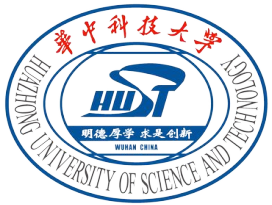
Exploiting Code Symmetries for Learning Program Semantics

Kexin Pei, Weichen Li*, Qirui Jin*, Shuyang Liu, Scott Geng
Lorenzo Cavallaro, Junfeng Yang, Suman Jana



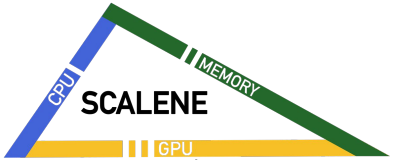
Machine Learning Shows Promise for Analyzing Programs

Detecting and Exploiting Vulnerabilities

Brandon Rossi
@0xConda

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

Python Profiler



GitHub Copilot

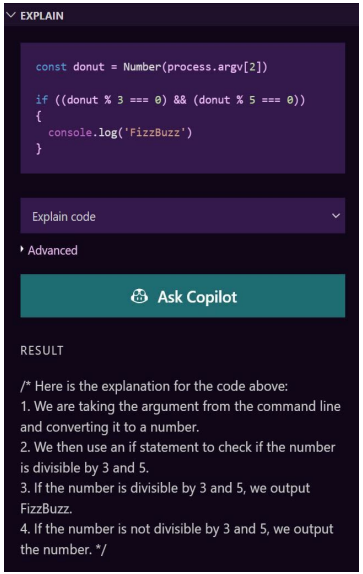
GPT-4

Program Optimization

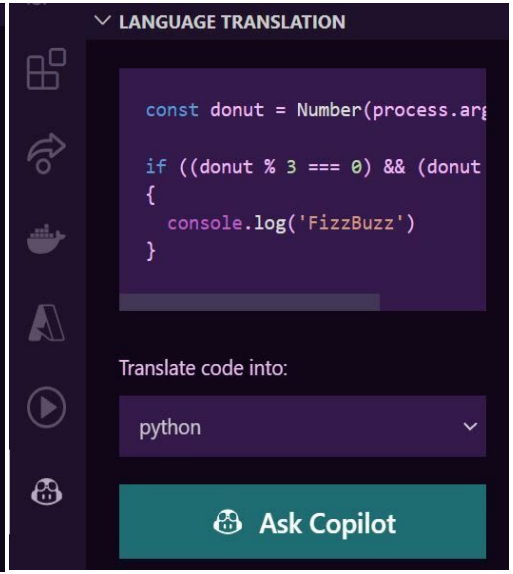
```
⚡    z1 = [i for i in range(0,300000)][299999]

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

Explain Code



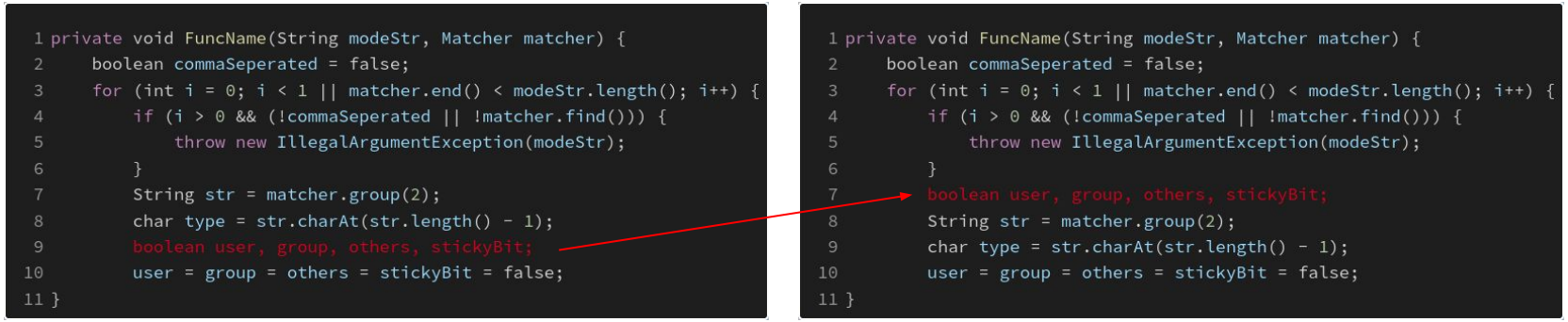
Translate Code



Limitations: Lack Understanding of Program Semantics

A code summarization based on GGNN (Li et al. 2017, Rabin et al. 2020, Bui et al. 2021)

Permute lines 7,8,9 (move line 9 before line 8).



```
1 private void FuncName(String modeStr, Matcher matcher) {
2     boolean commaSeperated = false;
3     for (int i = 0; i < 1 || matcher.end() < modeStr.length(); i++) {
4         if (i > 0 && (!commaSeperated || !matcher.find())) {
5             throw new IllegalArgumentException(modeStr);
6         }
7         String str = matcher.group(2);
8         char type = str.charAt(str.length() - 1);
9         boolean user, group, others, stickyBit;
10        user = group = others = stickyBit = false;
11 }
```

```
1 private void FuncName(String modeStr, Matcher matcher) {
2     boolean commaSeperated = false;
3     for (int i = 0; i < 1 || matcher.end() < modeStr.length(); i++) {
4         if (i > 0 && (!commaSeperated || !matcher.find())) {
5             throw new IllegalArgumentException(modeStr);
6         }
7         boolean user, group, others, stickyBit;
8         String str = matcher.group(2);
9         char type = str.charAt(str.length() - 1);
10        user = group = others = stickyBit = false;
11 }
```

“Run” “Update”

Consequences: Lacking Robustness and Generalization

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

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

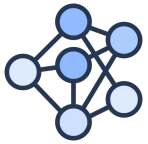
Existing Practice of Teaching ML Code Semantics

Common practice 1

Program

Data Augmentation

Semantics-preserving code transformations

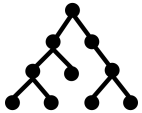


Code Models

Common practice 2

Program

Grounding

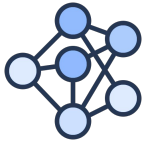


Structures

Execution Traces

Natural Language

Other code modalities



Code Models

Limitation: Expensive and No Guarantee

I. **Expensive:** Enumerate samples with different transformations and modalities

II. **No guarantee:** the make can still make mistakes on seen transformed samples

How to Specify Code Semantics and Build ML Models Respecting it?

Operational Semantics

$$\textsc{Pre}\ \frac{}{\alpha.t \xrightarrow{true,\alpha} t} \qquad \textsc{Cond}\ \frac{t \xrightarrow{b',\alpha} t'}{\texttt{if } b \texttt{ then } t \xrightarrow{b' \wedge b,\alpha} t'}\ fv(b) \cap bv(\alpha) = \emptyset$$

$$\textsc{Sum1}\ \frac{t \xrightarrow{b,\alpha} t'}{t + u \xrightarrow{b,\alpha} t'} \qquad \textsc{Sum2}\ \frac{u \xrightarrow{b,\alpha} u'}{t + u \xrightarrow{b,\alpha} u'}$$

$$\textsc{Par1}\ \frac{t \xrightarrow{b,\alpha} t'}{t \mid u \xrightarrow{b,\alpha} t' \mid u}\ bv(\alpha) \cap fv(u) = \emptyset \qquad \textsc{Par2}\ \frac{u \xrightarrow{b,\alpha} u'}{t \mid u \xrightarrow{b,\alpha} t \mid u'}\ bv(\alpha) \cap fv(t) = \emptyset$$

$$\textsc{Com}\ \frac{t \xrightarrow{b,c?x} t' \quad u \xrightarrow{b',c!e} u'}{t \mid u \xrightarrow{b \wedge b',\tau} t'[e/x] \mid u'} \qquad \textsc{Res}\ \frac{t \xrightarrow{b,\alpha} t'}{t \backslash L \xrightarrow{b,\alpha} t' \backslash L}\ Chan(\alpha) \notin L$$

$$\textsc{Ren}\ \frac{t \xrightarrow{b,\alpha} t'}{t[f] \xrightarrow{b,f(\alpha)} t'[f]} \qquad \textsc{Rec}\ \frac{t[\overline{e}/\overline{x}] \xrightarrow{b,\alpha} t'}{P(\overline{e}) \xrightarrow{b,\alpha} t'}\ P(\overline{x}) \Leftarrow t$$

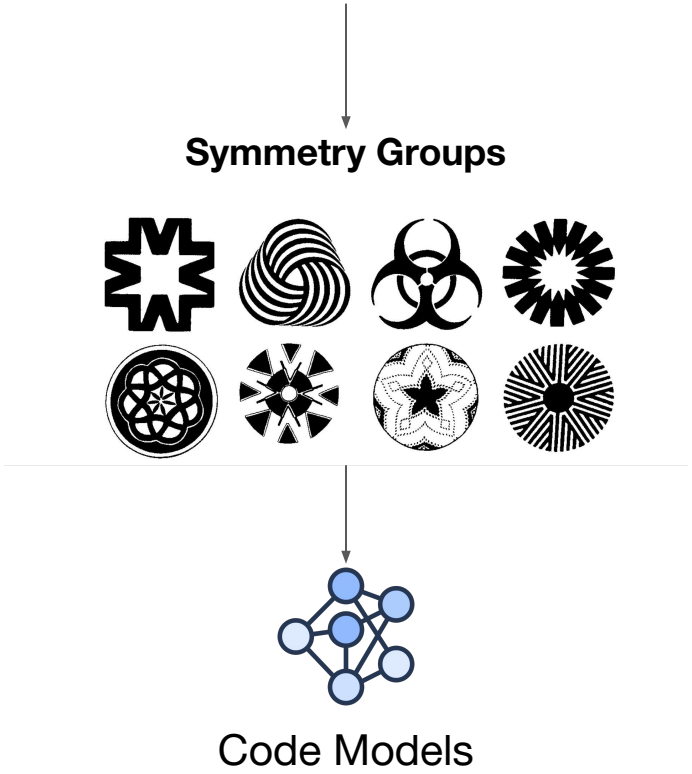
Bound by

- execution
- interpretation
- fix-point
- syntax

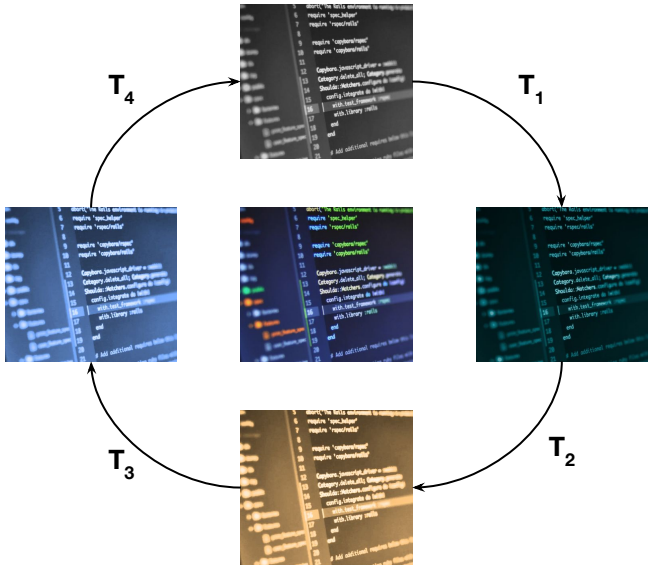
- **Interpreter**
- **Static Analysis**
- **…**

ML Models
e.g., LLMs

A New Group-Theoretic Framework to Formalize Learning Code Semantics



Code Symmetry Group



$T_i \in \boldsymbol{T}$

Semantics-preserving code transformations

***T** is a **Group** iff*

- ***Associative**: $T_i \cdot (T_j \cdot T_k) = (T_i \cdot T_j) \cdot T_k$*
- ***Inverse**: $\forall T_i \in T,\ \exists T_j \in T,\ T_i \cdot T_j = \boldsymbol{1}$*
- ***Identity**: $\forall \boldsymbol{1} \in T,\ \boldsymbol{1} \cdot T_i = T_i \cdot \boldsymbol{1}$*
- ***Closure**: $\forall T_i, T_j \in T,\ (T_i \cdot T_j) \in T$*

E.g., Permutation Group *T*

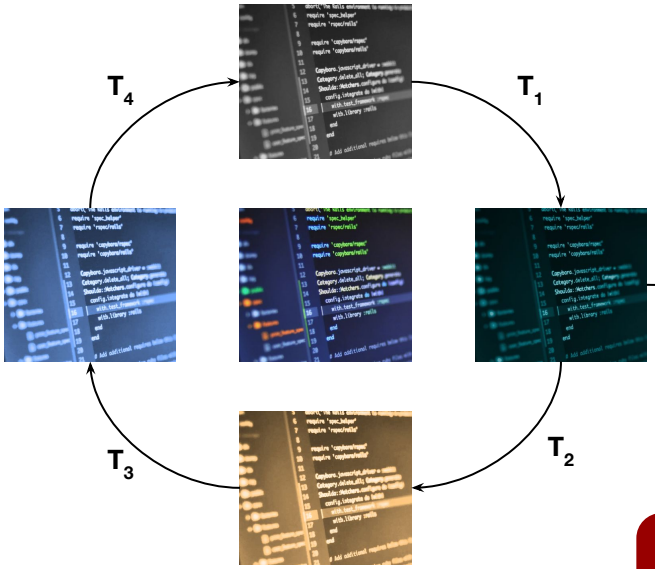
```
1: x=5                          1: y=6
2: y=6      Instruction         2: x=5
3: z=x+y    Reordering          3: z=x+y
```

.....

Key Benefit

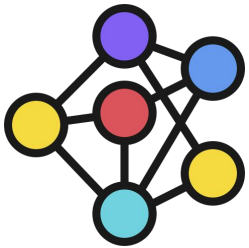
- Compositionality
  - Prove by Induction
- Amenable to Neural Architecture Design
  - AI4Science
  - Geometric DL

SymC: Permutation-Group-Invariant Transformer



Semantics-preserving permutations form a symmetry group **T**

Transformer Encoder
+
FF prediction head



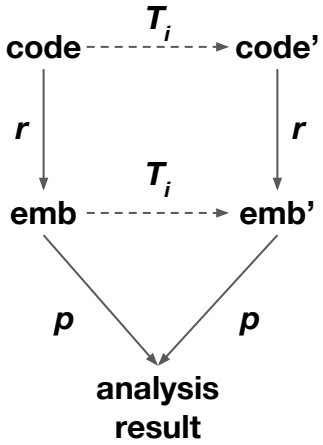
SymC

$$f(\boldsymbol{T}_i(\boldsymbol{P})) = f(\boldsymbol{P}) \quad \forall\, T_i \in \boldsymbol{T}$$

Data-augmentation-free
Pre-training-free

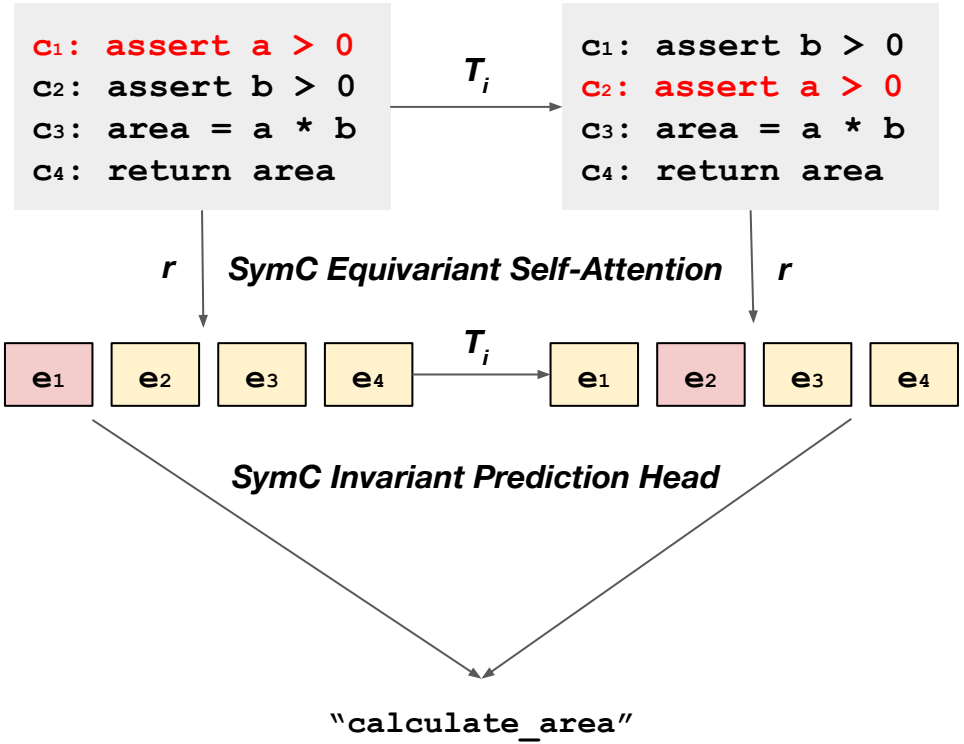
Provable Robustness
Guarantee by Construction

Group Invariance and Equivariance in SymC



r: representation learning module, e.g., self-attention layers

p: predictive learning module, e.g., fully-connected prediction head



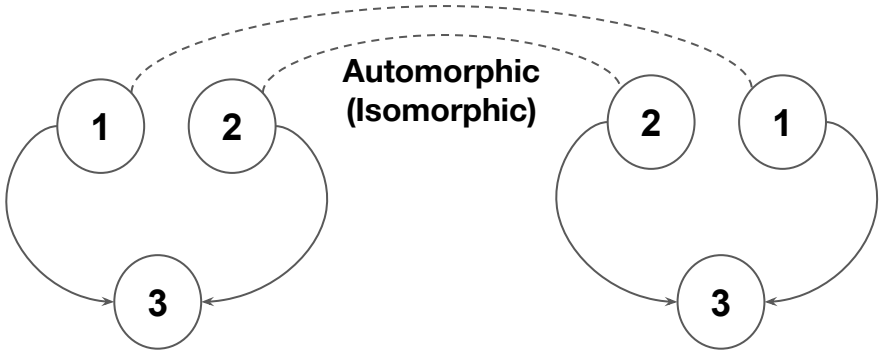
Aut(PDG): The Automorphism Group of Program Dependence Graph

```
1 int a = 1;
2 int b = 2;
3 int c = a + b;
```

```
1 int b = 2;
2 int a = 1;
3 int c = a + b;
```

- ***Data Dependency***
  - ***RAW****: Read after Write*
  - ***WAR****: Write after Read*
  - ***WAW****: Write after Write*
- ***Control Dependency***

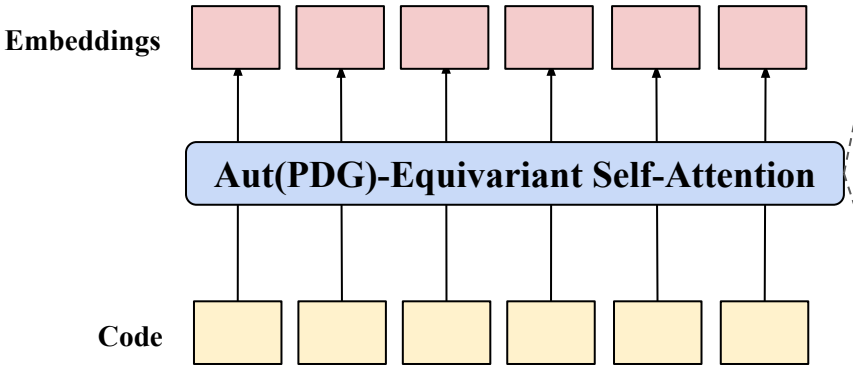
Program Dependence Graph **Program Dependence Graph**



Distance Matrix ***d***:
Relative distance to the lowest common ancestor (LCA)

0	inf	0
inf	0	0
1	1	0

Aut(PDG)-Equivariant Self-Attention



$Attn(x) = V \cdot (K^T \cdot Q)$

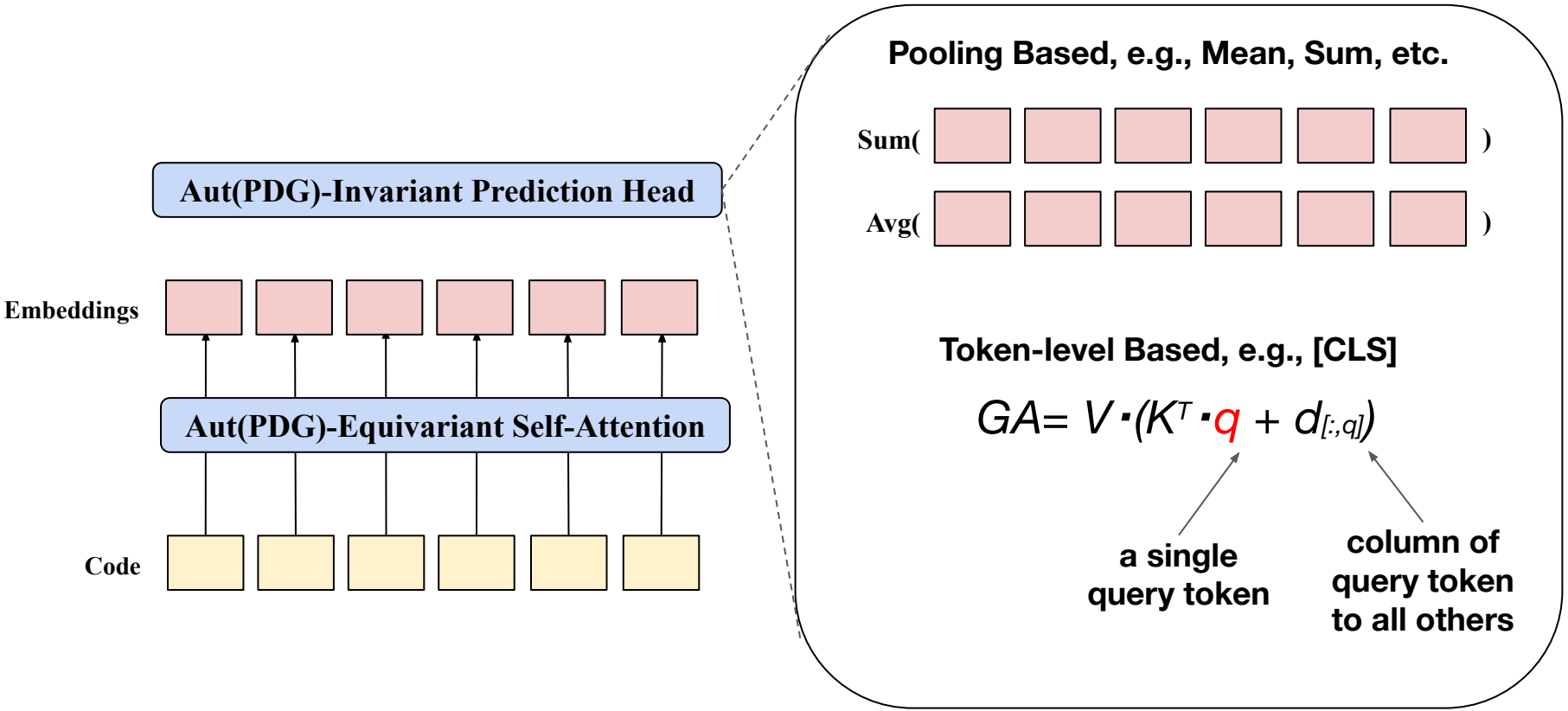
$Attn(xp) = Vp \cdot ((Kp)^T \cdot Qp)$
$= Vpp^T(K^T \cdot Q)p = V \cdot (K^T \cdot Q)p = Attn(x)p$

How to make it equivariant ***only*** ***to*** *Aut(PDG), the semantics-preserving permutations?*

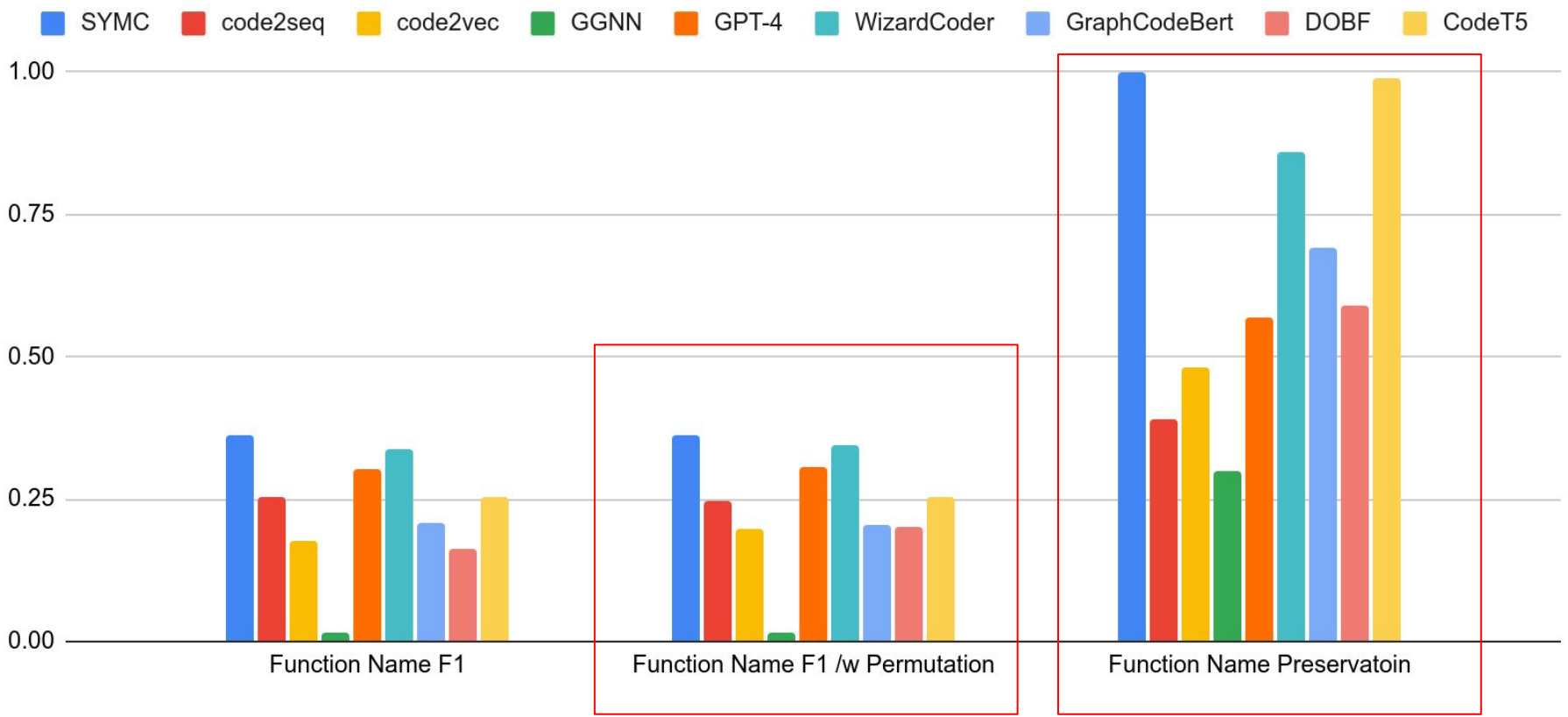
$G\text{-}Attn(x) = V \cdot (K^T \cdot Q + \boldsymbol{d_{PDG}})$

$G\text{-}Attn(xp) = Vp \cdot ((Kp)^T \cdot Qp + \boldsymbol{d_{PDG}})$
$= Vpp^T(K^T \cdot Q)p + Vp\boldsymbol{d_{PDG}}$
$= V \cdot (K^T \cdot Q + \boldsymbol{d_{PDG}})p = G\text{-}Attn(x)p$

Aut(PDG)-Invariant Prediction Head

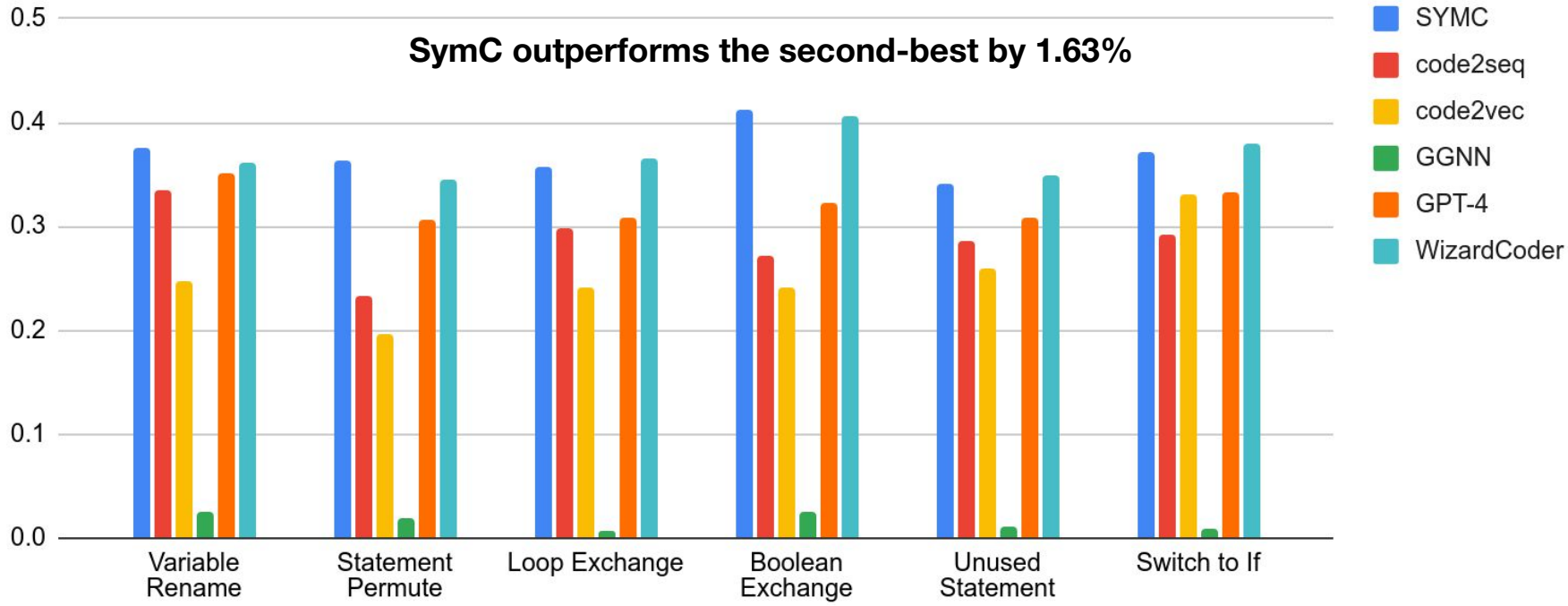


Results: Robustness Against Permutation



Results: Generalization against Unseen Transformations

The performance (F1) of SYMC and baselines against different unseen permuted code transformations.



Results: Invariance and Generalization