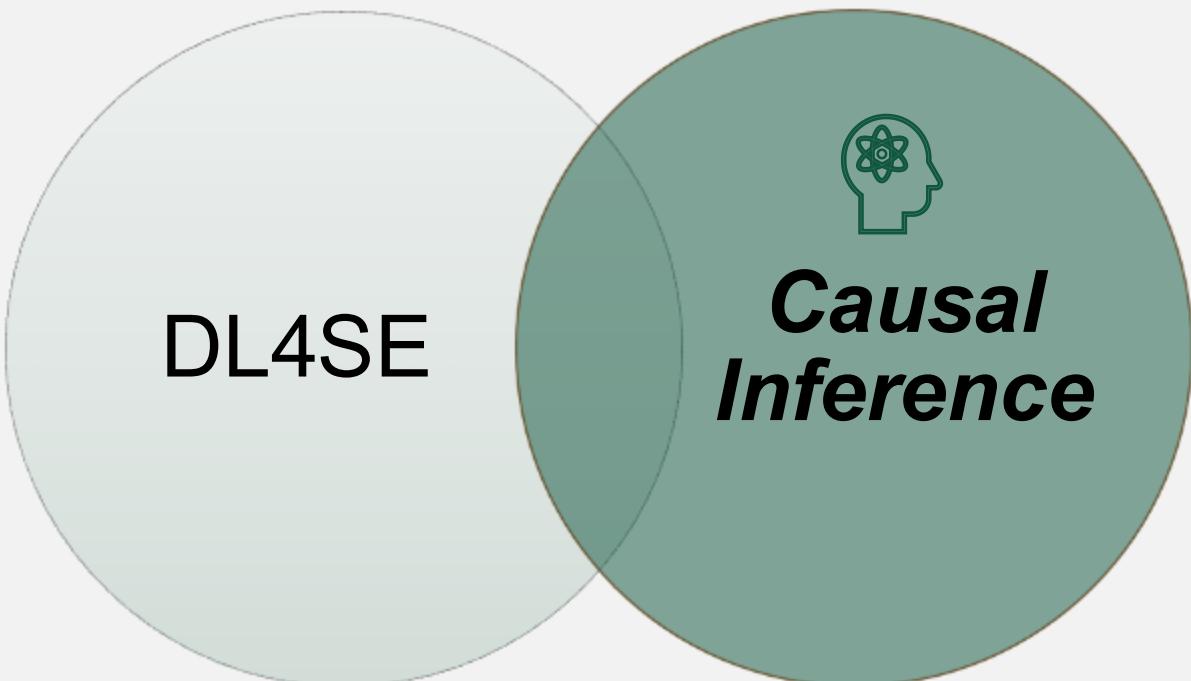
Is the generated output reliable?

Code BLEU

Accuracy

Perplexity

Canonical Evaluation



DL4SE

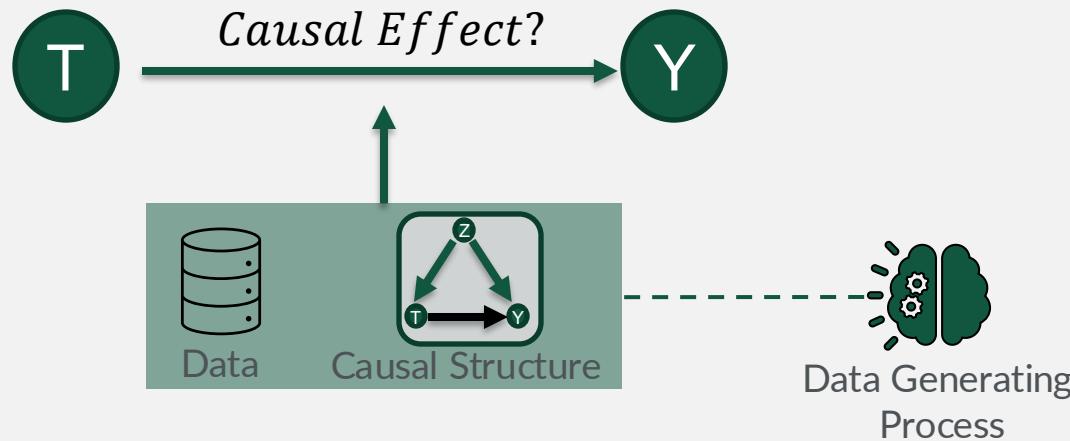
*Causal
Inference*



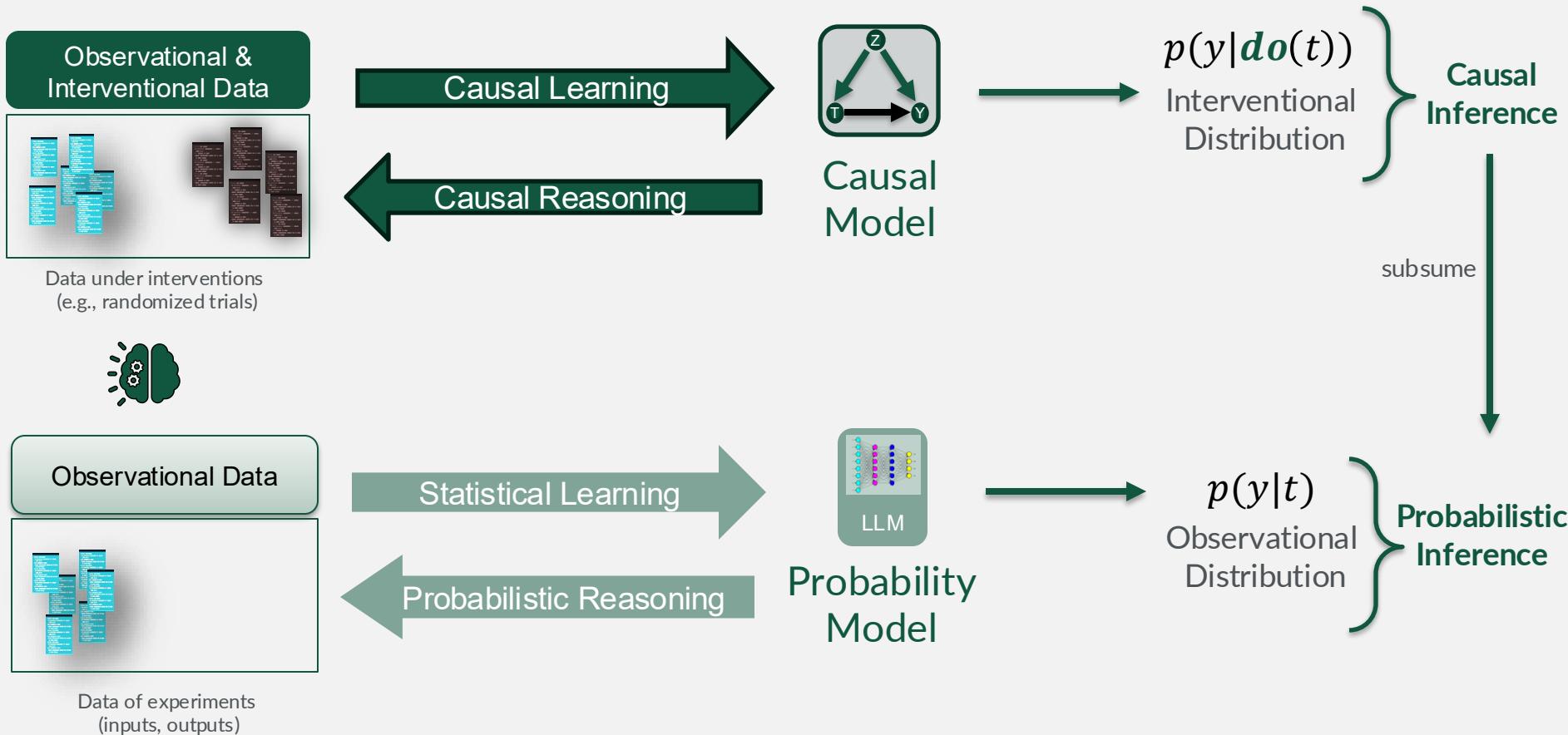
What is Causality?

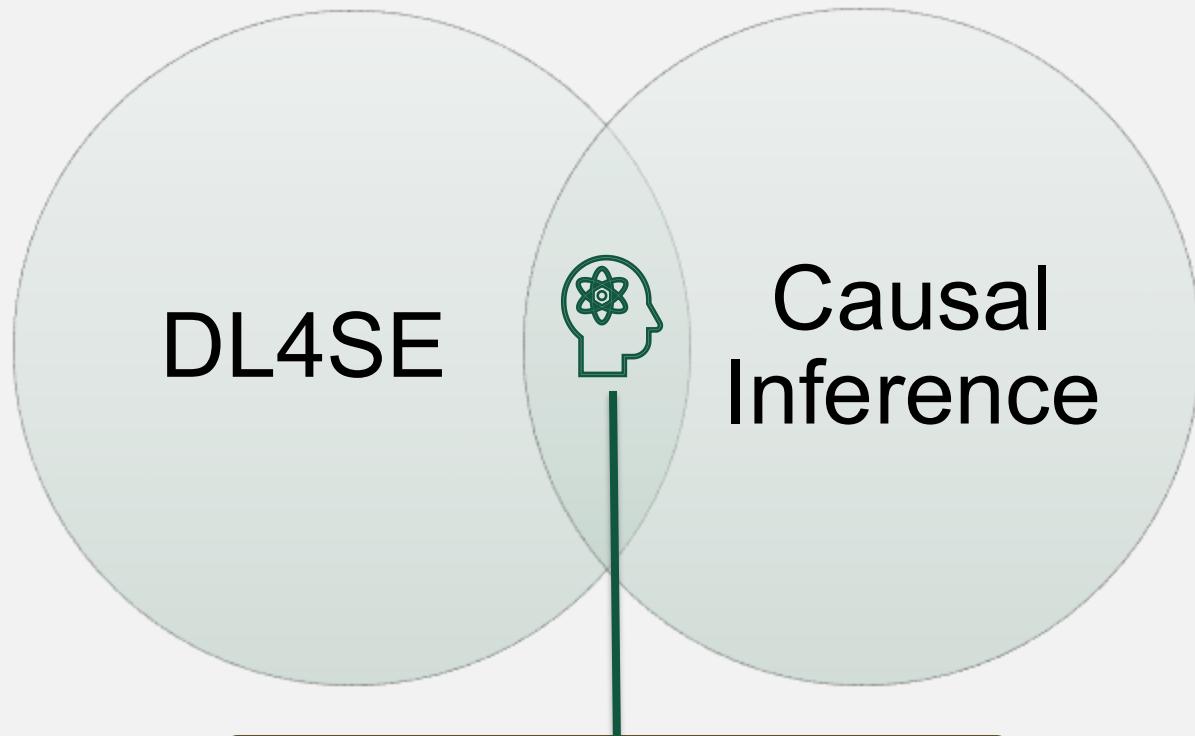
Judea Pearl Definition:
“T is a cause of Y if Y *listens to T*”

Causal Inference is a process of estimating causal effect from observational data



Background in Learning Theory





Interpretable Science of DL4SE

Deep Code Generation

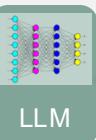
Prompt

```
1. import os  
2. import requests  
3.  
4. URL = "https://www.organization.com/api/fac/1"  
5.  
6. # Generate a Request of Data Organization
```

Data Leakage?

Generated Code

```
6. # Generate a Request of Data Organization  
7. API_KEY = 'https://thisisaprivatekey.sss.org'  
8.  
9. headers = {"Authorization": "Api-Key " + API_KEY}  
10. response = requests.get(URL, headers=headers)  
11. data = response.json()["data"][0]  
12. print(data)
```



Why?



Code BLEU

Accuracy

Perplexity

How can we formulate a causal question?

Interpretability as incompleteness in problem formalization

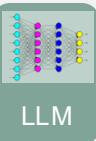
Prompt

```
1. import os  
2. import requests  
3.  
4. URL = "https://www.organization.com/api/fac/1"  
5.  
6. # Generate a Request of Data Organization
```

Data Leakage?

Generated Code

```
6. # Generate a Request of Data Organization  
7. API_KEY = 'https://thisisaprivatekey.sss.org'  
8.  
9. headers = {"Authorization": "Api-Key " + API_KEY}  
10. response = requests.get(URL, headers=headers)  
11. data = response.json()["data"][0]  
12. print(data)
```



Why?



Code BLEU

Accuracy

Perplexity

We experience an incompleteness in (DL4SE) problem formalization, making the adoption of NCM unreliable (less trustworthy).

How can we formulate a causal question?

What is **Interpretability**?

“Interpretability is a research field that makes machine learning systems and their decision-making process understandable to human beings” (Doshi-Velez and Kim, 2017)

Causal Interpretability for Code Generation

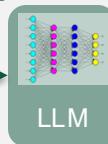
Prompt

```
1. import os  
2. import requests  
3.  
4. URL = "https://www.organization.com/api/fac/1"  
5.  
6. # Generate a Request of Data Organization
```

Data Leakage?

Generated Code

```
6. # Generate a Request of Data Organization  
7. API_KEY = 'https://thisisaprivatekey.sss.org'  
8.  
9. headers = {"Authorization": "Api-Key " + API_KEY}  
10. response = requests.get(URL, headers=headers)  
11. data = response.json()["data"][0]  
12. print(data)
```



How is the generated code $< Y >$ related to code concepts/tokens in the prompt $< T >$?

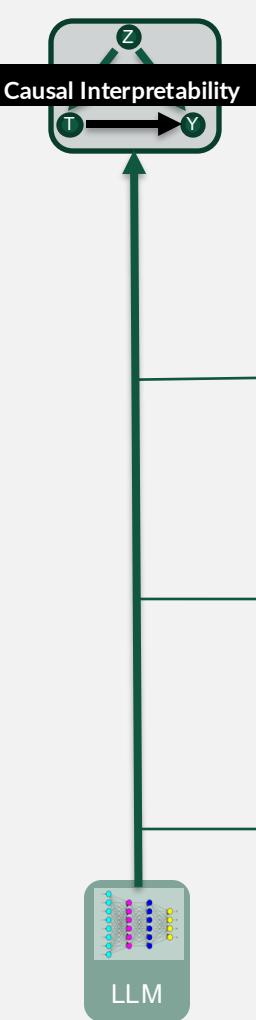
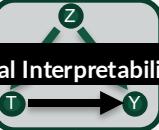
$$Q_1 = P(Y|T)$$

A mathematical language is required to formulate causal queries for code generation

Pearl's Ladder of Causation

Deep Learning for Software Engineering Queries

Causal Interpretability



Level 3.
Counterfactual

$$Q_3 = P(y_t|t', y')$$

Counterfactual Distribution

Would the LLM generate API Keys if URLs had been removed from training sets y_t?

Level 2.
Intervention

$$Q_2 = P(Y|do(T))$$

Interventional Distribution

If we remove URLs from the prompt T, will the LLM generate API Keys (data leakage) Y?

Level 1.
Association

$$Q_1 = P(Y|T)$$

Observational Distribution

How is the generated code Y related to code concepts in the prompt T?

The Causal Interpretability **Hypothesis**

do_{code} is a causal interpretability method that aims to make DL4SE systems (i.e., NCMs) and their decision-making process understandable for researchers and practitioners

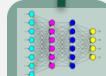
Pearl's Ladder of Causation

Deep Learning for Software Engineering Queries

Causal Interpretability



do code



LLM

Level 3.
Counterfactual

$$Q_3 = P(y_t|t', y')$$

Counterfactual Distribution

Would the LLM generate API Keys if URLs had been removed from training sets < y_t >?

Level 2.
Intervention

$$Q_2 = P(Y|do(T))$$

Interventional Distribution

If we remove URLs from the prompt < T >, will the LLM generate API Keys (data leakage) < Y >?

Level 1.
Association

$$Q_1 = P(Y|T)$$

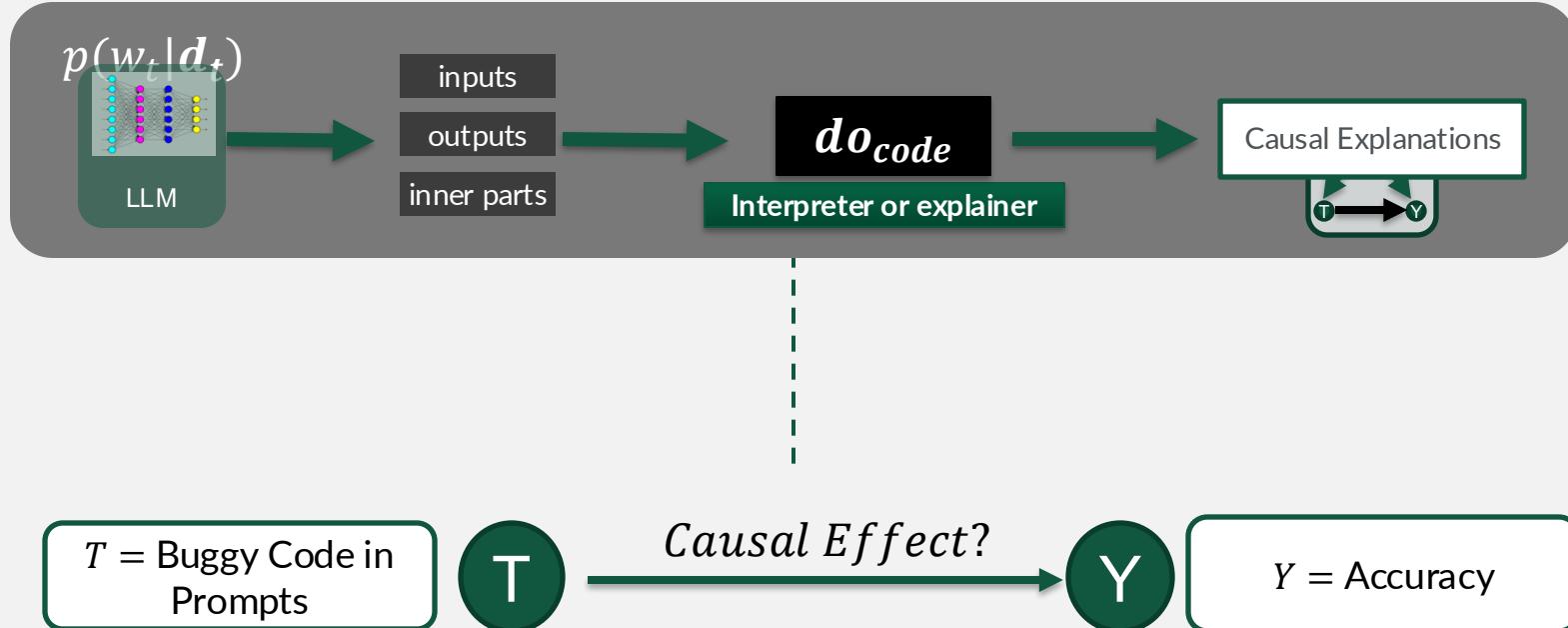
Observational Distribution

How is the generated code < Y > related to code concepts in the prompt < T >?

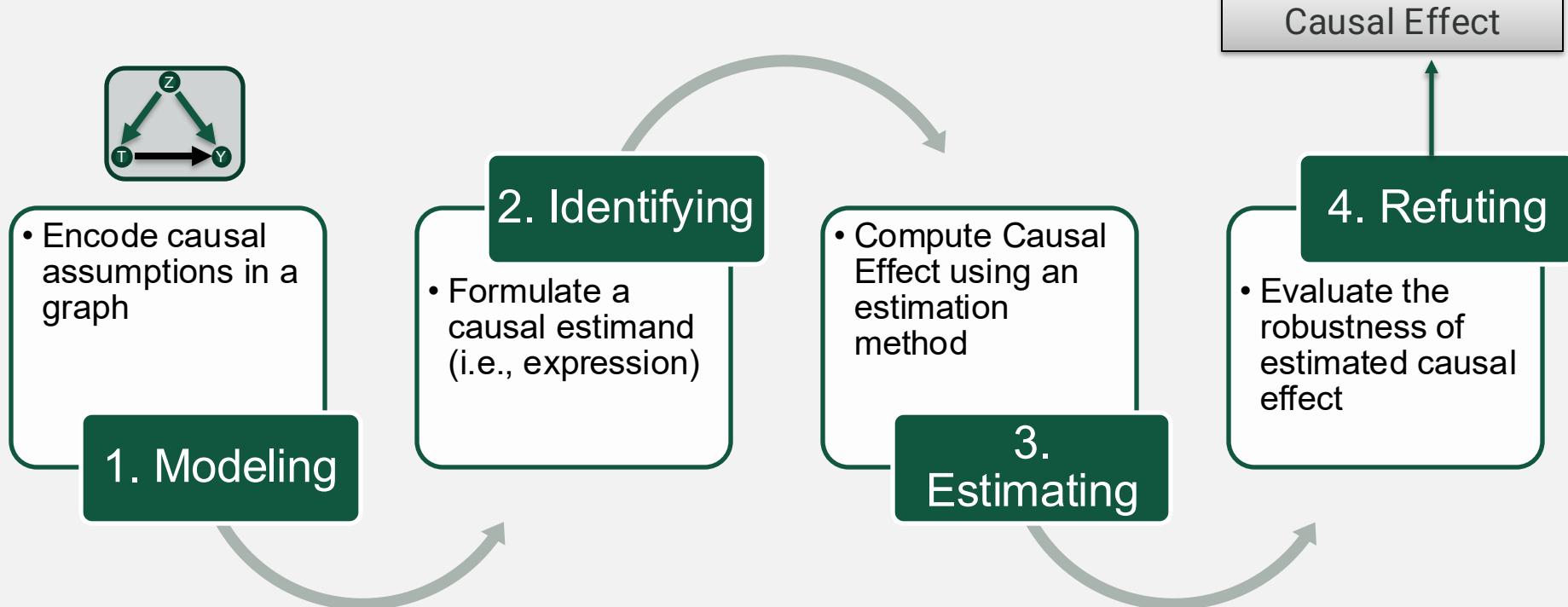
Topic 2 – *do_{code}*

A Causal Interpretability Approach for Deep Code Generation

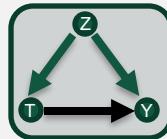
do_{code} generates causal explanations



do_code Interpretability Pipeline



Step 1: Modeling



- Encode causal assumptions in a graph

1. Modeling

Step 1: Modeling

Graph Criteria

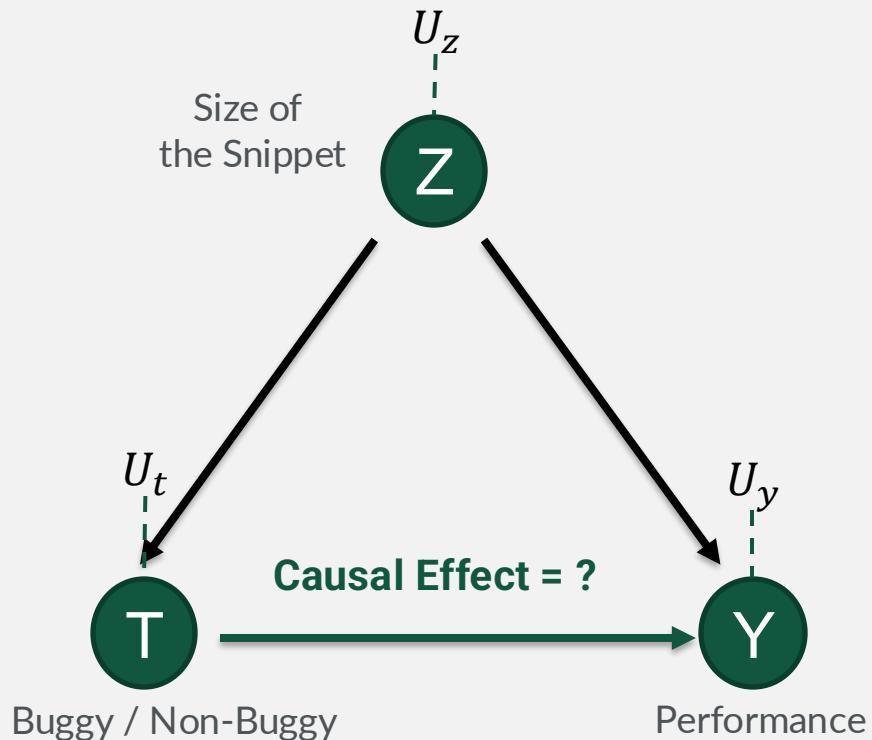
Treatment T:
Proposed SE intervention (e.g., Buggy Prompt)

Potential Outcome Y:
Code Prediction (e.g., loss, accuracy, token)

Confounder Z:
Code Features (e.g., cyclo, subwords,
keywords)

Noise or Exogenous Variables U

Structural Causal Model



Step 1: Modeling

Graph Criteria

Treatment T:
Proposed SE intervention

$T_A = \text{BuggyCode}$

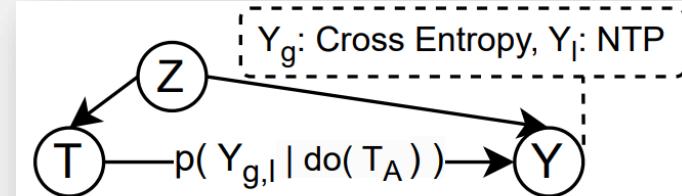
$T_B = \text{UnComments}$

$T_C = \text{Clones Type II}$

$T_D = \text{Clones Type III}$

Binary
Treatment

Structural Causal Model



Control (TA-0)

```
public Integer getMinElement(List myList) {  
    if(myList.size() >= 1) {  
        return ListManager.min(myList);  
    }  
    return null; }
```

Treatment (TA-1)

```
public Integer getMinElement(List myList) {  
    if(myList.size() >= 0) {  
        return ListManager.getFirst(myList);  
    }  
    return 0; }
```

Step 1: Modeling

Graph Criteria

Treatment T:
Proposed SE intervention

Binary
Lineal

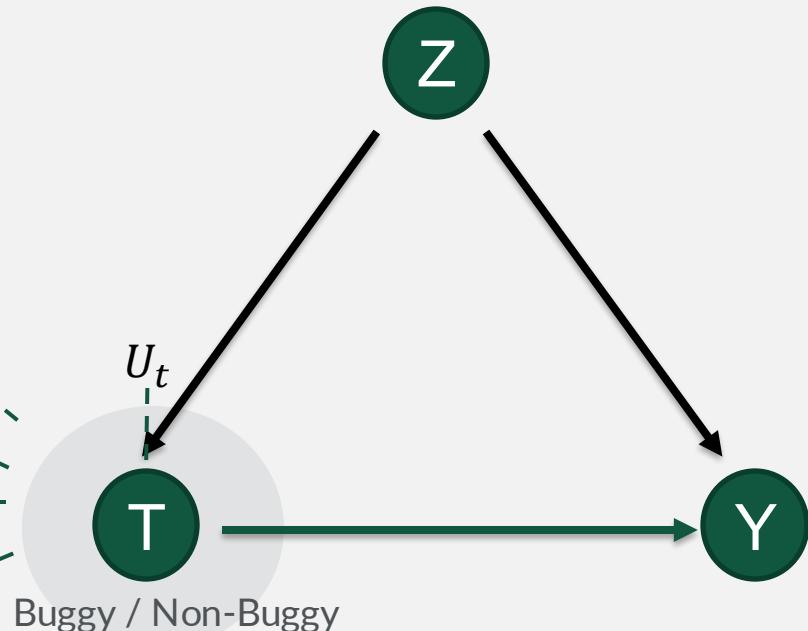
$T_A = \text{BuggyCode}$

$T_B = \text{UnComments}$

$T_C = \text{Clones Type II}$

$T_D = \text{Clones Type III}$

Structural Causal Model



Step 1: Modeling

Graph Criteria

Treatment T:
Proposed SE intervention (e.g., Buggy Prompt)

Potential Outcome Y:
Code Prediction (e.g., loss, accuracy, token)

$Y_g = \text{Global}$

$Y_l = \text{Local}$



Structural Causal Model

Step 1: Modeling

Graph Criteria

Treatment T:
Proposed SE intervention (e.g., Buggy Prompt)

Potential Outcome Y:
Code Prediction (e.g., loss, accuracy, token)

Confounder Z

McCabe's complexity

Variables

Line of Code

Max nested blocks

Lambda Expressions

Modifiers

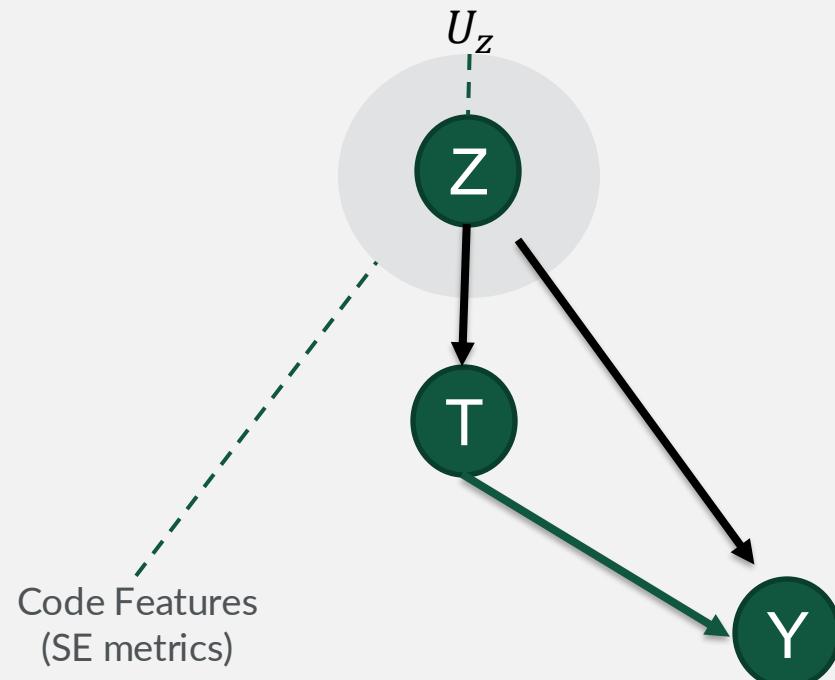
Returns

Unique Words

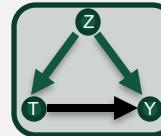
Comparisons

Try-Catch

Structural Causal Model



Step 2: Identifying



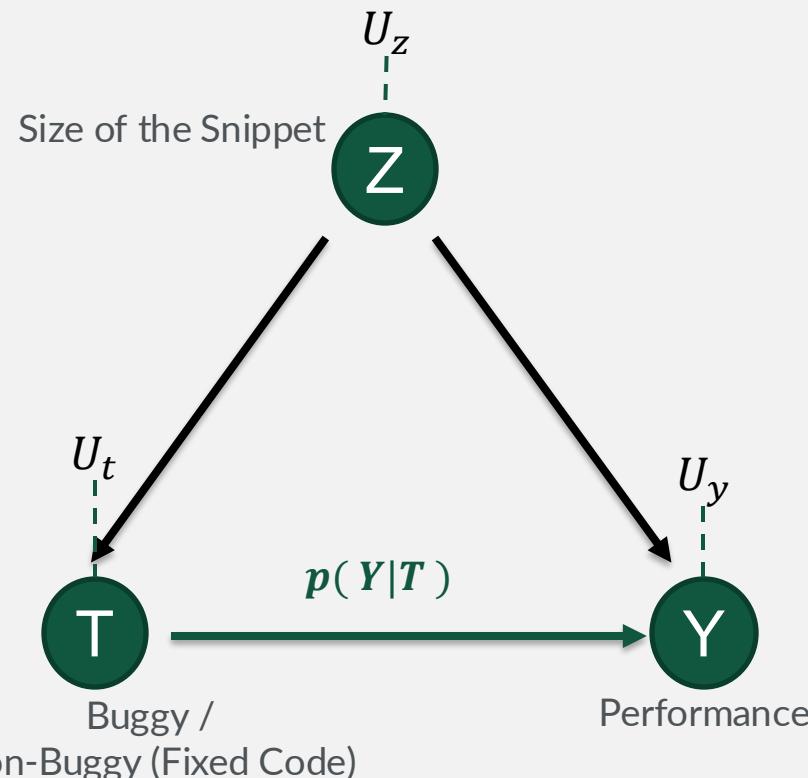
2. Identifying

- Formulate a causal estimand (i.e., expression)

1. Modeling

- Encode causal assumptions in a graph

Step 2: Identifying



Observational Distribution

$$p(Y \mid t = \text{FixedCode}) = \sum_z p(Y|t, z)p(z|t)$$

Step 2: Identifying

Observational Distribution

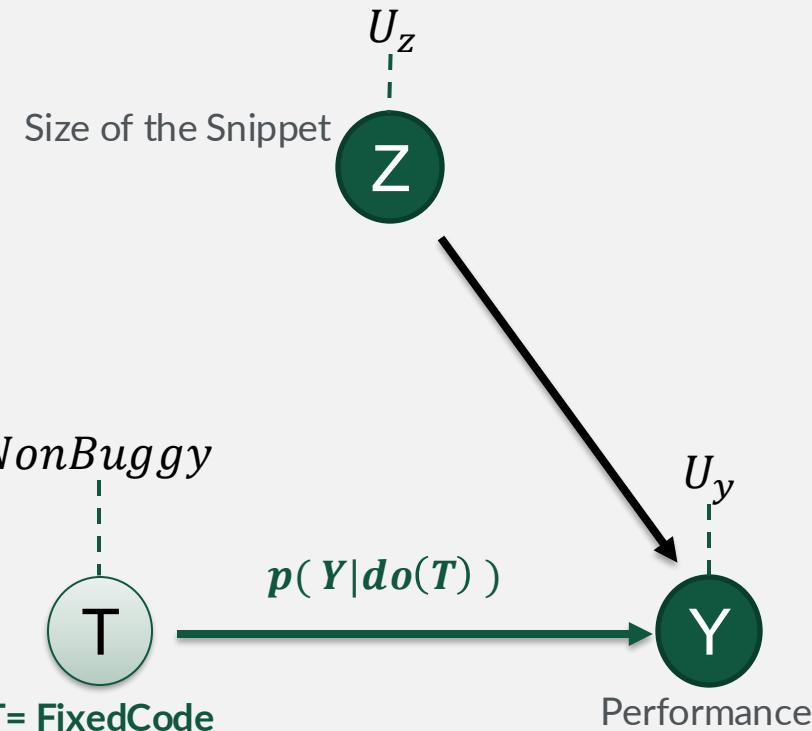
$$p(Y | t = \text{FixedCode}) = \sum_z p(Y|t, z)p(z|t)$$

do_{code}

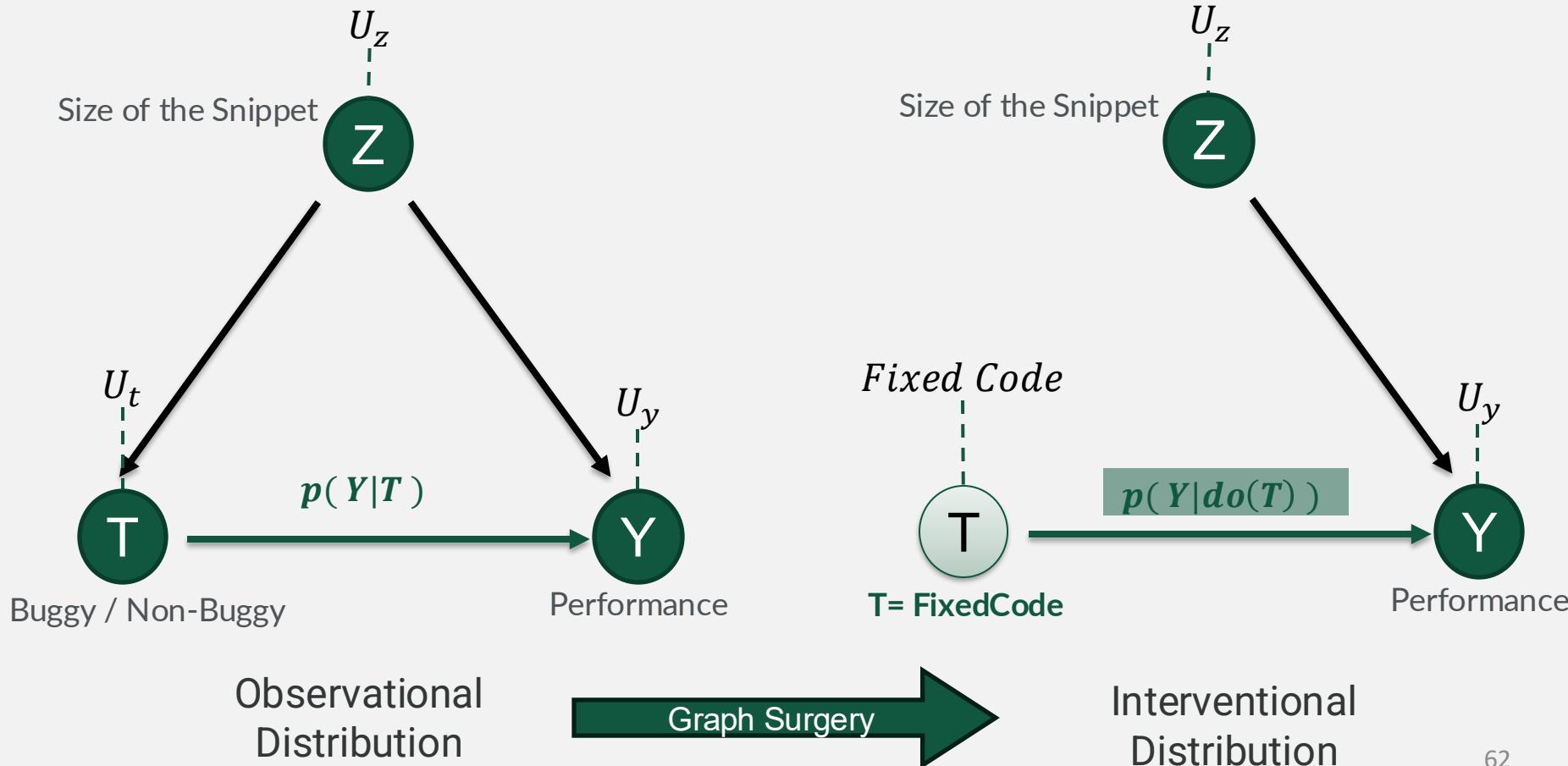
Interventional Distribution

$$p(Y | \text{do}(t = \text{FixedCode})) = \sum_z p(Y|t, z)p(z)$$

Adjustment Formula



Step 2: Identifying



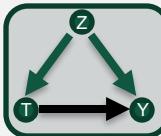
Step 3: Estimating

- Encode causal assumptions in a graph

1. Modeling

2. Identifying

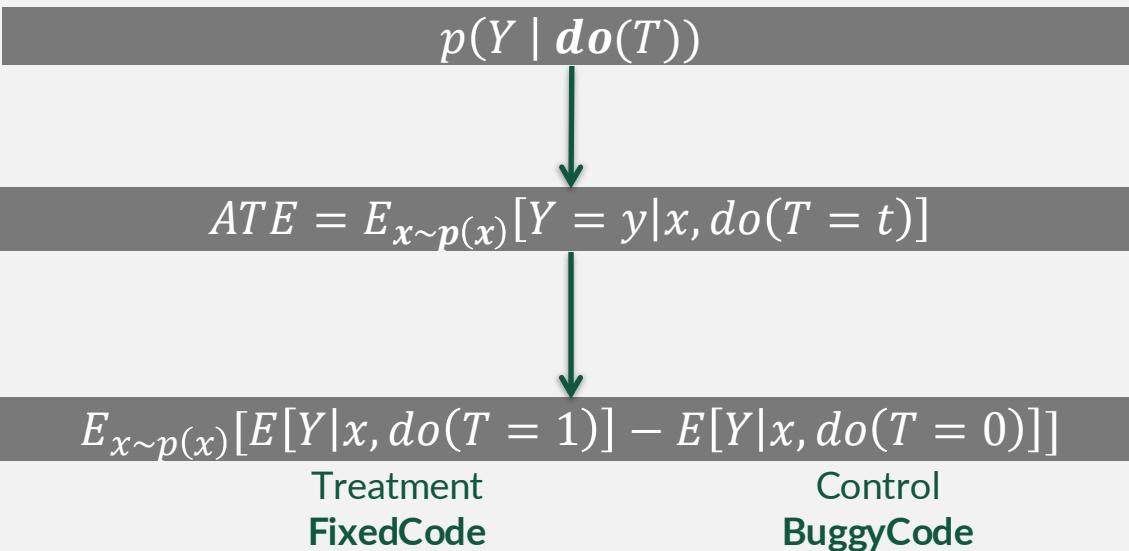
- Formulate a causal estimand (i.e., expression)



- Compute Causal Effect using an estimation method

3. Estimating

Step 3: Estimating



Interventional Distribution
(one sample)

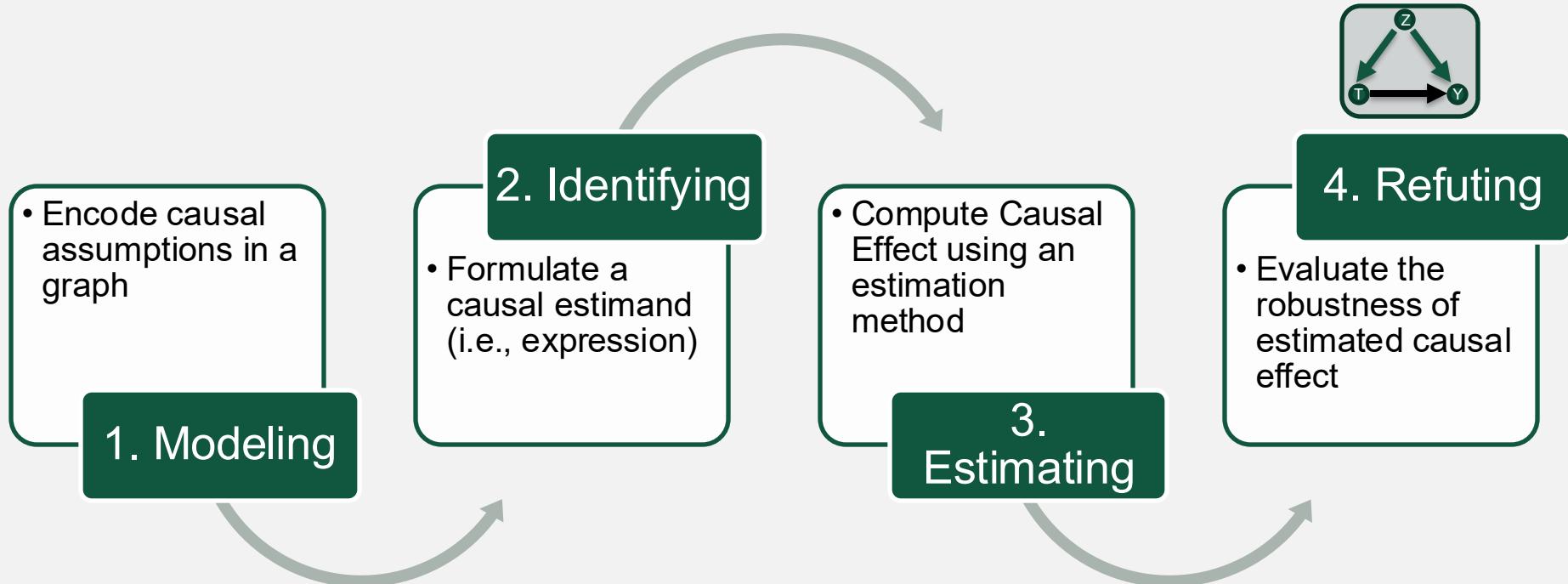
Average Treatment Effect
(across snippets)

Expected Value of
Potential Outcome

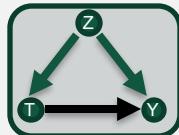
Both terms are quantities that can be estimated
from data

- Propensity Score Matching
- Linear Regression
- Machine Learning

Step 4: Refuting



Step 4: Refuting



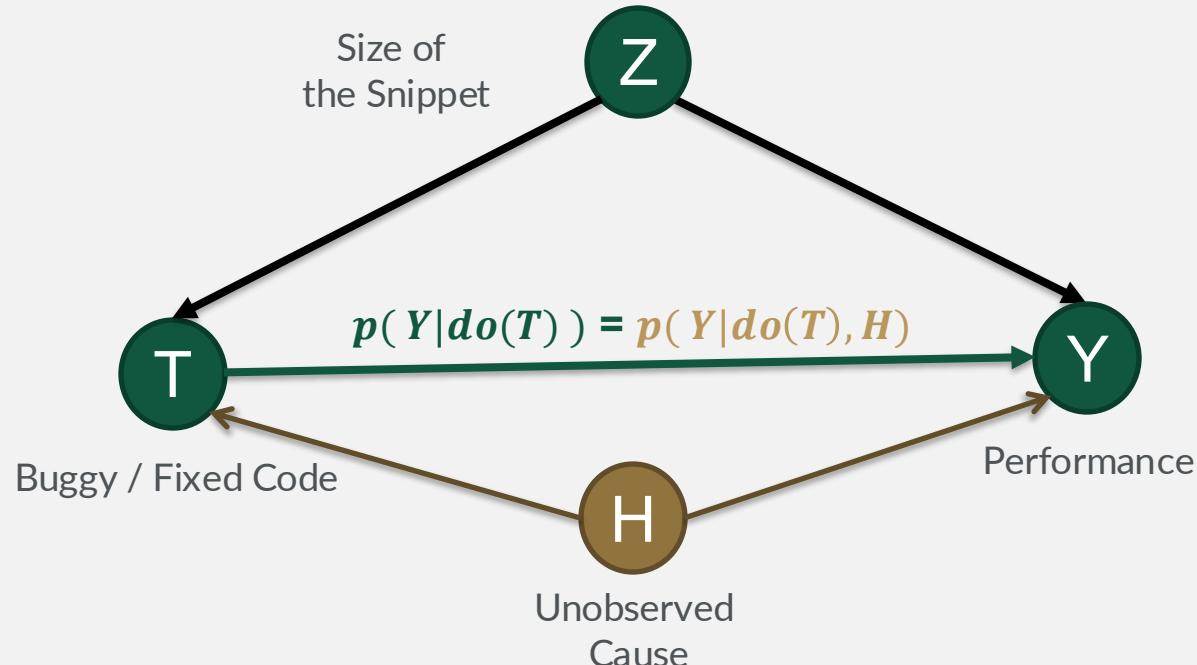
How can we falsify our assumptions/hypothesis?
Is the ATE estimated correct or valid?

Add Unobserved
Common Cause

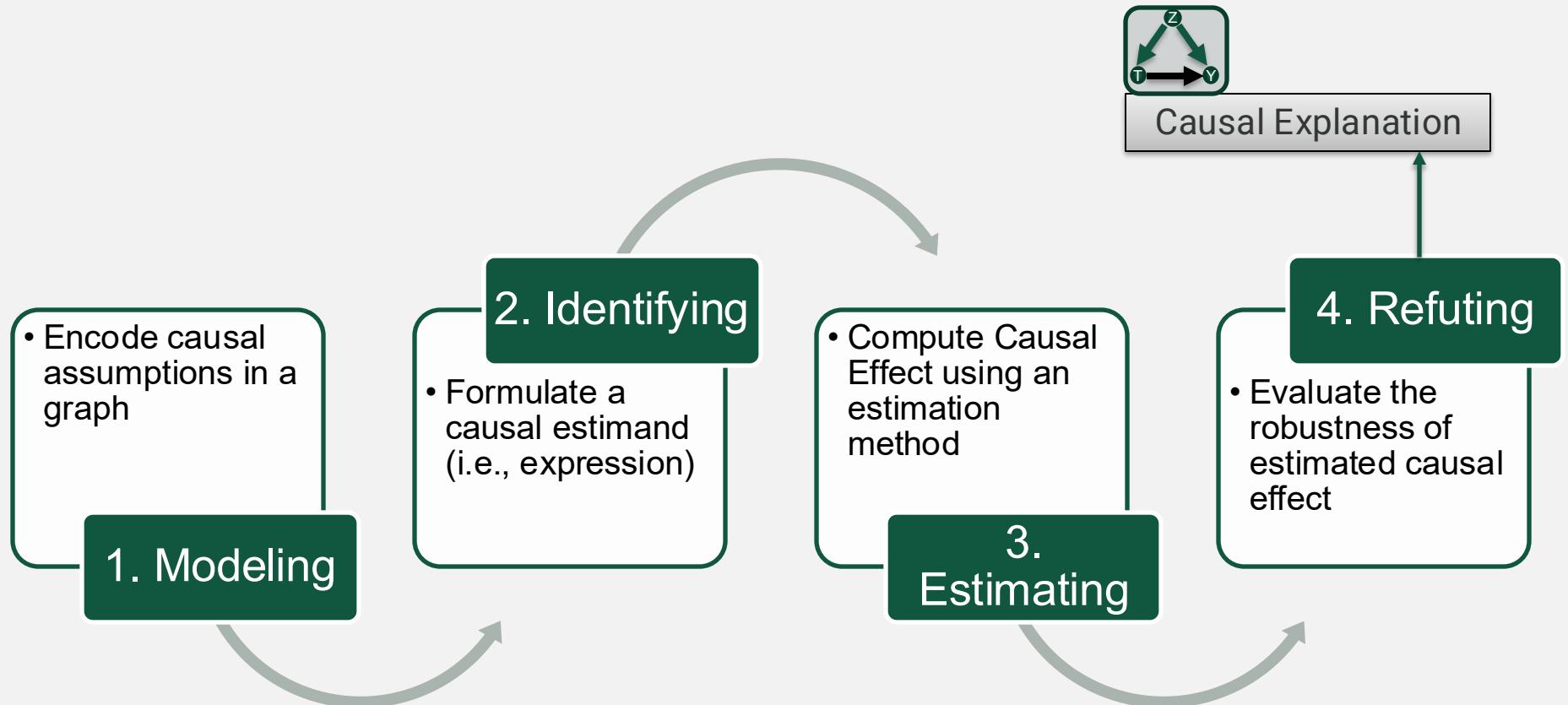
Add Random
Common Cause

Placebo Treatment

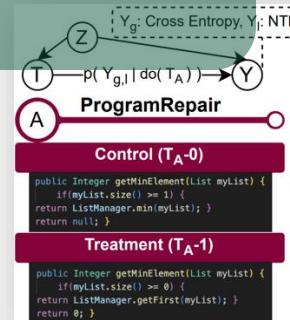
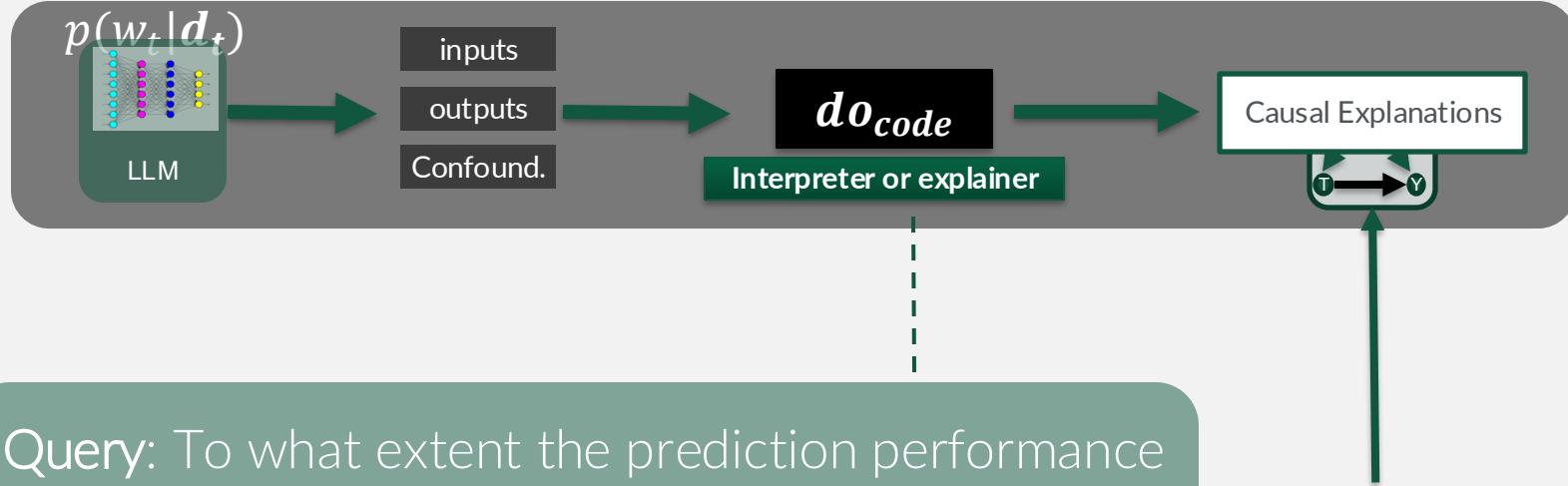
Data Subset
Validation



Step 5: Causal Explanations



Step 5: Causal Explanations

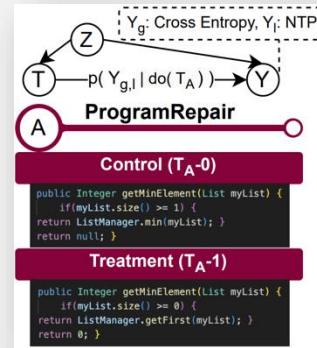


Topic 3 – Applicability

A Case Study on Deep Code Generation

Goal: To evaluate the feasibility of do_{code}

- ProgramRepair
- UnCommenting
- Semantic Preserving
- Node Type Masking
- Hyper-parameters

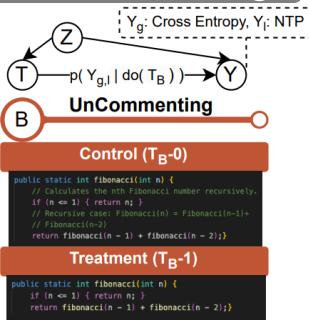


Feasibility?

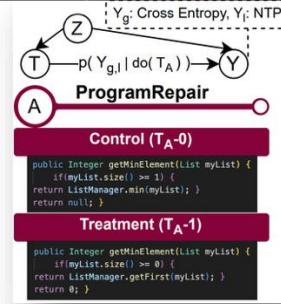


Proposed 7 SE-Based Treatments

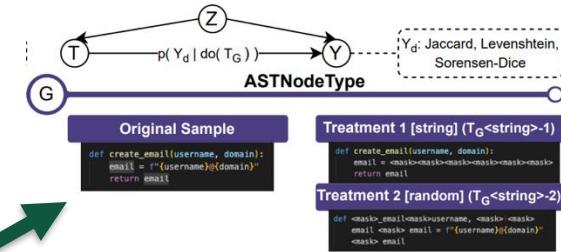
UnCommenting



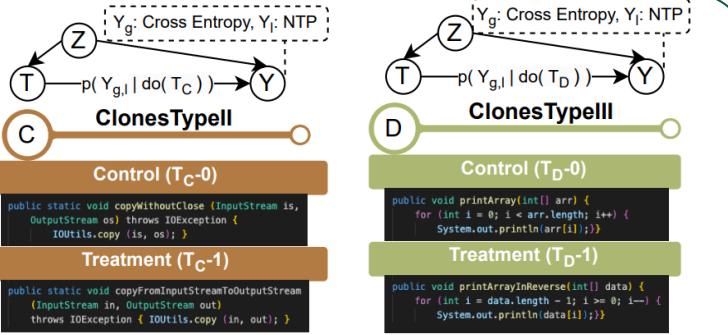
ProgramRepair



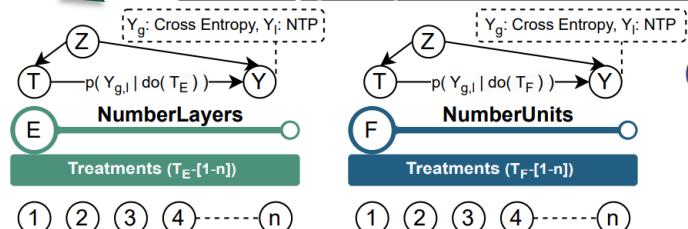
Node Type Masking



Semantic Preserving



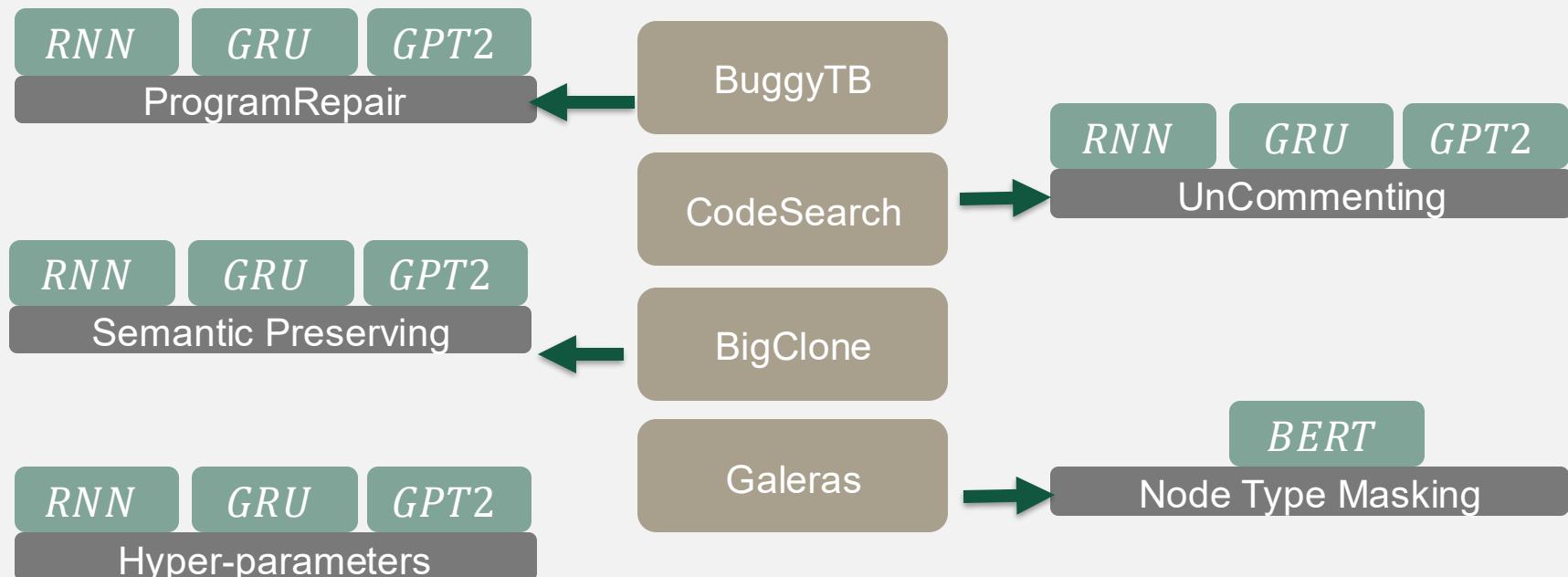
Hyper-parameters



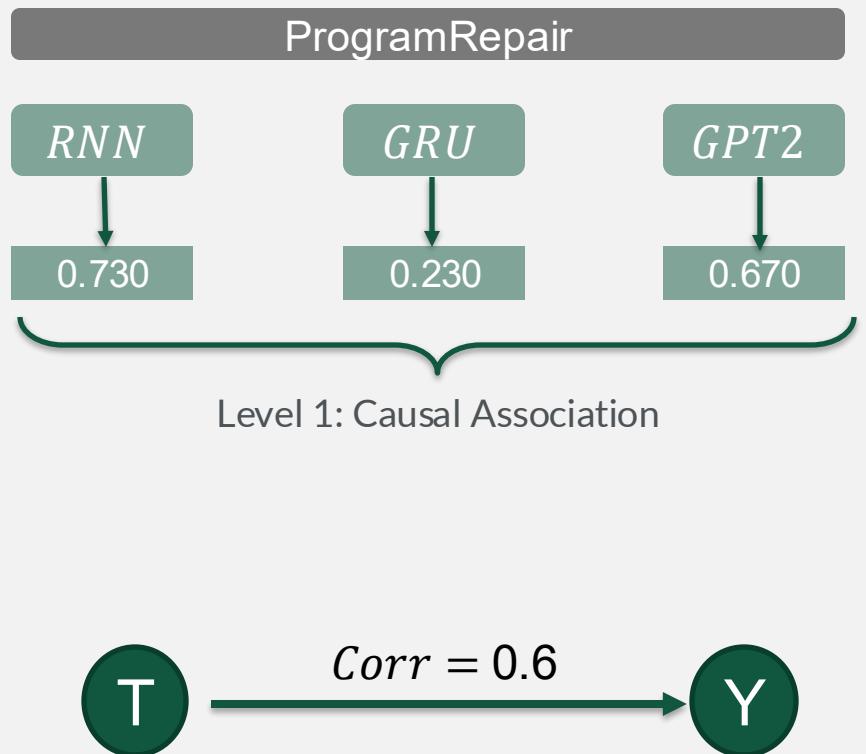
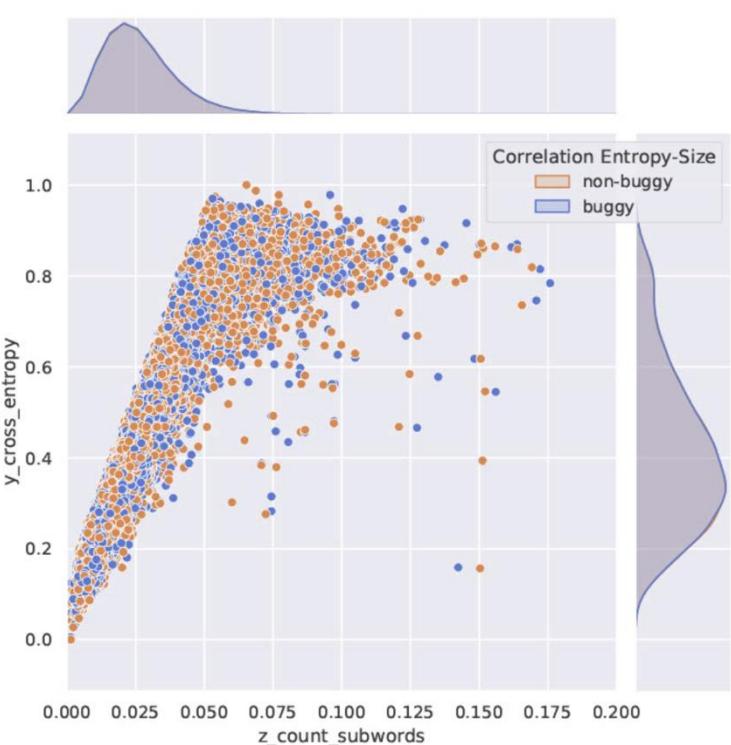
Study Context: Neural Code Models



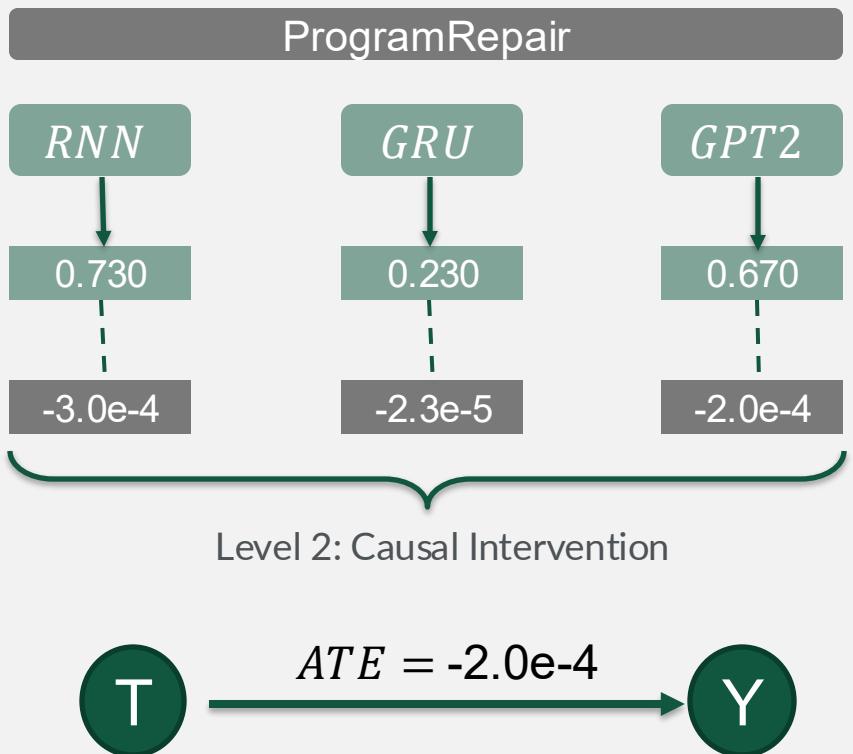
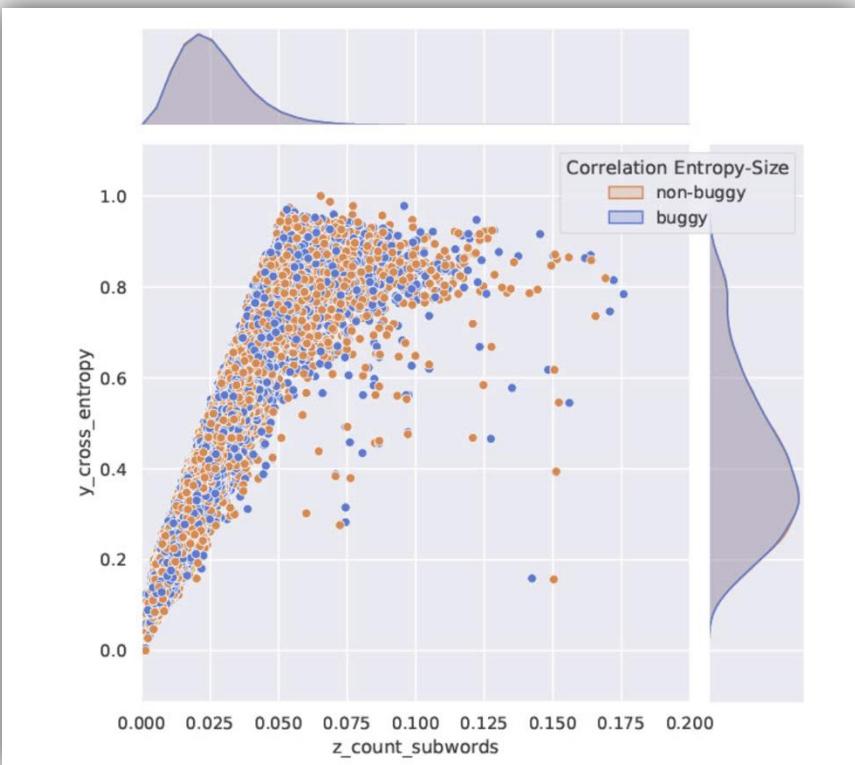
Study Context: Testbeds



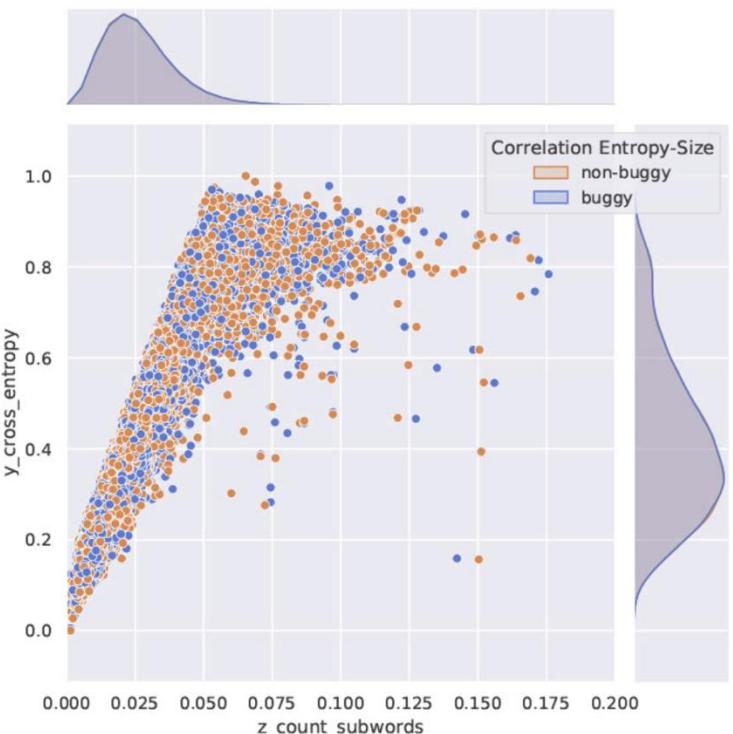
Case 1: [R₁] To what extent the prediction performance of a given NCM is affected by the presence of bugs in the prompt?



Case 1: [R₁] To what extent the prediction performance of a given NCM is affected by the presence of bugs in the prompt?



Case 1: [R₁] To what extent the prediction performance of a given NCM is affected by the presence of bugs in the prompt?

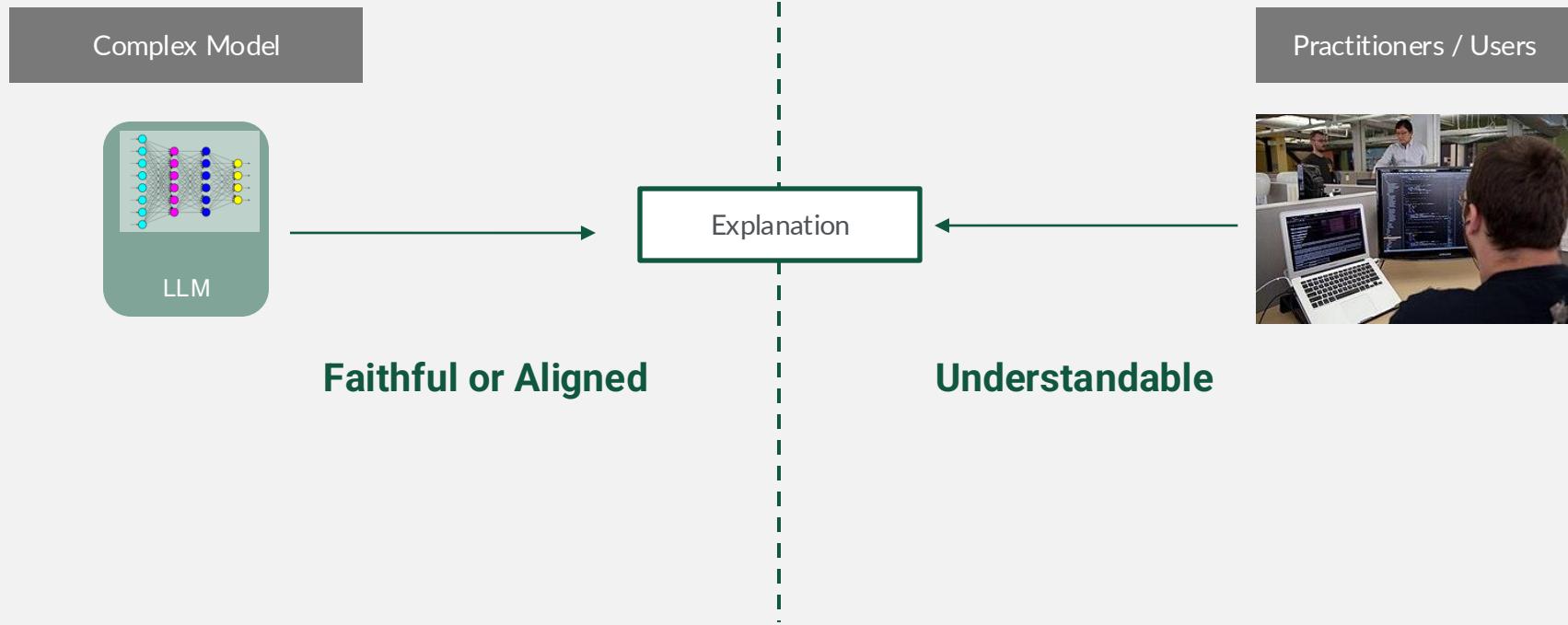


Causation is not Correlation

The presence or absence of buggy code does not appear to causally influence (or explain) the prediction performance of our NCMs even under measured high correlation

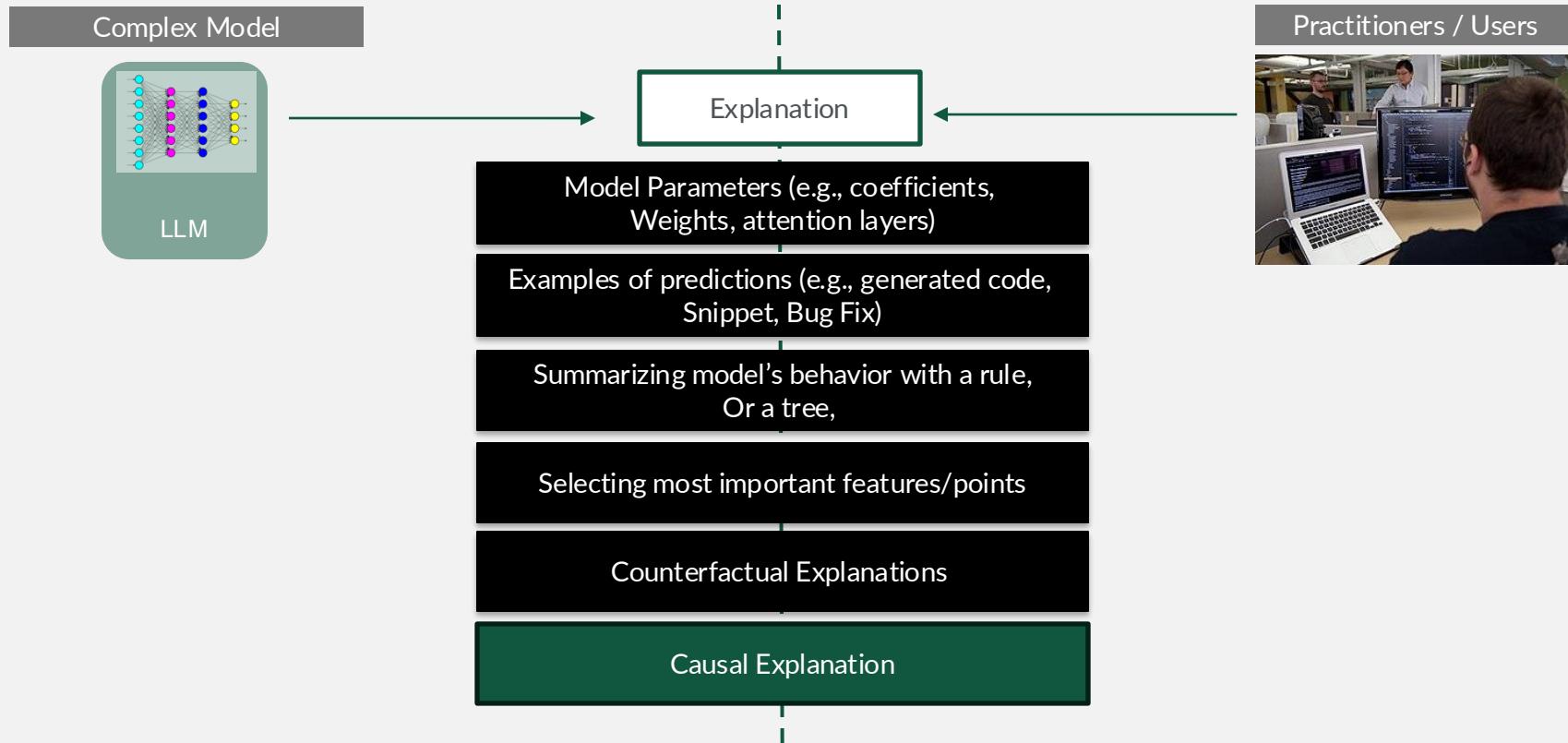
What is an Explanation?

Descriptions of the model behavior



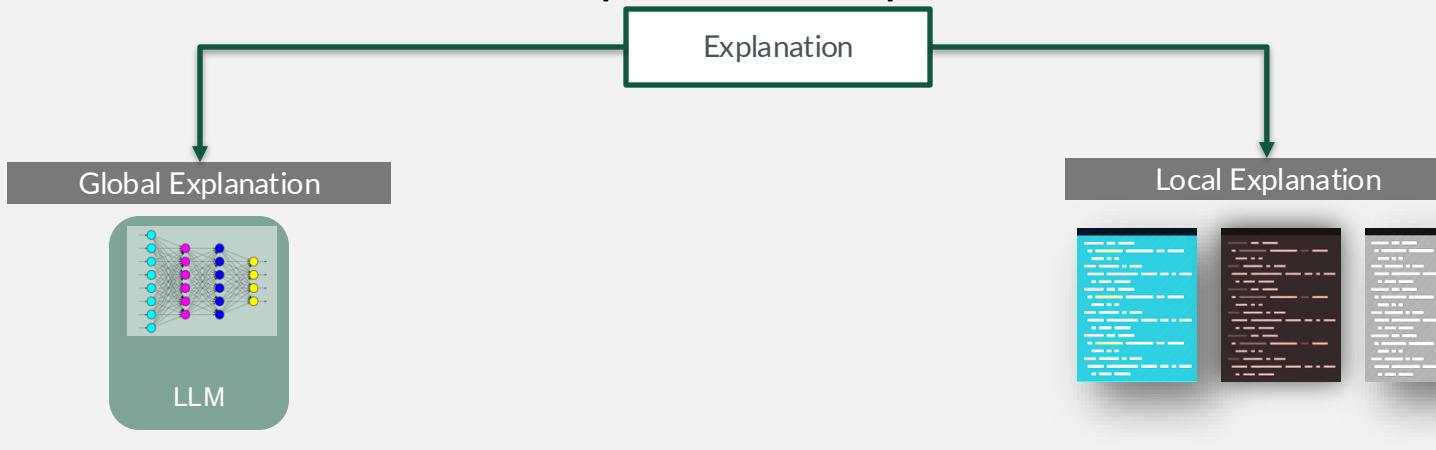
What is an Explanation?

Types of Explanations



What is an Explanation?

Scope of the Explanation



Explain overall behavior

Explain specific domain of samples

Help to detect biases at a high level

Help to detect biases in the local neighborhood

Help vet if the model is suitable for deployment

Help vet if individual samples are being generated for the right reasons

Topic 5 – Discussion and Implications

Challenges, Limitations, and Future Research

Applying do_{code} for Neural Code Models

Guidelines

1

Construct Syntax Clustering
Function

2

Defining a Structural Causal
Model

4

Estimating and Interpreting
Causal Effects

3

Connecting SCM with
Testbeds

5

Checking for Robustness
with Refutations

Challenges: *do_{code}* in practice

Toward a Theory of Causation for Interpreting Neural Code Models

David N. Palacio, *Student Member, IEEE*, Alejandro Velasco, *Student Member, IEEE*, Nathan Cooper, *Member, IEEE*, Alvaro Rodriguez, Kevin Moran, *Member, IEEE*, Denys Poshyvanyk, *Fellow, IEEE*

Abstract—Neural Language Models of Code, or Neural Code Models (NCMs), are rapidly progressing from research prototypes to commercial developer tools. As such, understanding the capabilities and limitations of such models is becoming critical. However, the abilities of these models are typically measured using automated metrics that often only reveal a portion of their real-world performance. While, in general, the performance of NCMs appears promising, currently much is unknown about how such models arrive at decisions. To this end, this paper introduces *do_{code}*, a post hoc interpretability method specific to NCMs that is capable of explaining model predictions. *do_{code}* is based upon causal inference to enable programming language-oriented explanations. While the theoretical underpinnings of *do_{code}* are extensible to exploring different model properties, we provide a concrete instantiation that aims to mitigate the impact of *spurious correlations* by grounding explanations of model behavior in properties of programming languages. To demonstrate the practical benefit of *do_{code}*, we illustrate the insights that our framework can provide by performing a case study on two popular deep learning architectures and ten NCMs. The results of this case study illustrate that our studied NCMs are sensitive to changes in code syntax. All our NCMs, except for the BERT-like model, statistically learn to predict tokens related to blocks of code (e.g., brackets, parenthesis, semicolon) with less confounding bias as compared to other programming language constructs. These insights demonstrate the potential of *do_{code}* as a useful method to detect and facilitate the elimination of confounding bias in NCMs.

Index Terms—Causality, Interpretability, Neural Code Models.

1 INTRODUCTION

THE combination of large amounts of freely available code-related data, which can be mined from open source repositories, and ever-more sophisticated deep learning architectures for code, which we refer to as Neural Code Models (NCMs), have fueled the development of Software Engineering (SE) tools with increasing effectiveness. NCMs have (seemingly) illustrated promising performance across a range of different SE tasks [1], [2], [3], [4], [5], [6], [7]. In particular, *code generation* has been an important area of SE research for decades, enabling tools for downstream tasks such as code completion [8], program repair [9], and test case generation [1]. In addition, industry interest in leveraging Large Language Models (LLMs), a scalable version of NCMs, has also grown as evidenced by tools such as Microsoft’s IntelliCode [10], Tabnine [11], OpenAI’s Codex [12], and GitHub’s Copilot [13]. Given the prior popularity of code completion engines within IDEs [14] and investment in commercial tools, NCMs for code generation will almost certainly be used to help build production software systems in the near future if they are not being used already.

Recently, there has been increased interest in evaluating NCMs for code generation. A recent study from Chen et al.

[15] illustrates that certain issues, such as alignment failures and biases, do exist for large-scale NCMs. Most of the conclusions from Chen et al.’s study were uncovered through manual analysis, e.g., through sourcing counterexamples, making it difficult to rigorously quantify or to systematically apply such an analysis to research prototypes [16]. Given the rising profile and role that NCMs for code generation play in SE and the current limitations of adopted evaluation techniques, it is clear that *new methods are needed to provide deeper insight into NCMs’ performance*.

Much of the quality assessment work on NCMs has primarily concentrated on accuracy-based metrics (e.g., Accuracy, BLEU, METEOR, ROUGE) as opposed to multi-modal evaluations (e.g., robustness, fairness, bias, efficiency). Moreover, skepticism within the NLP research community is growing regarding the efficacy of current accuracy-based metrics, as these metrics tend to overestimate model performance [17], [18], [19]. Even benchmarks that span multiple tasks and metrics have been shown to lack robustness, leading to incorrect assumptions of model comparisons [20]. Notable work has called for a more systematic approach that aims to understand a given model’s behavior according to its linguistic capabilities [17], while others have suggested the need for holistic evaluations of Language Models [21].

In addition to limitations with current methods of model quality assessment, some of the most popular NCMs have been adapted from the field of Natural Language Processing (NLP), and thus may inherit the various limitations often associated with such models — including biases, memorization, and issues with data inefficiency, to name a few [22]. However, perhaps the most problematic aspect of current neural language models is the fact that they are incapable

• David Nader Palacio, Alejandro Velasco, Nathan Cooper, and Denys Poshyvanyk are with the Department of Computer Science, William & Mary, Williamsburg, VA, 23185.
E-mail: danaderpalacio.velasco@wm.edu, ncooper, poshyvanyk@wm.edu.
• Kevin Moran is with the Department of Computer Science, George Mason University, Fairfax, VA, 22303.
E-mail: kpmoran@gmu.edu
• Alvaro Rodriguez is with Anonymous University.
E-mail: alvaro@rodriguezlab.org
Manuscript received January 16th, 2023;

Proposing a new syntax clustering criterion

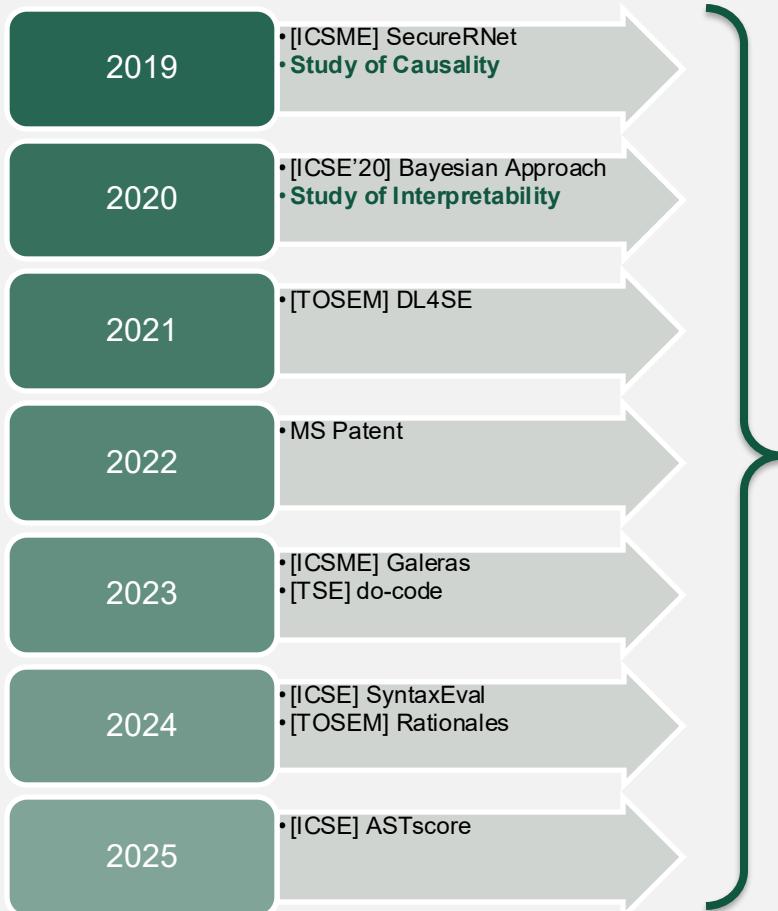
Causal Discovery and Validity

Integrating *do_{code}* in DL4SE Life-Cycle

Collecting data for formulating SE-based interventions

Causal Inference Impact in Software Research

Causality as building blocks for...



Project A: Trust and Trustworthiness

Project B: Interpretable DL Models

- (Interpretable) Program Repair with Transformers
- (Interpretable) Code Generation
- (Interpretable) Machine Translation
- (Interpretable) AI-Based Assisted Tools for Dev.

Project C: Advanced Code Data Mining

- Manifold Analysis for Data Assessment and Exploration
- Empirical Studies in Software Engineering Adopting CI

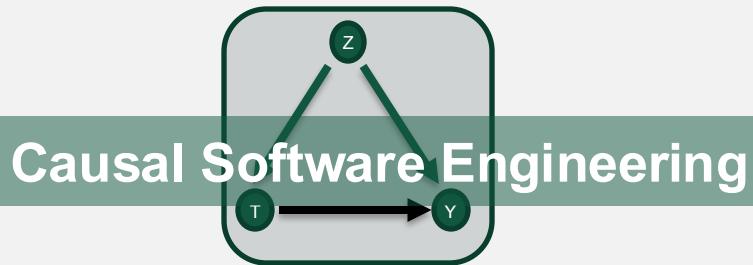
Project D: Software Data Representation

- On the Causal Nature of Software Information

Causal SE Information

Conjecture: Software Information is not only “natural” on a sequence but also exhibits causal properties.

The intersection between **Causal Inference** and Deep Learning for **SE** is beyond *interpretability*. It is a whole new science that must be employed to enhance software data analysis (to reduce confounding bias) and causal discovery (to elaborate explanations).



Thank you ☺

This presentation was designed based on David Nader Palacio's dissertation