Machine Learning Security

Kexin Pei
Department of Computer Science
The University of Chicago

“Software is Eating the World”

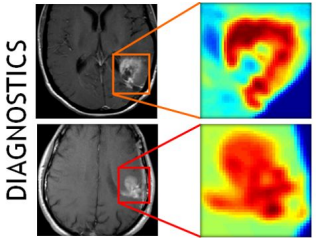
- Marc Andreessen

“ML is Software 2.0”

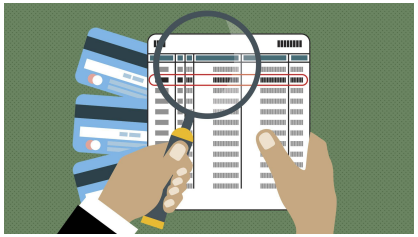
- Andrej Karpathy

ML is Increasingly Used in Security and Safety-Critical Systems

Self-Driving Cars

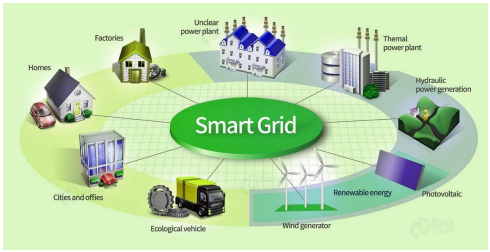


Medical Diagnosis

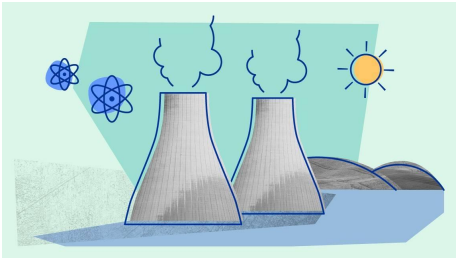


Fraud/Intrusion/Malware Detection

Flight Control Systems



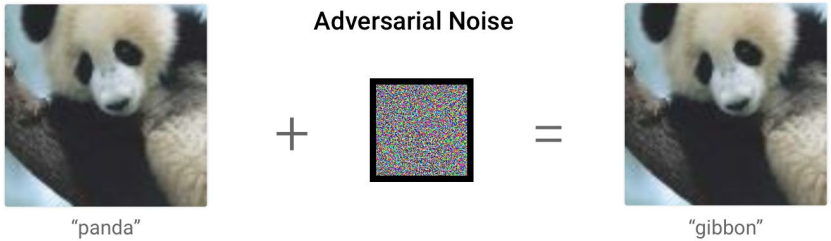
Smart Grids Energy Distribution



Nuclear Energy Monitor System

Forecasting Natural Disasters

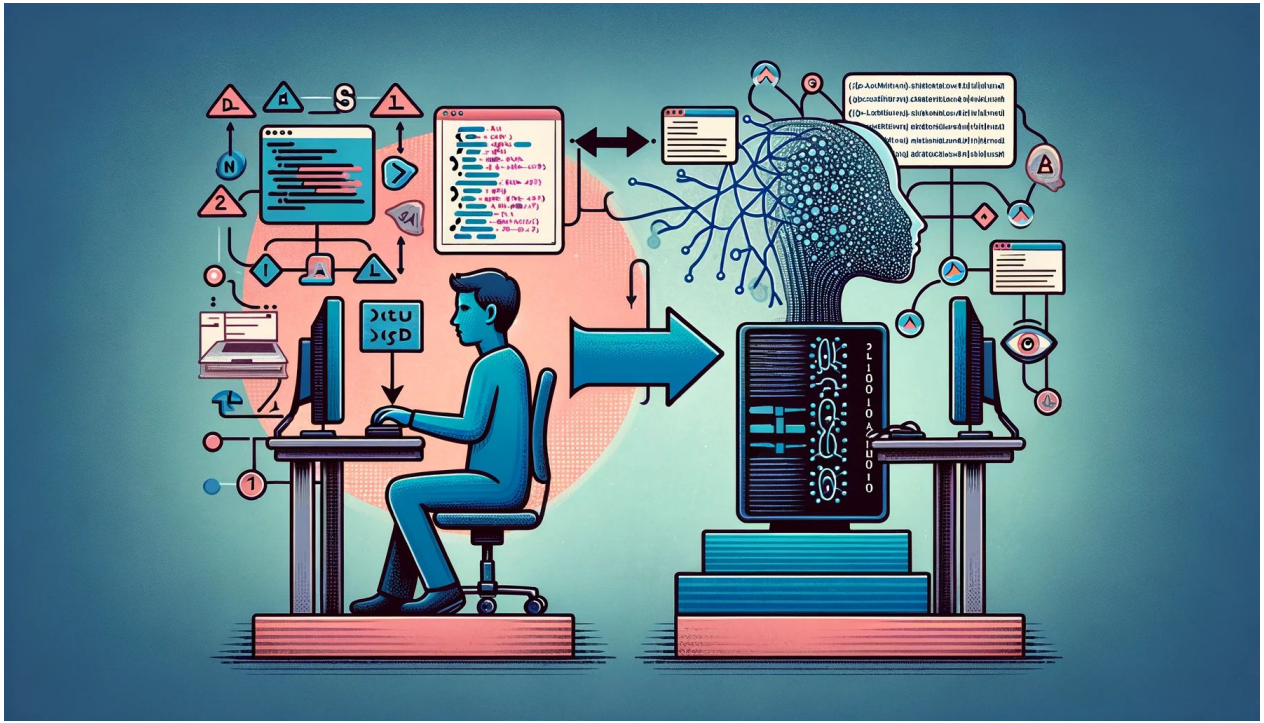
Machine Learning is Vulnerable



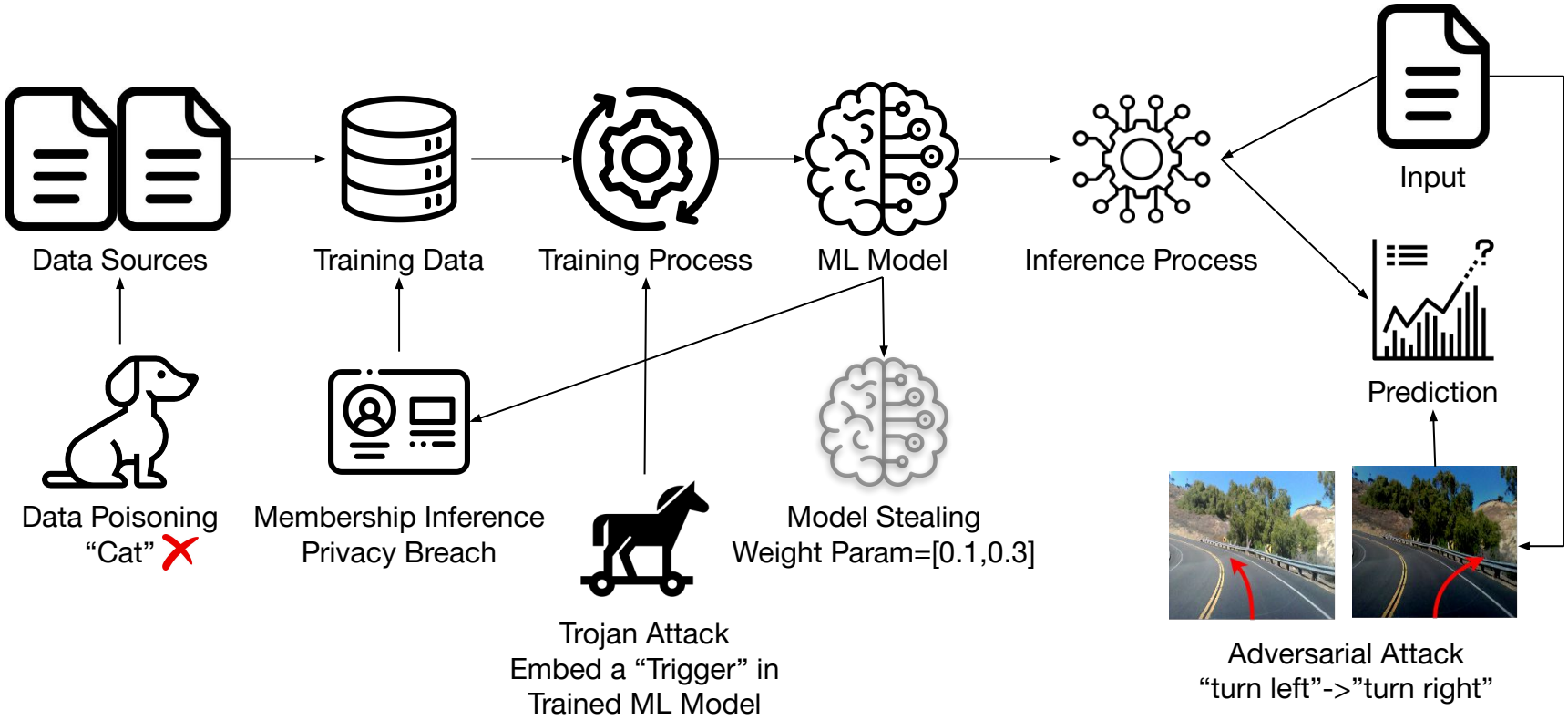
Adversarial Examples

Tesla Autopilot Failed to Recognize a White Truck against Bright Sky Leading to Fatal Crash

Paradigm Shift of Software Development



Threat Landscape Exposed by ML



"OWASP Machine Learning Security Top Ten": https://owasp.org/www-project-machine-learning-security-top-10/

ML Security in a Nutshell

Security of Machine Learning

Machine Learning for Security?

Machine Learning has Automated Many Security Applications

Research Threat intelligence Vulnerabilities and exploits · 4 min read

Microsoft researchers work with Intel Labs to explore new deep learning approaches for malware classification

By Microsoft Threat Intelligence

A Smarter Way to Detect Suspicious Cloud Logins

10/25/2021 | ActZero | 4 minutes

How Proofpoint Aegis Uses Machine Learning

SHARE WITH YOUR NETWORK!

FEBRUARY 21, 2023 | ADAM STARR AND LUIS BLANDO

The advent of transformer-based natural language processing (NLP) models, large language models (LLMs) and generative models has made advanced machine learning (ML) systems accessible to everyone with internet access through tools like ChatGPT. Unfortunately, malicious actors can also use these models to create phishing emails.

In this post, we'll explain how Proofpoint defends against artificial intelligence (AI)-generated phishing threats and deploys advanced ML models throughout our detection systems.

Defending against AI-generated phishing threats

Learn / Microsoft Defender for Cloud Apps /

Anomaly detection policies in Defender for Cloud Apps

environment and triggers alerts with respect to a baseline that was learned on your organization's activity. These detections also use machine-learning algorithms designed to profile the users and sign in pattern to reduce false positives.

09 MARCH 2021 TECH TOPICS

Detecting threats in AWS Cloudtrail logs using machine learning

By Craig Chamberlain

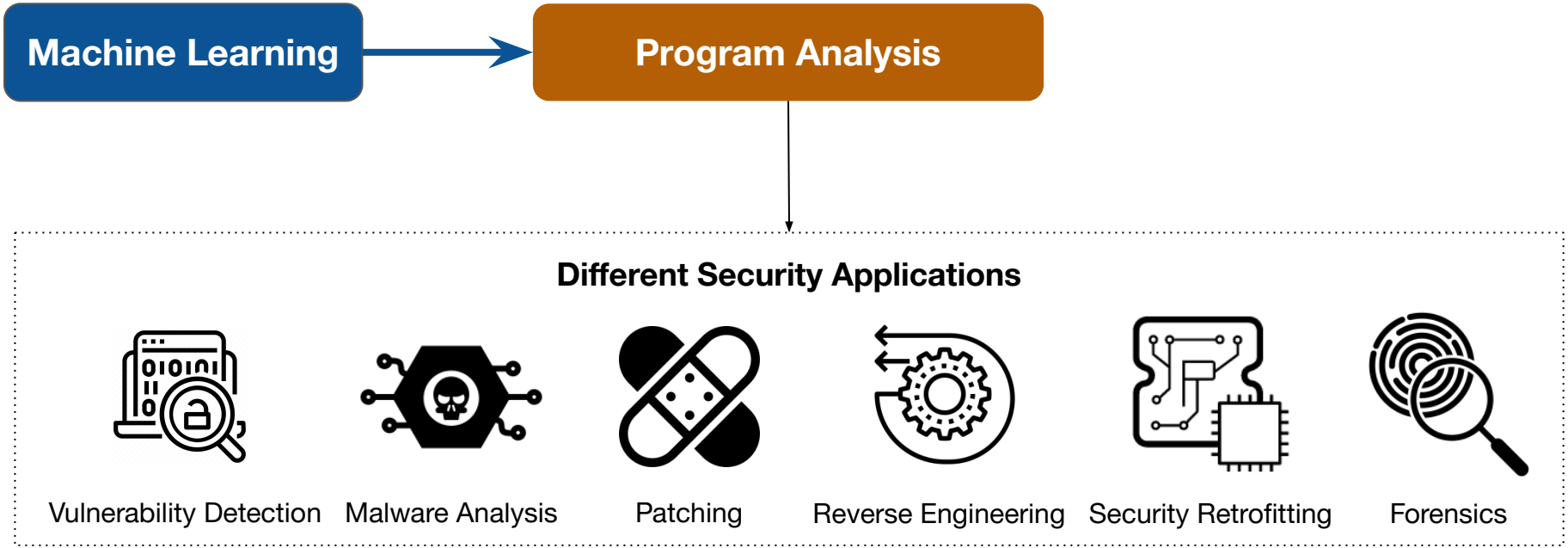
Share

Engineering Security

Leveraging machine learning to find security vulnerabilities

A behind-the-scenes peek into the machine learning framework powering new code scanning security alerts.

My Research: Machine Learning for Software Security



Program Analysis is Crucial for Building Trustworthy Software

Software Programs

Questions

Can pointer p be NULL at line L?

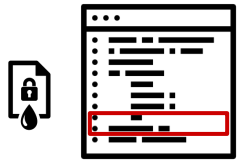
Program Analysis

Answers

Help to Build

Vulnerability

Privacy Leak



Trustworthy Software

- **Security**
- **Reliability**
- **Safety**
- **Privacy**
- **Performance**

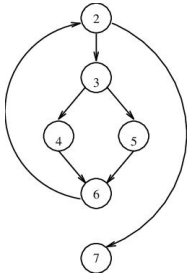
Challenges of Traditional Program Analysis

Human Expert

Hand-Curate

Rules and Heuristics

How to



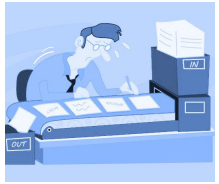
Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

Output: $in(B)$ and $out(B)$ for every basic bloc B.

for each B **repeat**
  $out(B) := gen(B)$
while changes to any $out(B)$ occur **repeat**
  $in(B) := \bigcup_{B' \in pred(B)} out(B')$
  $out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

Represent a program?

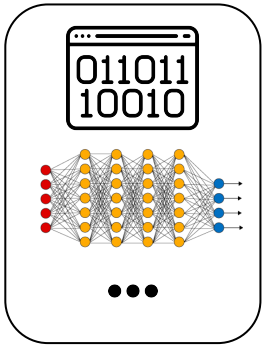
Design analysis rules?



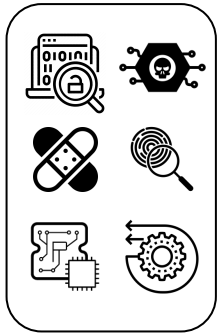
Significant Manual Effort

Tune

Heterogeneous Software



Various Security Applications



Machine Learning Shows Promise for Analyzing Programs

Human Expert

Hand-Curate

Rules and Heuristics

How to

Represent a program?

Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

Output: $in(B)$ and $out(B)$ for every basic bloc B.

for each B **repeat**
  $out(B) := gen(B)$
  while changes to any $out(B)$ occur **repeat**
    $in(B) := \bigcup_{B' \in \text{pred}(B)} out(B')$
    $out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

Design analysis rules?

Abundant Public Code

Abundant Compute

Learn

Program Representations

Analysis Rules

Machine Learning Shows Promise for Analyzing Programs

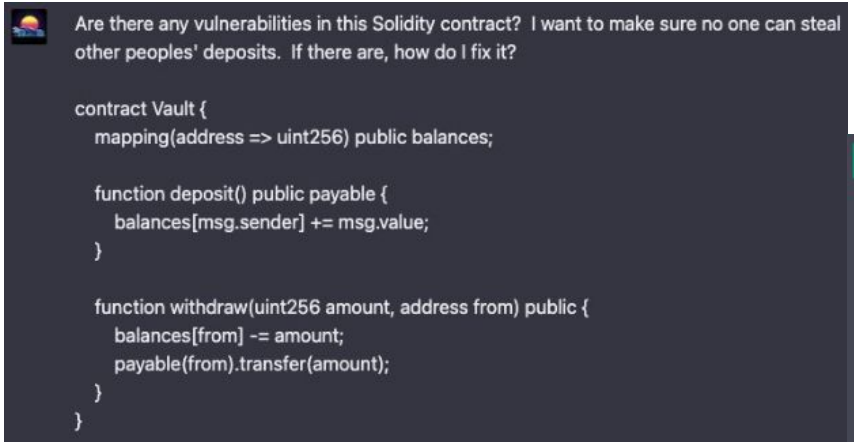
Detecting and Exploiting Vulnerabilities

devtooligan 
@devtooligan

omg. seriously mind-blown 🤯

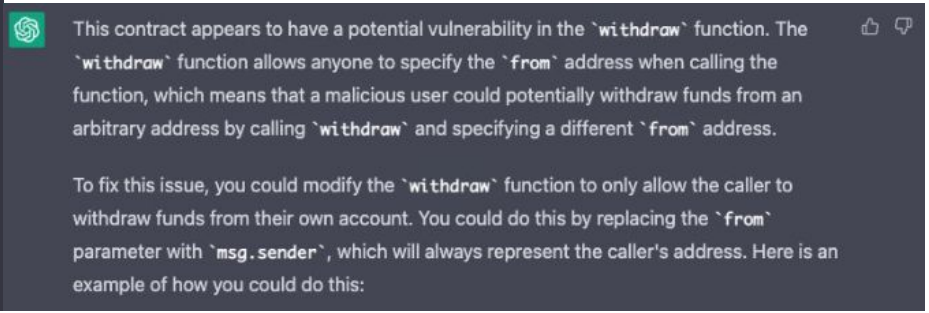
Brandon Rossi
@0xConda

I told ChatGPT to pentest a scope of IPs and it found me $69,420 worth of bugs.



Are there any vulnerabilities in this Solidity contract? I want to make sure no one can steal other peoples' deposits. If there are, how do I fix it?

```
contract Vault {
    mapping(address => uint256) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount, address from) public {
        balances[from] -= amount;
        payable(from).transfer(amount);
    }
}
```

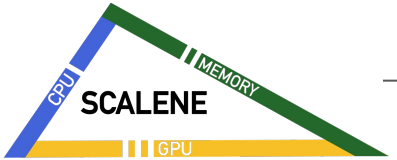


This contract appears to have a potential vulnerability in the `withdraw` function. The `withdraw` function allows anyone to specify the `from` address when calling the function, which means that a malicious user could potentially withdraw funds from an arbitrary address by calling `withdraw` and specifying a different `from` address.

To fix this issue, you could modify the `withdraw` function to only allow the caller to withdraw funds from their own account. You could do this by replacing the `from` parameter with `msg.sender`, which will always represent the caller's address. Here is an example of how you could do this:

Machine Learning Shows Promise for Analyzing Programs

Python Profiler



GPT-4

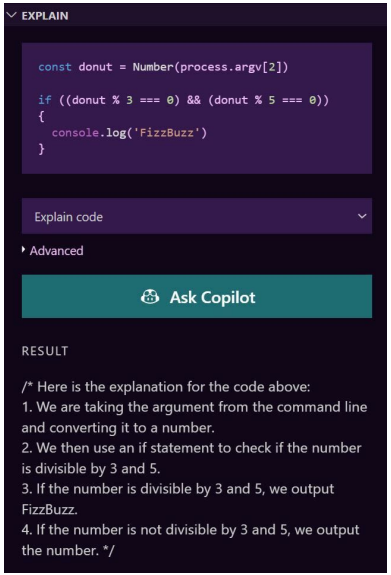
Program Optimization

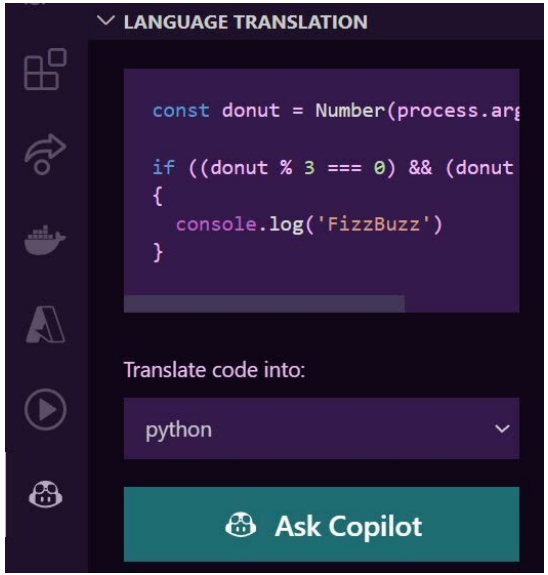
```
    z1 = [i for i in range(0,300000)][299999]

15  # Proposed optimization:
    # This code can be optimized by using the built-in function max()
    z1 = max(range(0, 300000)) # ~10x faster
```

GitHub Copilot

Explain Code

Translate Code





Google

3% Code Written by ML

ML-Powered Program Fixing, Repair, Refactoring, etc.

Huge Academic Contributions: 1,000+ Papers
https://ml4code.github.io/

Limitations: Lack Understanding of Program Semantics

A code summarization example (Alon et al., 2019, Yefet et al., 2020, Henkel et al. 2022)
code2vec.org / code2seq.org

```
void f1(int[] array){
  boolean swapped = true;
  for (int i = 0;
    i < array.length && swapped; i++){
    swapped = false;
    for (int j = 0;
    j < array.length-1-i; j++) {
      if (array[j] > array[j+1]) {
        int temp = array[j];
        array[j] = array[j+1];
        array[j+1]= temp;
        swapped = true;
      }
    }
  }
}
```

Prediction: **sort** (98.54%)

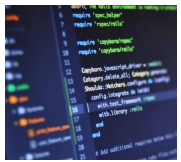
```
void f2(int[] ttypes){
  boolean swapped = true;
  for (int i = 0;
    i < ttypes.length && swapped; i++){
    swapped = false;
    for (int j = 0;
    j < ttypes.length-1-i; j++) {
      if (ttypes[j] > ttypes[j+1]) {
        int temp = ttypes[j];
        ttypes[j] = ttypes[j+1];
        ttypes[j+1]= temp;
        swapped = true;
      }
    }
  }
}
```

Prediction: **contains** (99.97%)

Yefet, Noam, Uri Alon, and Eran Yahav. "Adversarial examples for models of code." OOPSLA 2020.

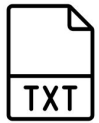
Common Practice of ML on Code

Common Practice:

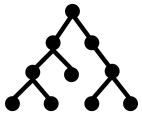


Program

Static Text

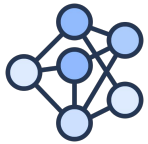


Text Sequence



Trees (AST)

Graphs (CFG/DFG)



ML Model

Specific Task

Program semantics does not just manifest in **static text**

Consequences: Lacking Robustness and Generalization

I. **Overfit** to **spurious textual** and **task-specific** patterns

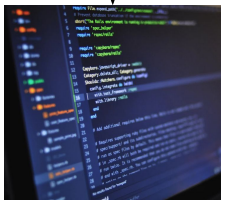
II.**Distribution shift**: **program syntax** and **task requirement** changes

Security Applications Require More Rigorous Understanding of Program Semantics

A Popular Type of Program Analysis in Security: Binary Analysis

Program Analysis

Binary Analysis



Stripped Binaries

Proprietary software,
Third-party component
Malware

Additional Complications:

```
int variable_init()
{
    int var = 22;
    return var;
}
```

```
mov eax,0x16
```

```
xor eax,eax
sub eax,-0x16
```

Variables
Arguments
Types
Data Structures
Names
...

Semantic Hints are Absent

Registers
Memory Accesses

Various Code Transformations

GCC Visual C++ — Compilers

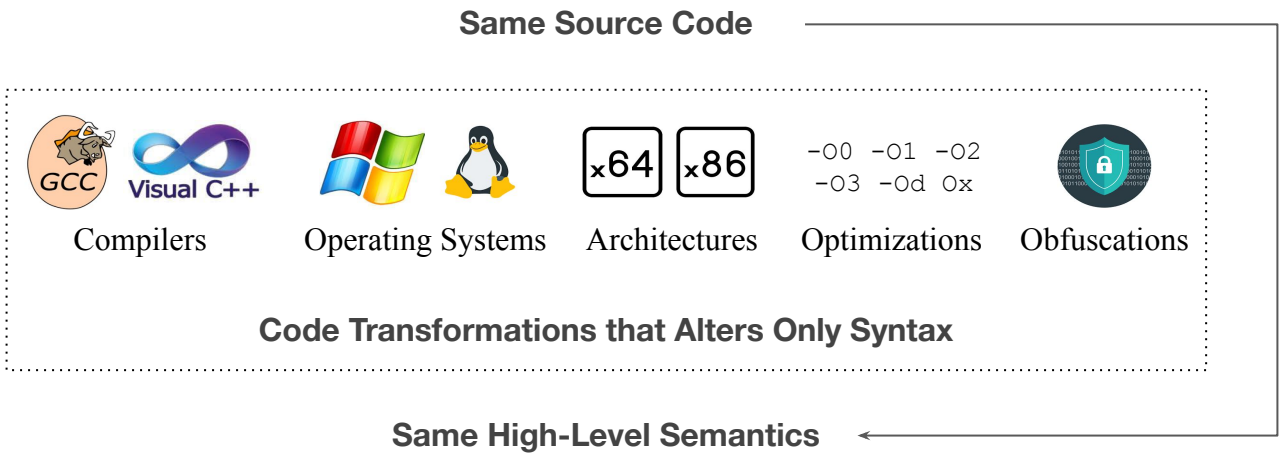
Operating Systems

x64 x86 — Architectures

-O0 -O1 -O2 -O3 -Od Ox — Optimizations

Obfuscations

What do Robustness and Generalization Imply in Binary Program Analysis?

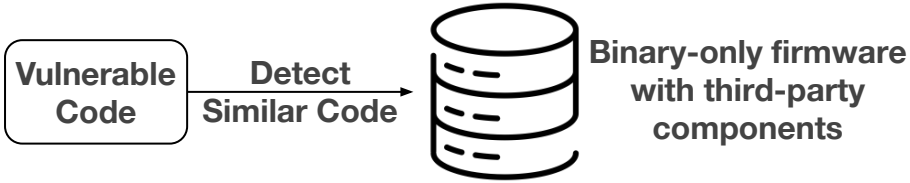


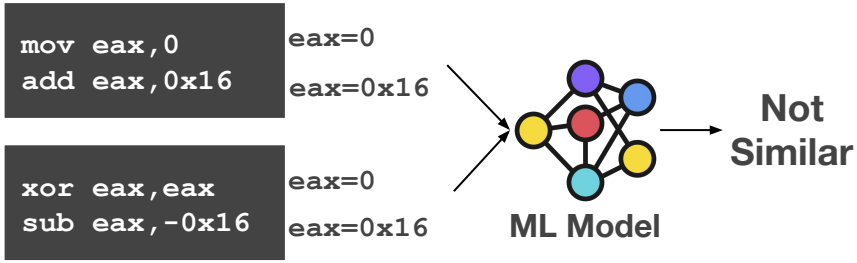
Robustness: Stay Invariant to **Syntactic** Changes

Generalization: Generalize to new **Syntactic** Changes

Security Applications Require Rigorous Understanding of Program Semantics

Detecting Binary Code Reuse Vulnerability





Without understanding **mov**, **xor**, **add**, **sub**, etc.

ML model cannot reason about program behavior to predict similarity

KP

```
mov eax,0
add eax,0x16
```

This is assembly code…these instructions **initialize the EAX register to 22**.

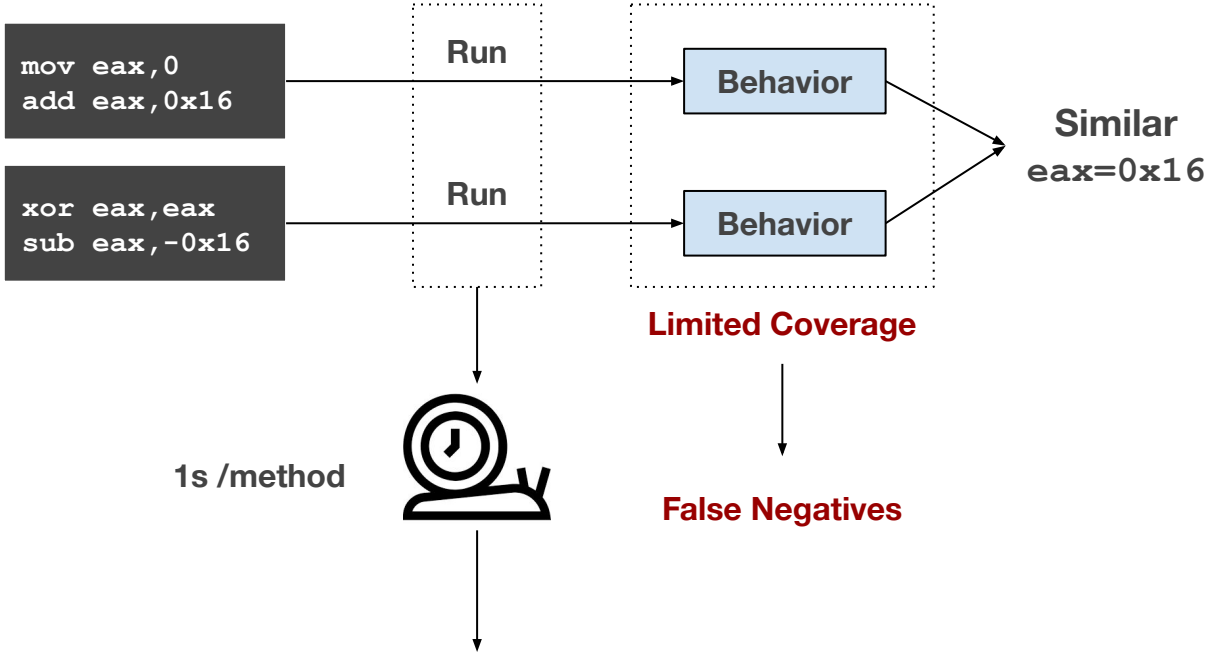
KP

is it similar to

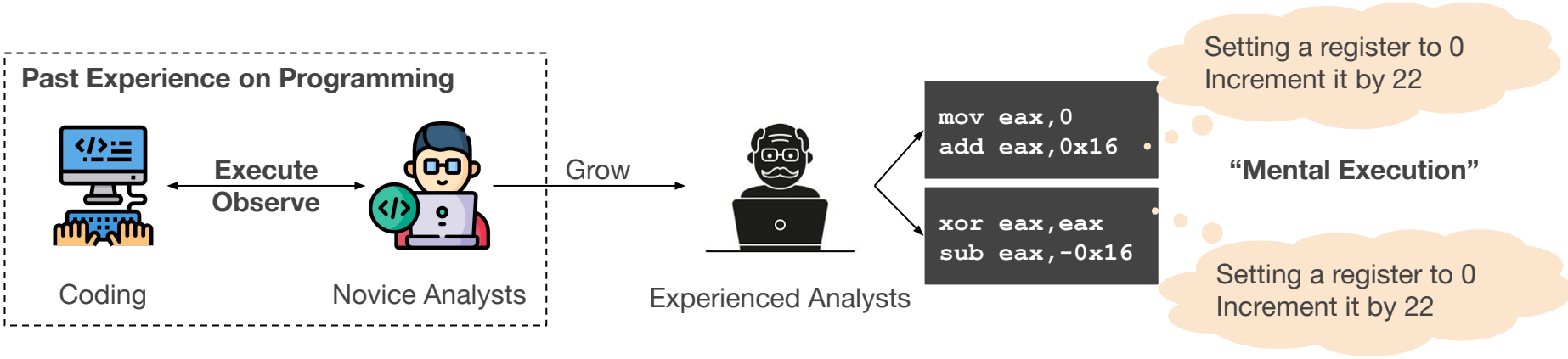
```
xor eax,eax
sub eax,-0x16?
```

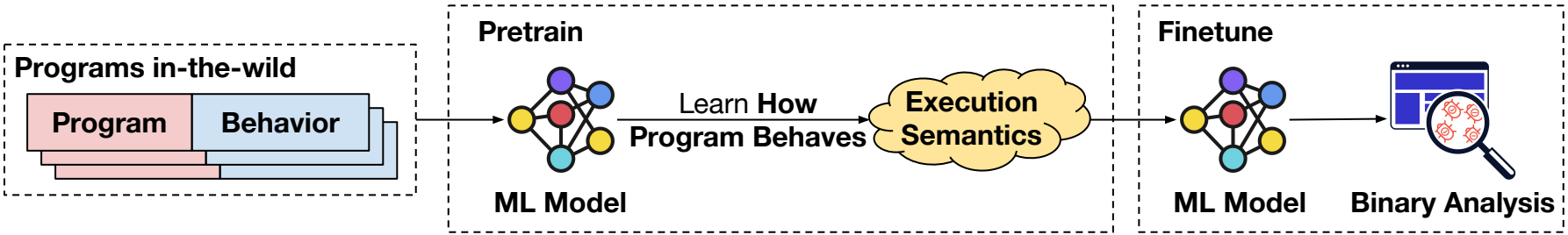
No...the second set uses the "`xor`" and "`sub`" instructions to **set the value of EAX register to -22**...

Why not dynamic analysis?

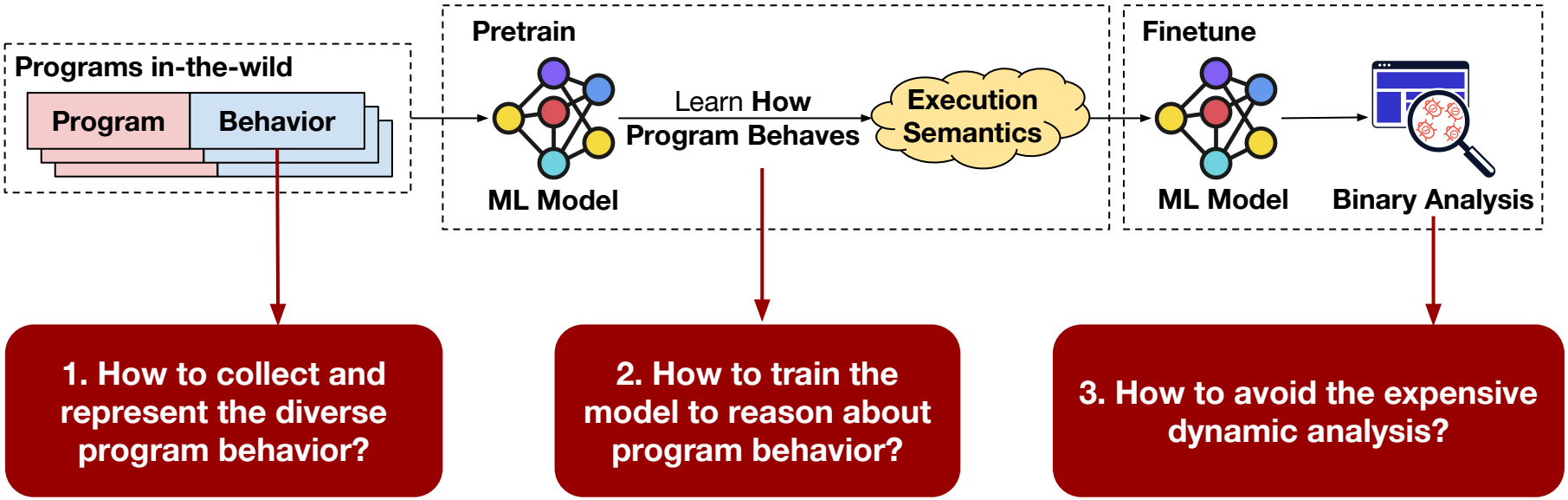


Learning Execution Semantics and Transferring it without Dynamic Analysis

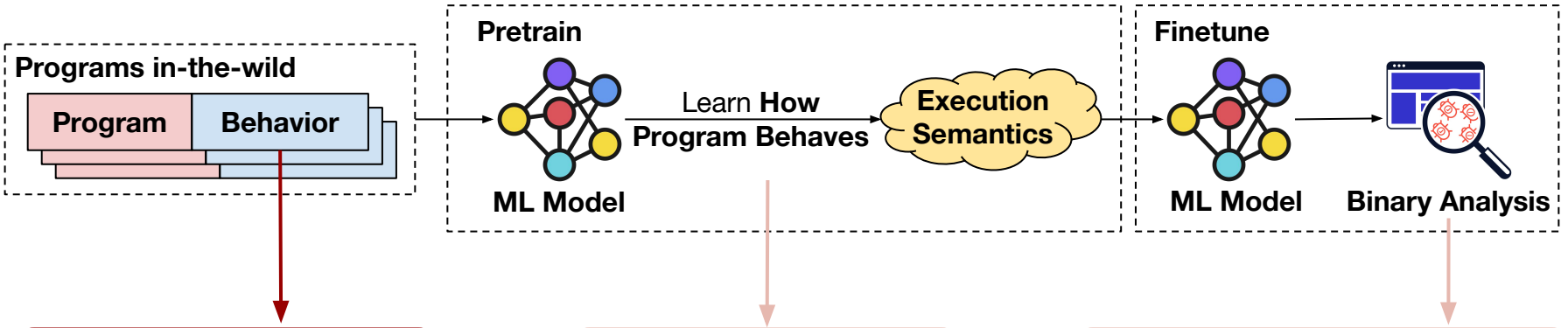




Challenges of Learning Execution Semantics



Challenges of Learning Execution Semantics

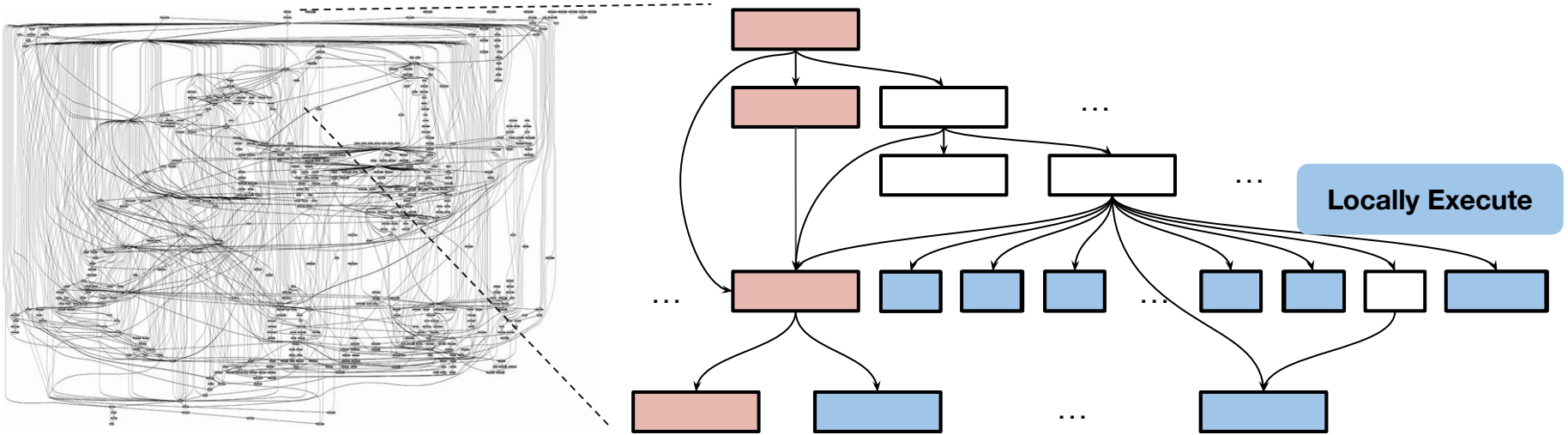


1. How to collect and represent the diverse program behavior?

2. How to train the model to reason about program behavior?

3. How to avoid the expensive dynamic analysis?

How to Collect Diverse Program Behaviors?

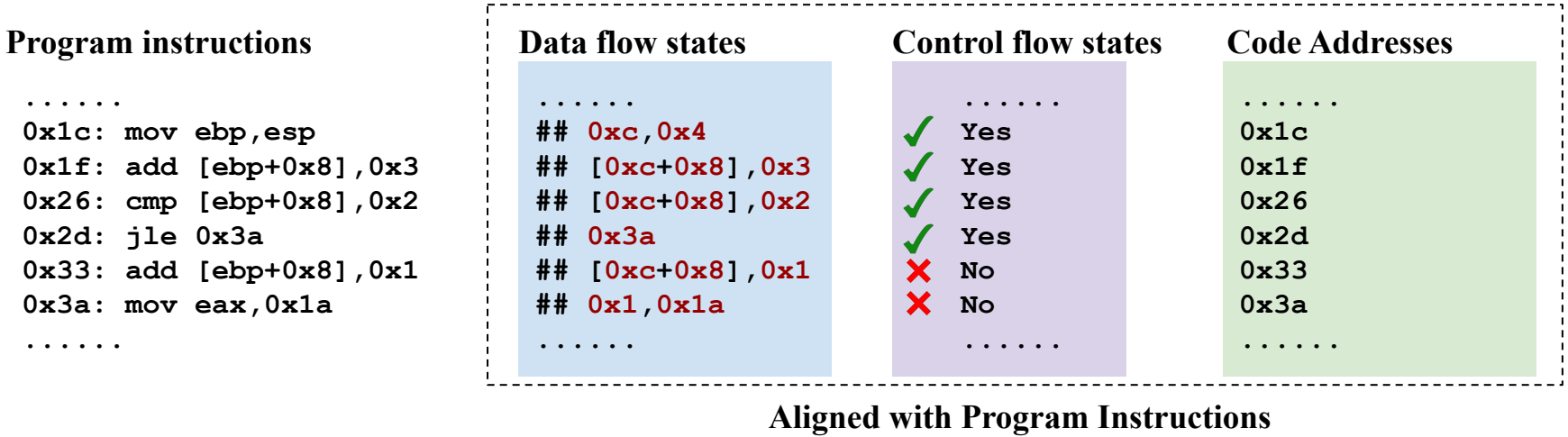


Microsoft IIS Call Graph

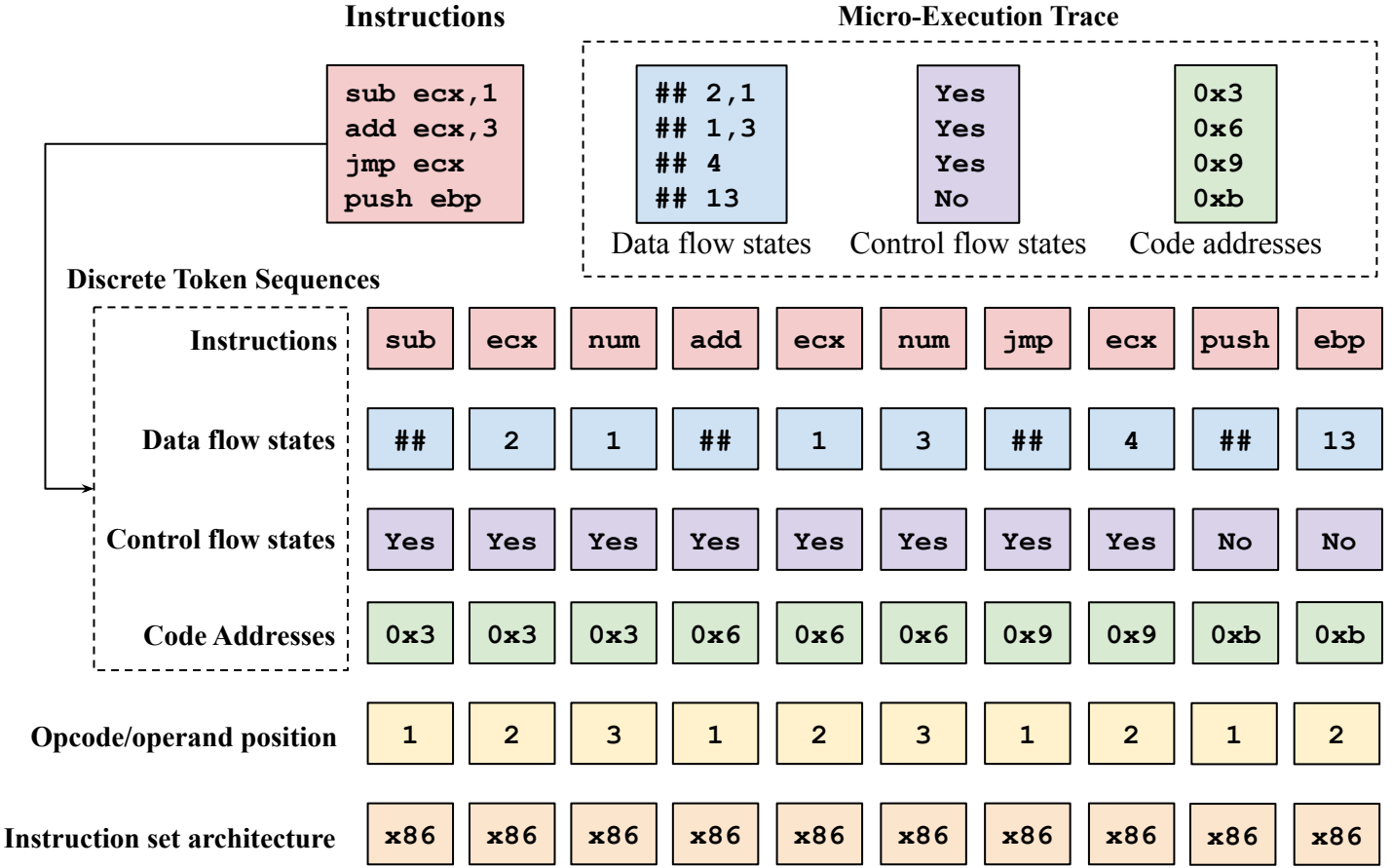
Under-Constrained Micro-Execution:
Specify arbitrary code piece to execute

- **Expose diverse code behaviors**
- **Benefit large-scale pretraining on diverse execution behavior**

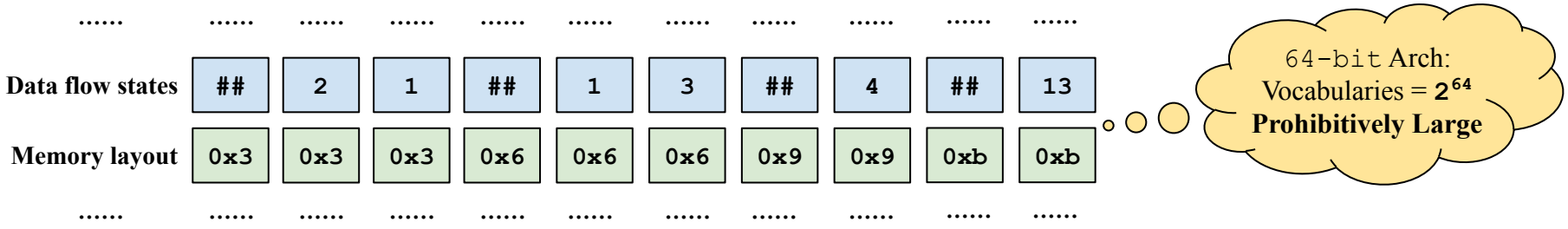
How to Collect Diverse Program Behaviors?



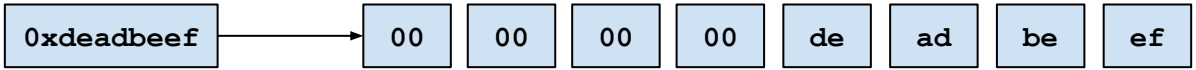
Input Representation



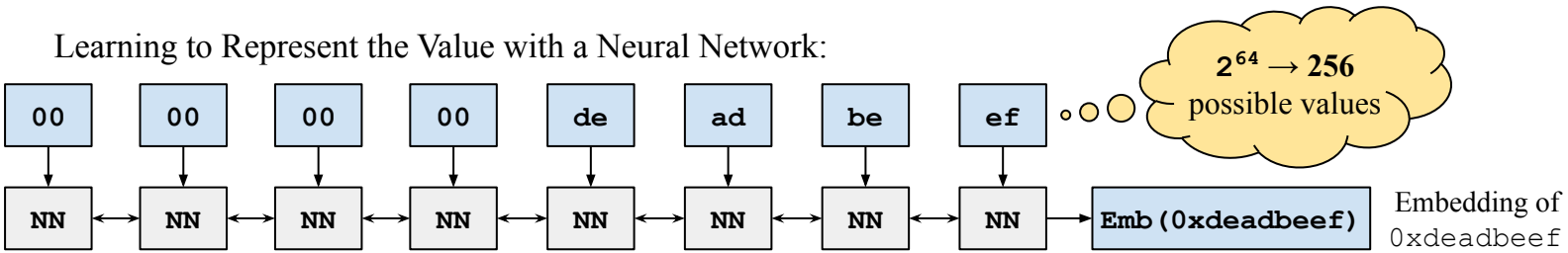
Numerical Representation



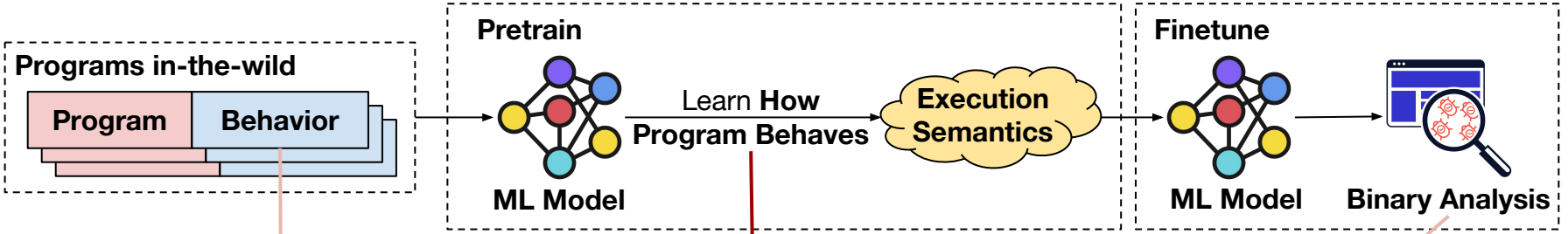
Pad Each Numeric Token as a Fixed-Length 8-Byte Sequence:



Learning to Represent the Value with a Neural Network:



Challenges of Learning Execution Semantics

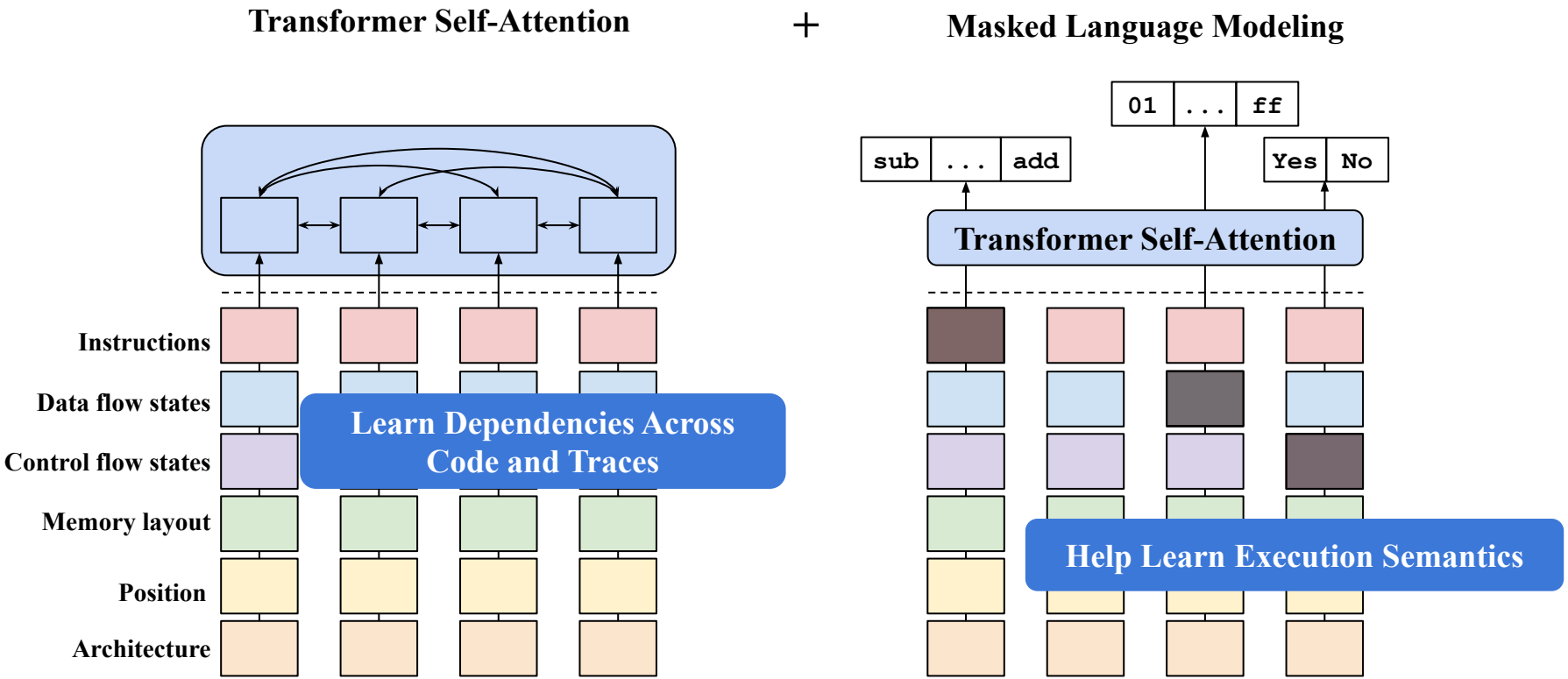


1. How to collect and represent the diverse program behavior?

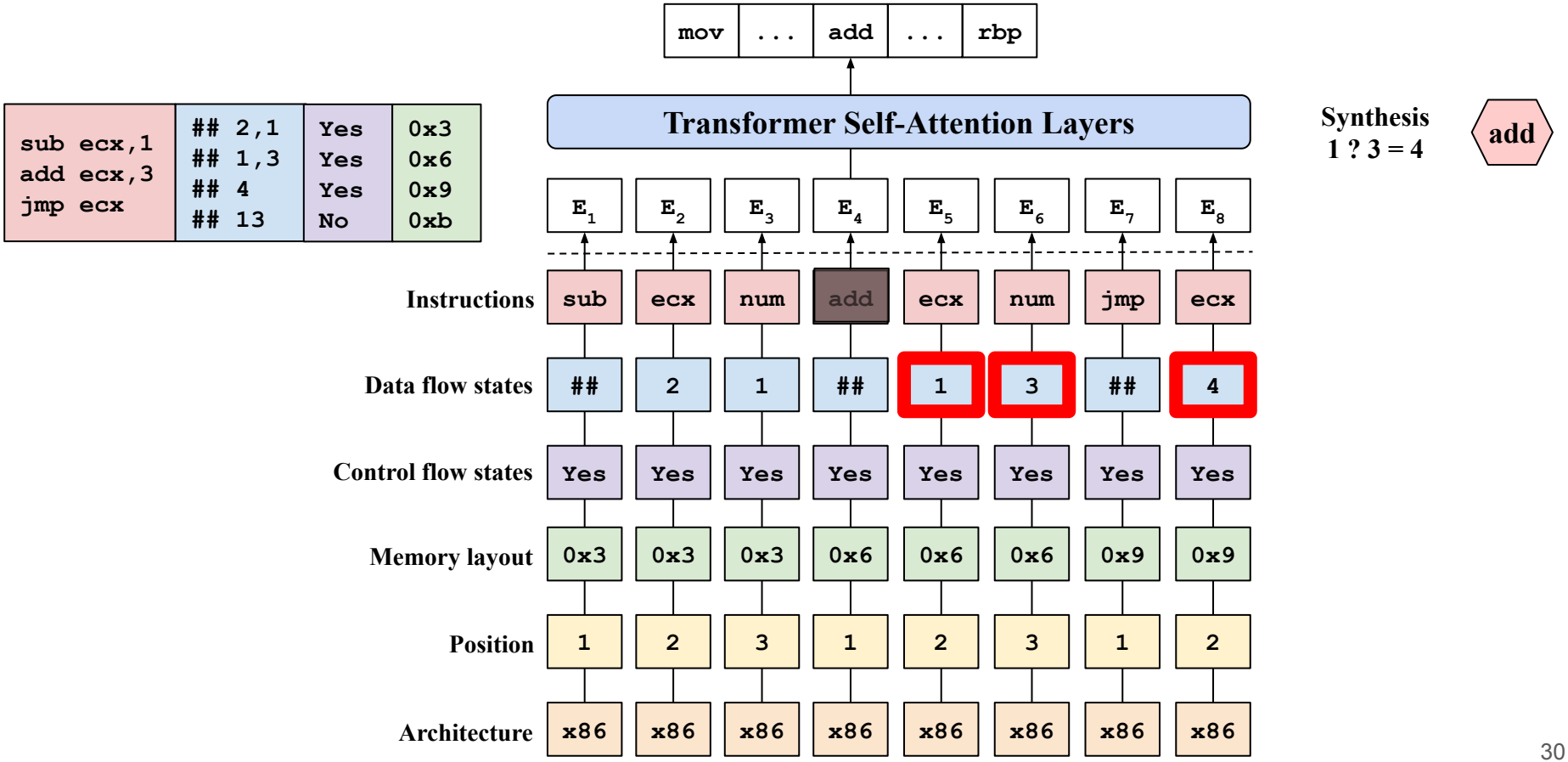
2. How to train the model to reason about program behavior?

3. How to avoid the expensive dynamic analysis?

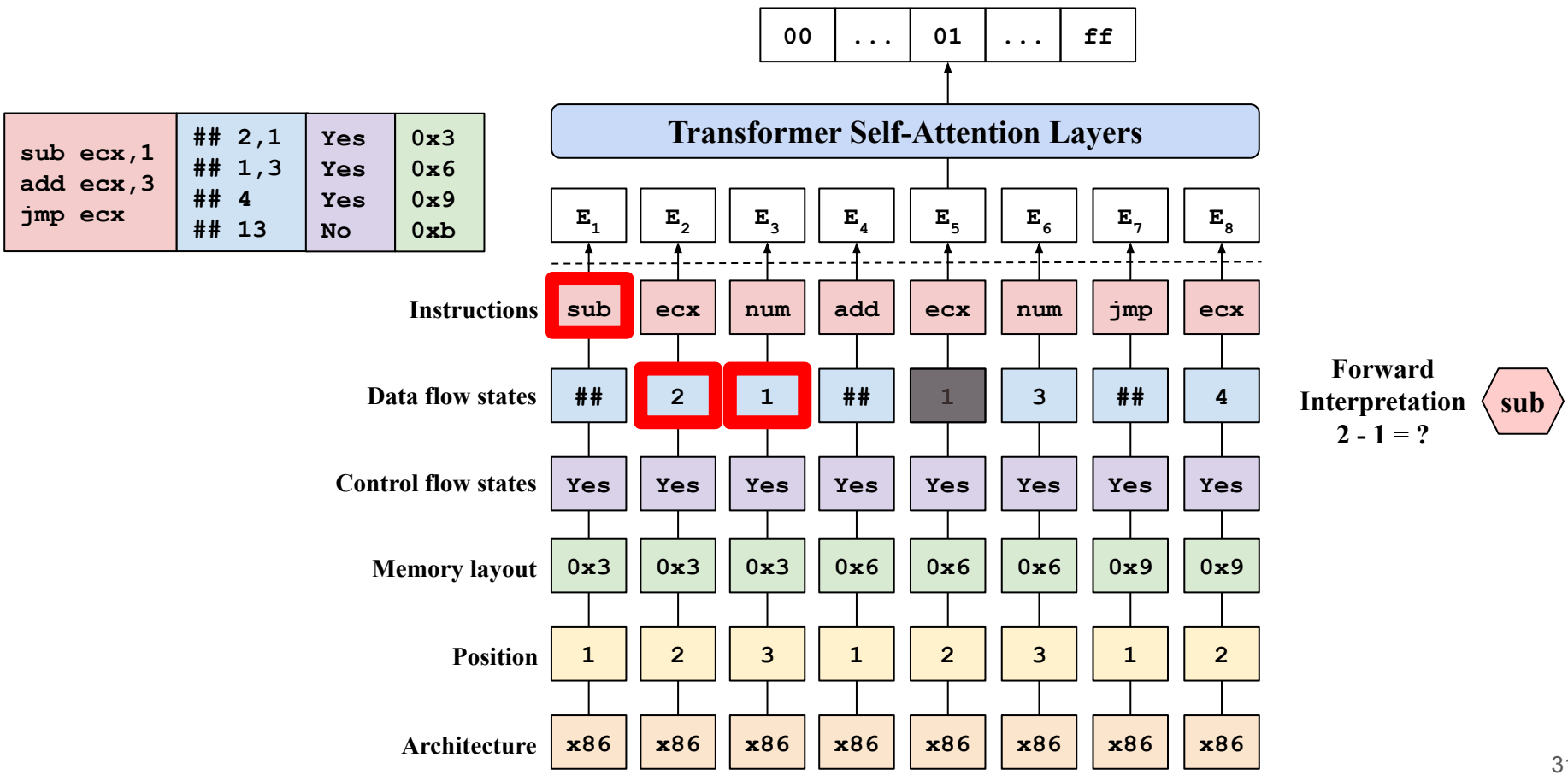
How to train the model to reason about program behavior?



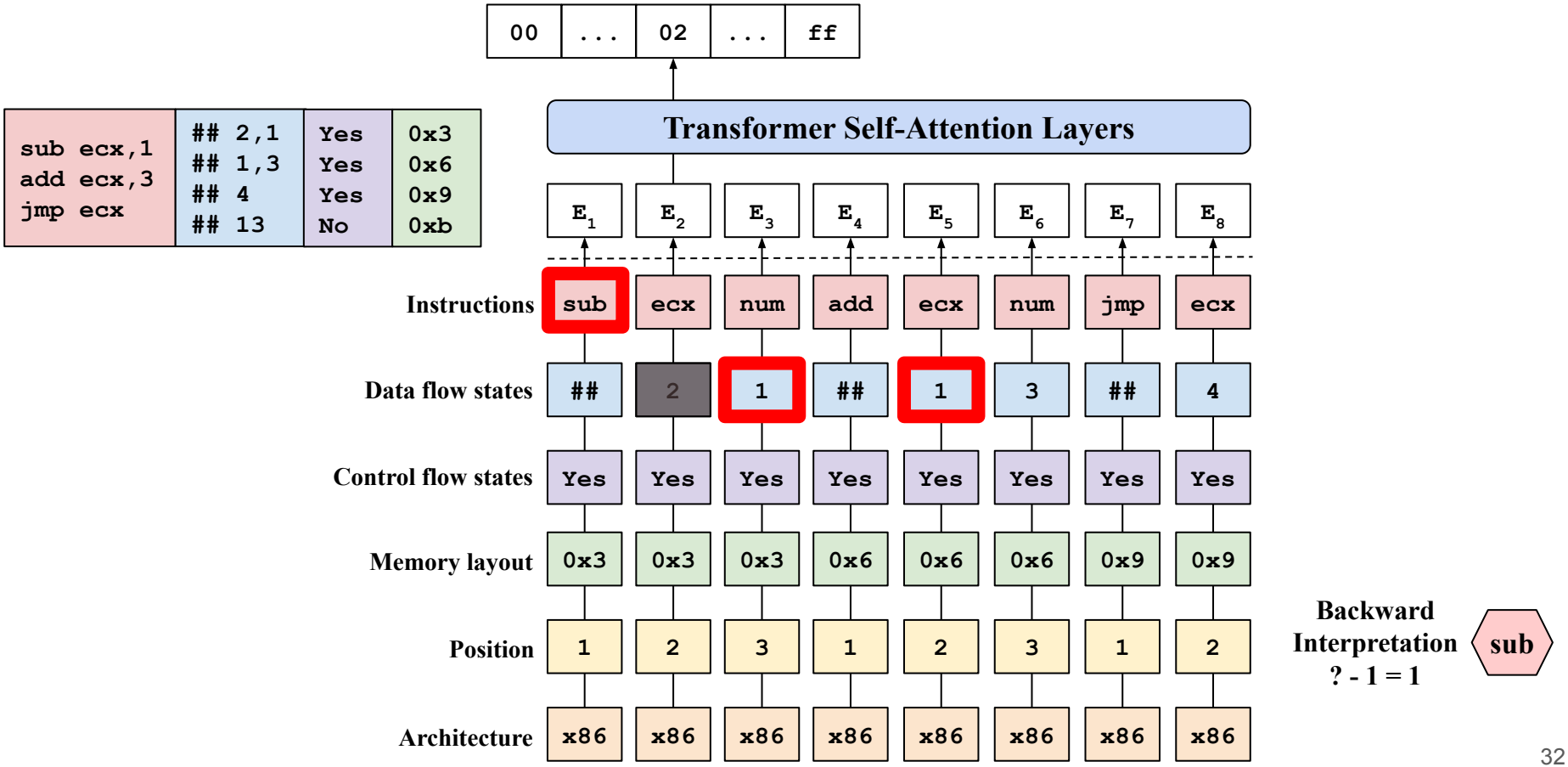
Motivating Example



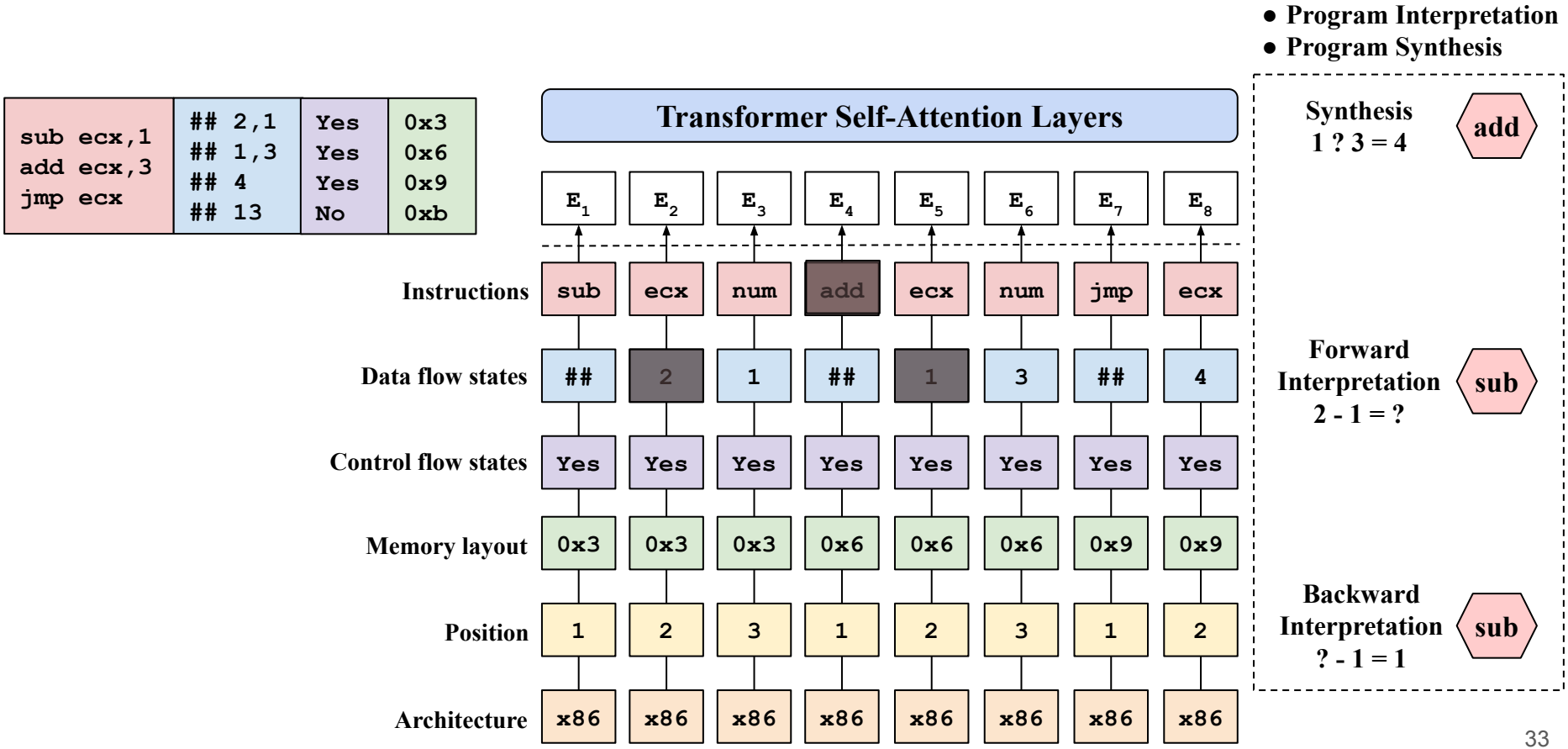
Motivating Example



Motivating Example



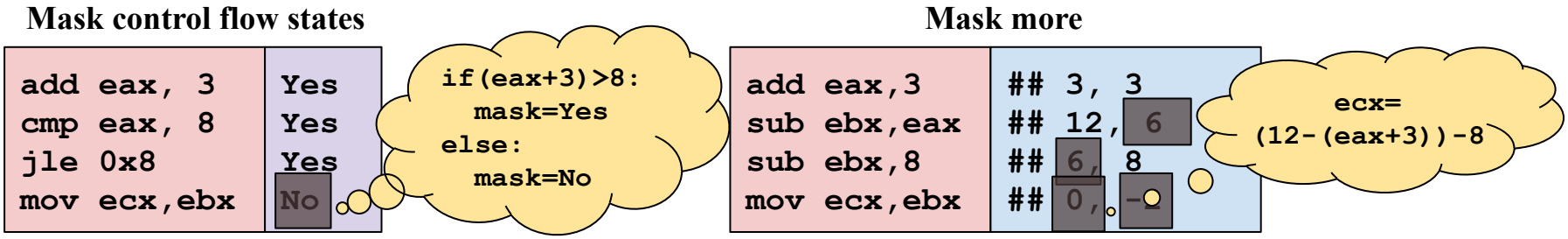
Motivating Example

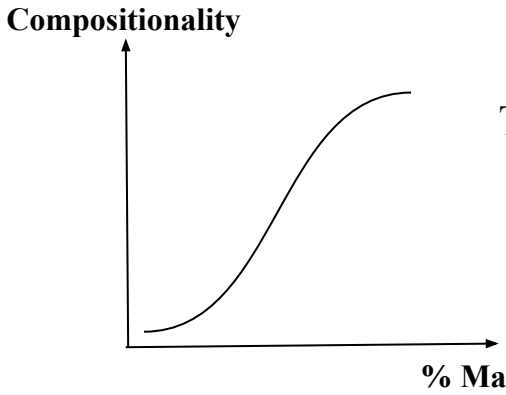


How to train the model to reason about program behavior?

How to train the model to reason about **control flow**?

How to train the model to reason about instruction **compositions**?

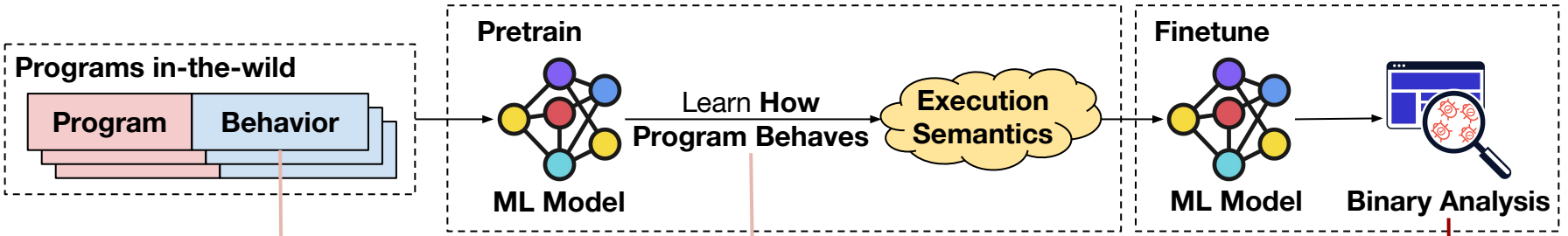




Training

- **Curriculum**: linearly increase % masks by training iterations
- **Randomized** mask selection at each iteration and samples

Challenges of Learning Execution Semantics

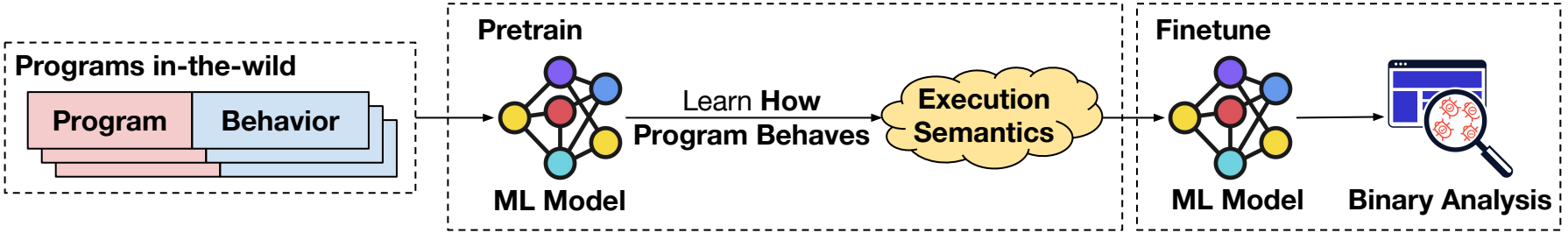


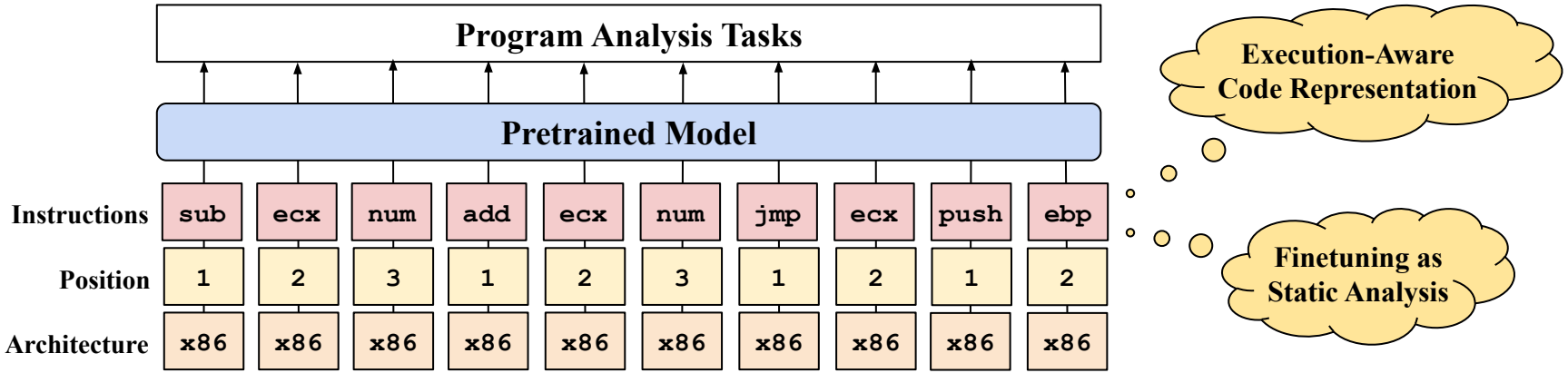
1. How to collect and represent the diverse program behavior?

2. How to train the model to reason about program behavior?

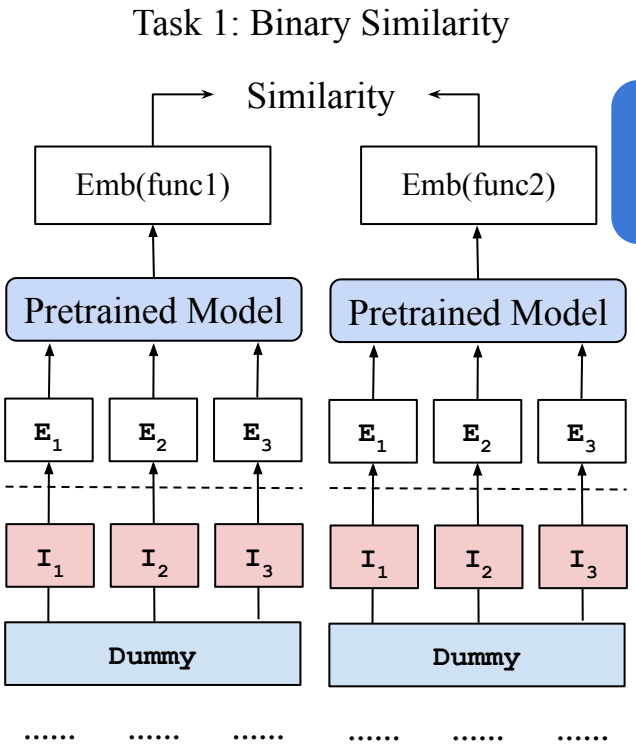
3. How to avoid the expensive dynamic analysis?

How to Avoid the Expensive Dynamic Analysis?

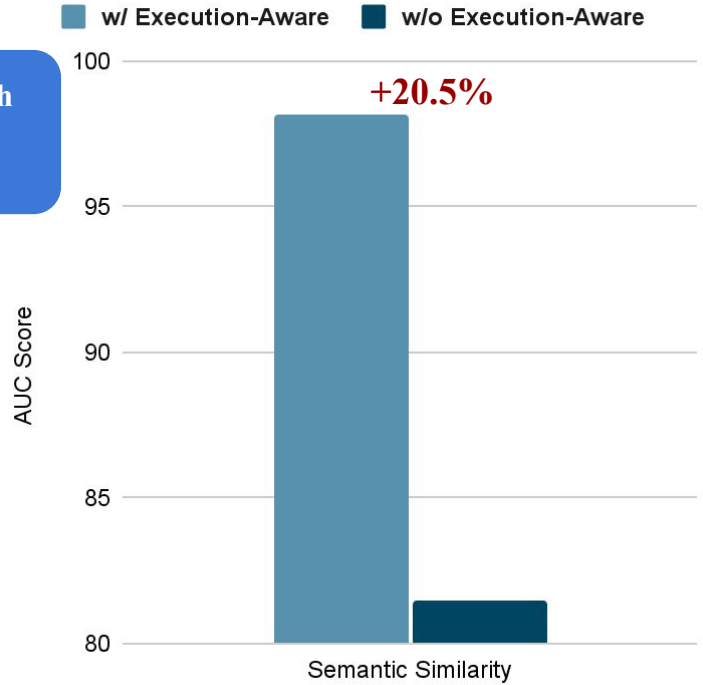




How Much do Learned Execution-Aware Program Representations Help?

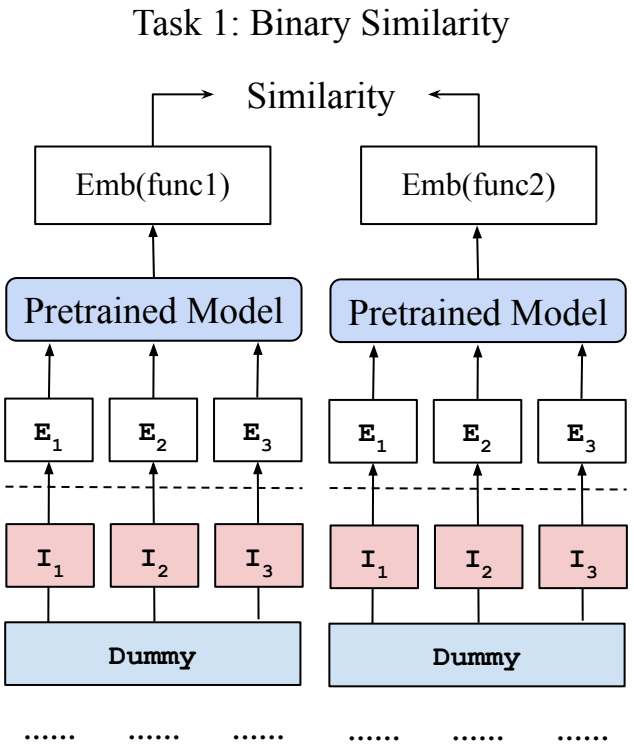


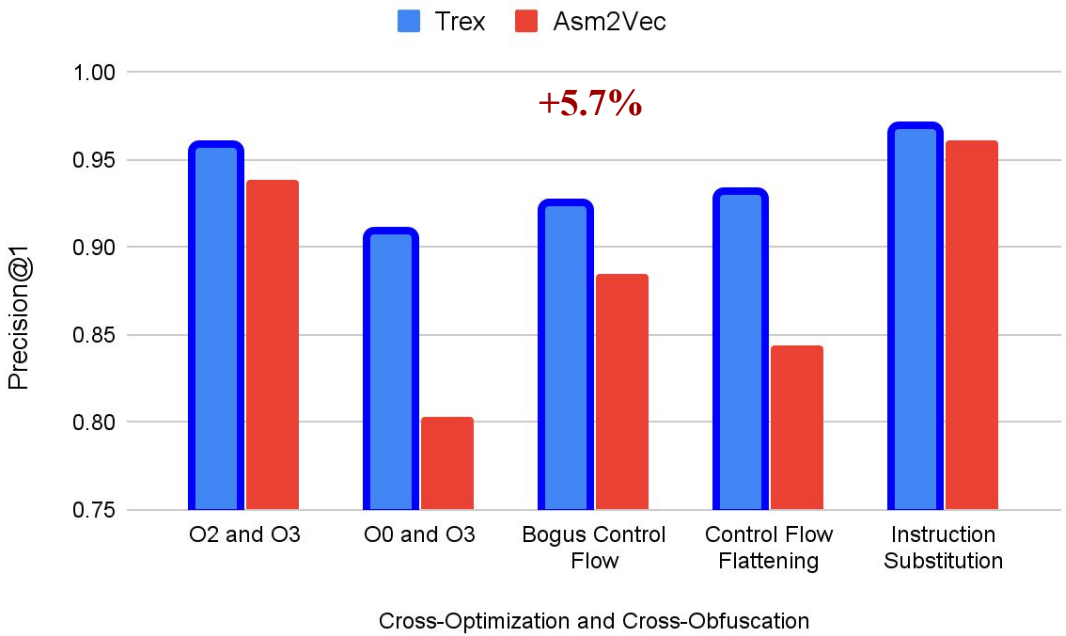
- Vulnerability search
- Malware detection
- Software patching



Pei, Xuan, Yang, Jana, Ray. Trex: Learning Execution Semantics from Micro-traces for Binary Similarity. TSE'22

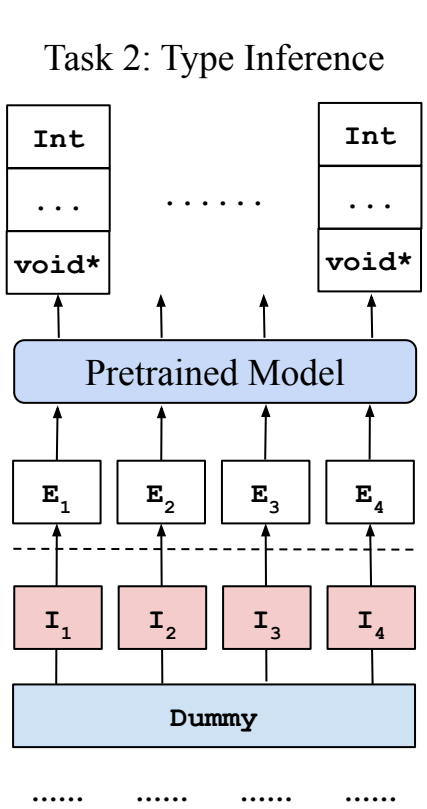
Finetuning for Matching Semantically Similar Binary Functions



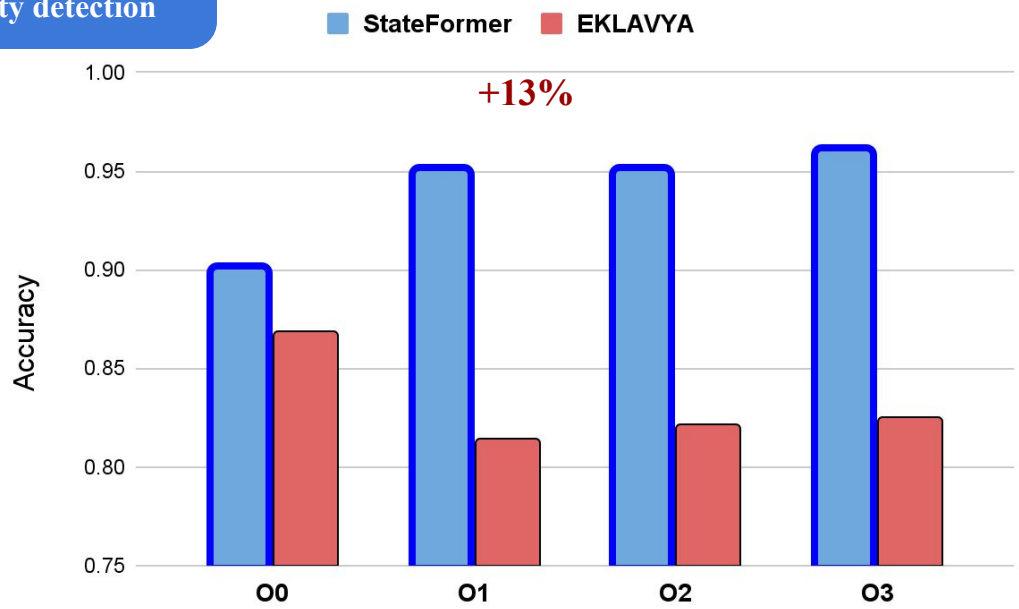


Pei, Xuan, Yang, Jana, Ray. Trex: Learning Execution Semantics from Micro-traces for Binary Similarity. TSE'22

Finetuning for Predicting Function Signatures and Type Inference



- Security retrofitting
- Decompilation
- Vulnerability detection

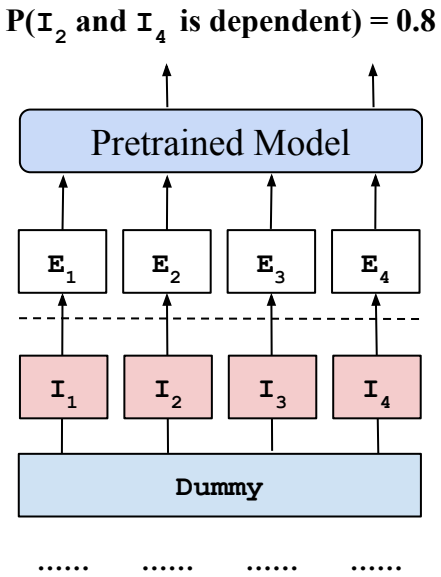


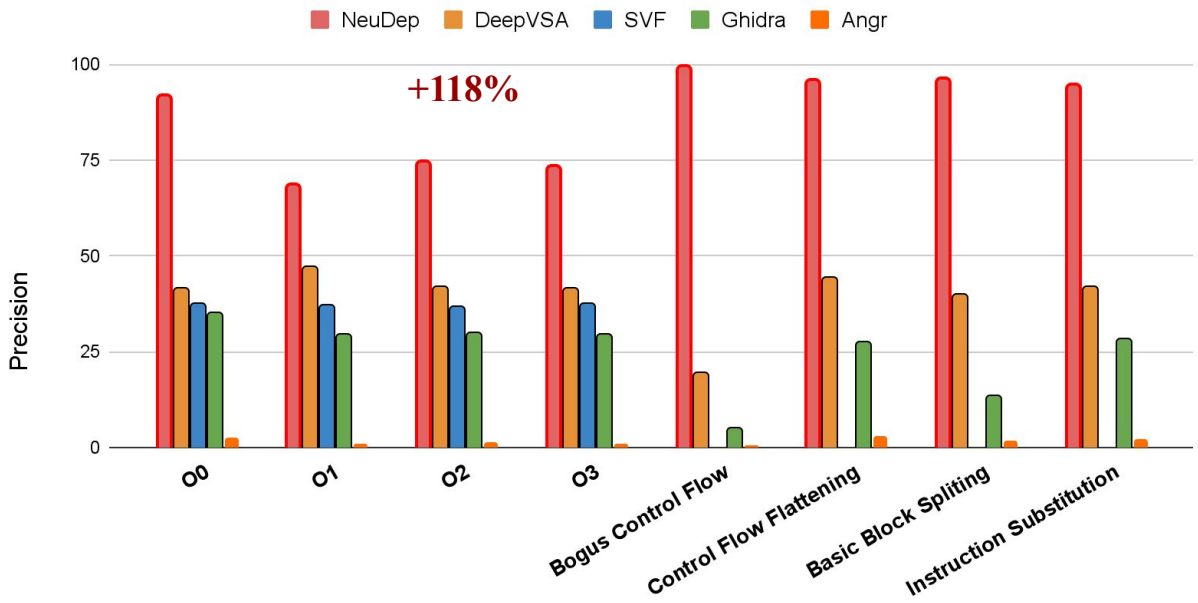
Pei, Guan, Broughton, Chen, Yao, Williams-King, Ummadisetty, Yang, Ray, Jana. StateFormer: Fine-grained Type Recovery from Binaries Using Generative State Modeling. ESEC/FSE'21

Finetuning for Analyzing Memory Dependence

Task 3: Memory Dependence Analysis

- **Taint analysis**
- **Malware Analysis**

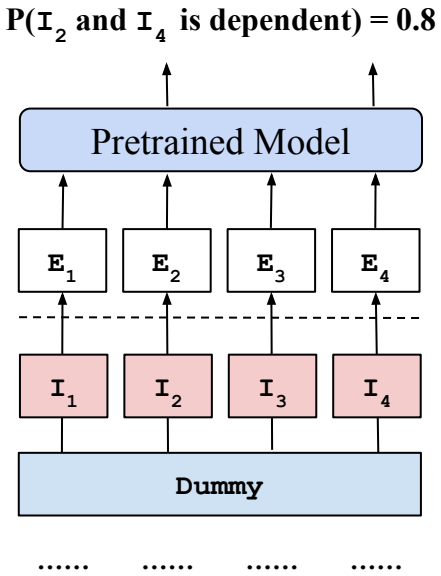


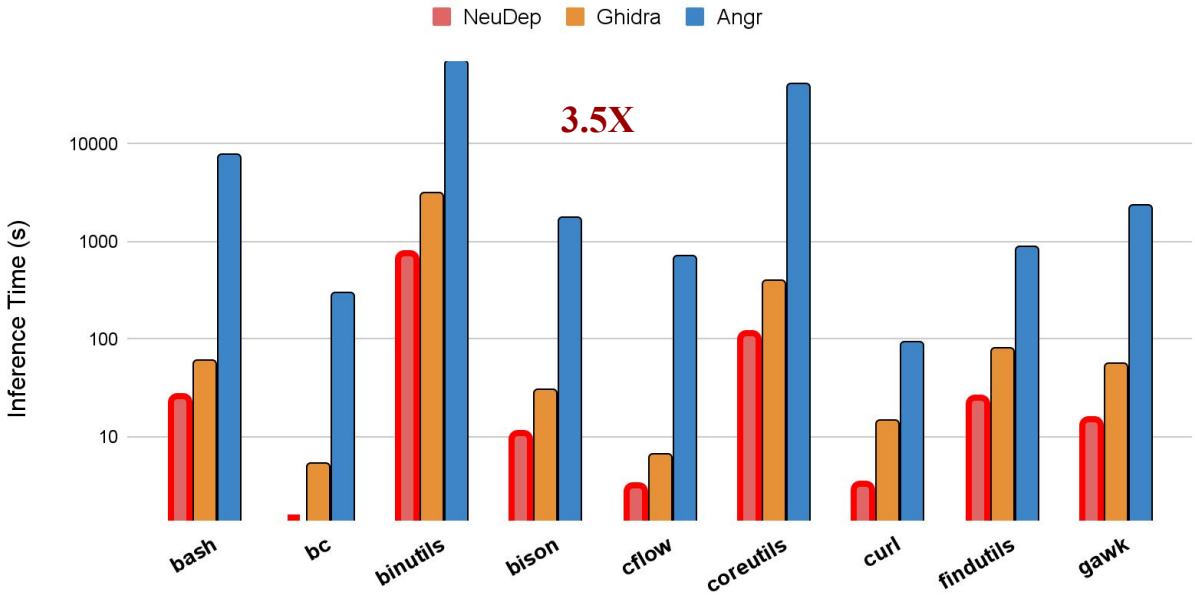


Pei, She, Wang, Geng, Xuan, David, Yang, Jana, Ray. NeuDep: Neural Binary Memory Dependence Analysis. ESEC/FSE'22

Finetuning for Analyzing Memory Dependence: Inference Time

Task 3: Memory Dependence Analysis

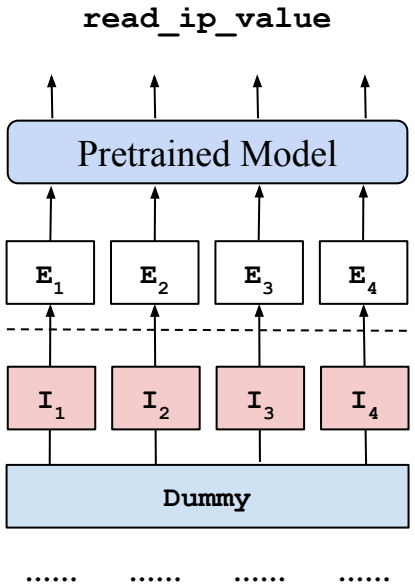




Pei, She, Wang, Geng, Xuan, David, Yang, Jana, Ray. NeuDep: Neural Binary Memory Dependence Analysis. ESEC/FSE'22

Finetuning for Function Name Prediction and Memory Region Prediction

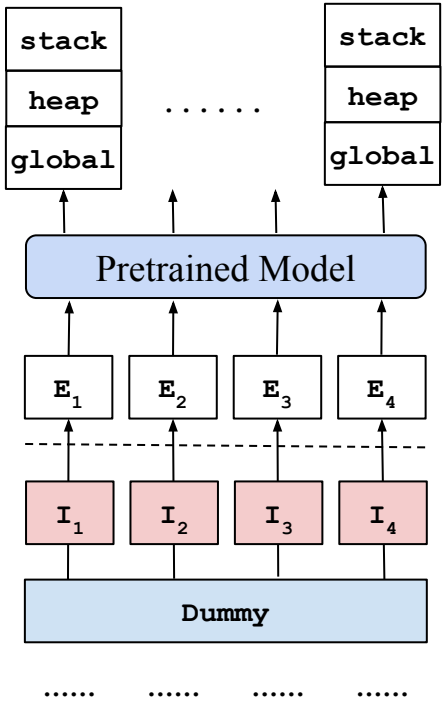
Task 4: Function Name Prediction



- Decompilation
- Reverse engineering

+35% Across All **Architectures** and **Optimizations**

Task 5: Memory Region Prediction



- Value set analysis
- Taint analysis

+7.9% Across All **Types of Memory Regions**

Jin, **Pei**, Wang, Won, Lin. NeuDep: SymLM: Predicting Function Names in Stripped Binaries via Context-Sensitive Execution-Aware Code Embeddings. CCS'22

Pei, She, Wang, Geng, Xuan, David, Yang, Jana, Ray. NeuDep: Neural Binary Memory Dependence Analysis. ESEC/FSE'22

Case Studies: Vulnerability Search in Firmware

CVE	Library	Description
CVE-2019-1563	OpenSSL	Decrypt encrypted message
CVE-2017-16544	BusyBox	Allow executing arbitrary code
CVE-2016-6303	OpenSSL	Integer overflow
CVE-2016-6302	OpenSSL	Allows denial-of-service
CVE-2016-2842	OpenSSL	Allows denial-of-service
CVE-2016-2182	OpenSSL	Allows denial-of-service
CVE-2016-2180	OpenSSL	Out-of-bounds read
CVE-2016-2178	OpenSSL	Leak DSA private key
CVE-2016-2176	OpenSSL	Buffer over-read
CVE-2016-2109	OpenSSL	Allows denial-of-service
CVE-2016-2106	OpenSSL	Integer overflow
CVE-2016-2105	OpenSSL	Integer overflow
CVE-2016-0799	OpenSSL	Out-of-bounds read
CVE-2016-0798	OpenSSL	Allows denial-of-service
CVE-2016-0797	OpenSSL	NULL pointer dereference
CVE-2016-0705	OpenSSL	Memory corruption

16 Vulnerabilities (Compiled in x86)

Search

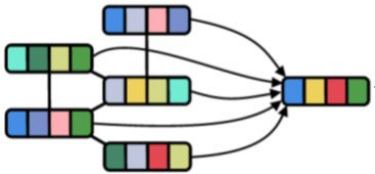
Firmware Images: `Arm`, `MIPS` with unknown compiler flags

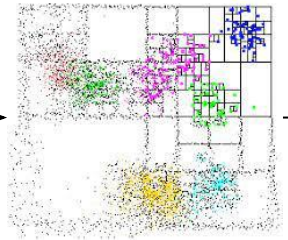

Ubiquiti sunMax — 15 CVEs


TP-Link Deco-M4 — 16 CVEs


NETGEAR R7000 — 7 CVEs


Linksys RE7000 — 14 CVEs

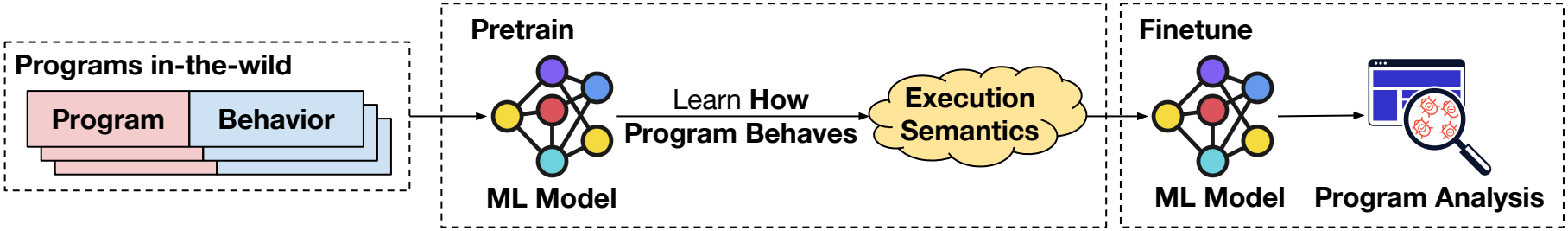

Learned Function Embeddings


Approximate Nearest Neighbors


Search over **1.4M** functions within **6.3 seconds**

Pei, Xuan, Yang, Jana, Ray. Trex: Learning Execution Semantics from Micro-traces for Binary Similarity. TSE'22

Summary: Learning **Program Semantics** via **Execution-Aware Pre-training** Improves Program Analysis for Security Applications



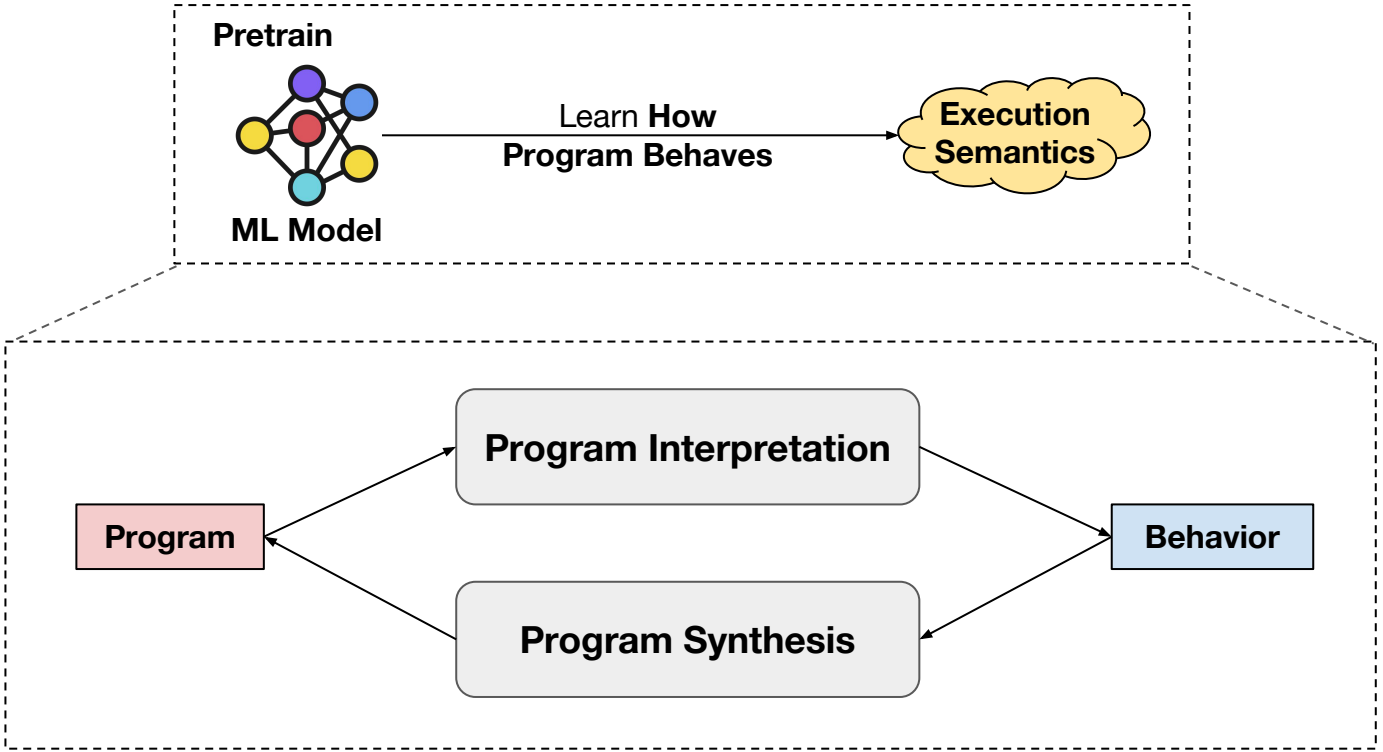
Precise: Outperforms the state-of-the-art by up to **118%**

Efficient: Speedup over the off-the-shelf tool by up to **98.1x**

Generalizable and Robust: Remains accurate across

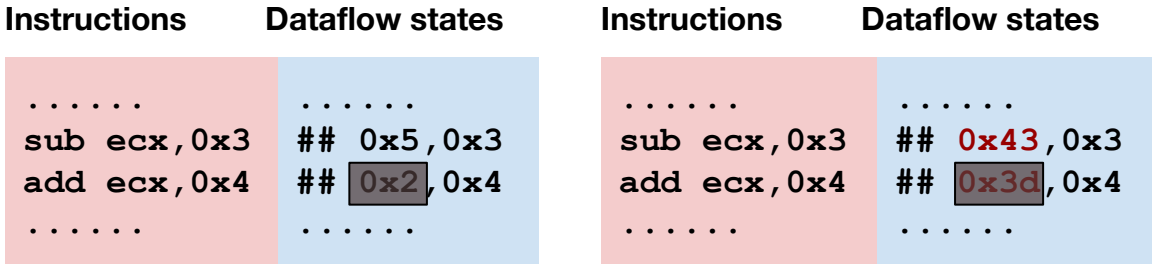
Compilers | Architectures | Optimizations | Obfuscations

Limitation: Learning Execution-Aware Program Representations is Challenging



Extremely challenging to learn precise semantics

Limitation: What the Model has Learned during Pretraining?



Perturb dataflow states from 0x5 to 0x43

Ground-truth	Top-1	Top-2
0x2	0x2 (98%)	0x3 (2%)
0x3d	0x3a (28%)	0x33 (13%)

Does not extrapolate well

Machine Learning Security