Is “Naturalness" a result of deliberate choice?

Kevin Lee and Casey Casalnuovo
Adviser: Prem Devanbu
University of California, Davis

Grant #1414172
Exploiting the Naturalness of Software

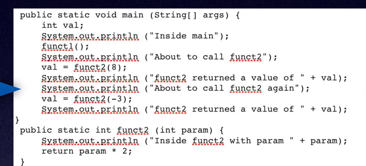
- Programs are written to be read

"Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on ***explaining to human beings what we want a computer to do****."* [Don Knuth]

The Dual Channel Hypothesis (Barr/Sutton)

```
public static void main (String[] args) {
     int val;
     System.out.println ("Inside main");
     funct1();
     System.out.println ("About to call funct2");
     val = funct2(8);
     System.out.println ("funct2 returned a value of " + val);
     System.out.println ("About to call funct2 again");
     val = funct2(-3);
     System.out.println ("funct2 returned a value of " + val);
}
public static int funct2 (int param) {
     System.out.println ("Inside funct2 with param " + param);
     return param * 2;
}
```

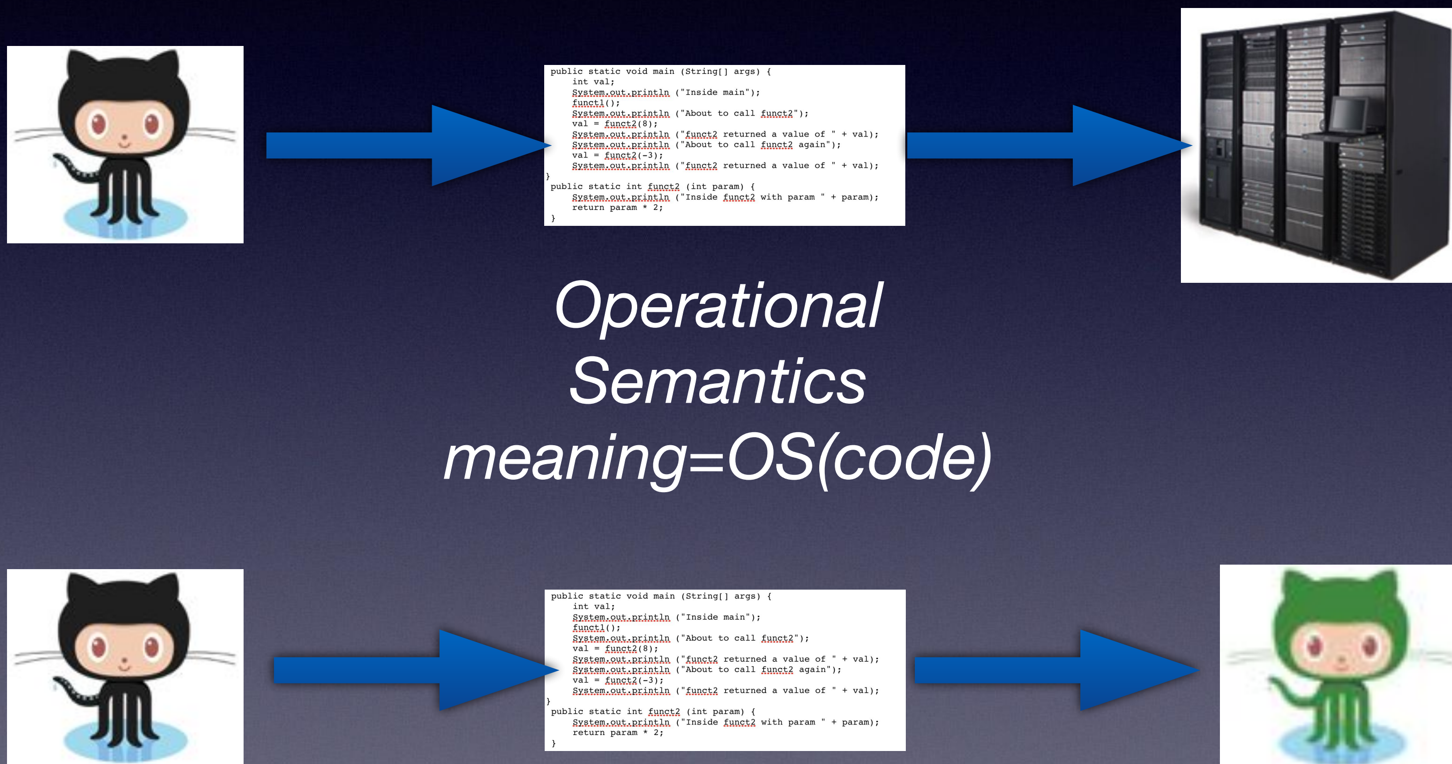
The Dual Channel Hypothesis
(Barr/Sutton)



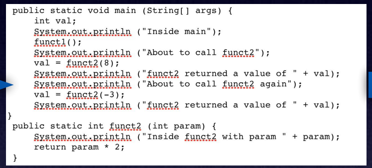
```
public static void main (String[] args) {
    int val;
    System.out.println ("Inside main");
    funct1();
    System.out.println ("About to call funct2");
    val = funct2(8);
    System.out.println ("funct2 returned a value of " + val);
    System.out.println ("About to call funct2 again");
    val = funct2(-3);
    System.out.println ("funct2 returned a value of " + val);
}
public static int funct2 (int param) {
    System.out.println ("Inside funct2 with param " + param);
    return param * 2;
}
```

Operational
Semantics
meaning=OS(code)

The Dual Channel Hypothesis (Barr/Sutton)

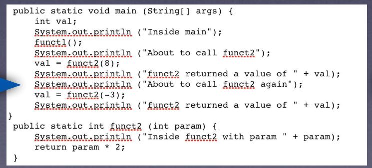


The Dual Channel Hypothesis (Barr/Sutton)



```
public static void main (String[] args) {
    int val;
    System.out.println ("Inside main");
    funct1();
    System.out.println ("About to call funct2");
    val = funct2(8);
    System.out.println ("funct2 returned a value of " + val);
    System.out.println ("About to call funct2 again");
    val = funct2(-3);
    System.out.println ("funct2 returned a value of " + val);
}
public static int funct2 (int param) {
    System.out.println ("Inside funct2 with param " + param);
    return param * 2;
}
```

Operational Semantics
meaning=OS(code)




```
public static void main (String[] args) {
    int val;
    System.out.println ("Inside main");
    funct1();
    System.out.println ("About to call funct2");
    val = funct2(8);
    System.out.println ("funct2 returned a value of " + val);
    System.out.println ("About to call funct2 again");
    val = funct2(-3);
    System.out.println ("funct2 returned a value of " + val);
}
public static int funct2 (int param) {
    System.out.println ("Inside funct2 with param " + param);
    return param * 2;
}
```


Identifiers, structural patterns, program style….

The Dual Channel Hypothesis
(Barr/Sutton)

```
public static void main (String[] args) {
    int val;
    System.out.println ("Inside main");
    funct1();
    System.out.println ("About to call funct2");
    val = funct2(8);
    System.out.println ("funct2 returned a value of " + val);
    System.out.println ("About to call funct2 again");
    val = funct2(-3);
    System.out.println ("funct2 returned a value of " + val);
}
public static int funct2 (int param) {
    System.out.println ("Inside funct2 with param " + param);
    return param * 2;
}
```

Operational
Semantics
meaning=OS(code)

```
ublic static void main (String[] args) {
    int val;
    System.out.println ("Inside main");
    funct1();
    System.out.println ("About to call funct2");
    val = funct2(8);
    System.out.println ("funct2 returned a value of " + val);
    System.out.println ("About to call funct2 again");
    val = funct2(-3);
    System.out.println ("funct2 returned a value of " + val);

public static int funct2 (int param) {
    System.out.println ("Inside funct2 with param " + param);
    return param * 2;
```

$$argmax_{meaning} P(meaning \mid code)$$

The "noisy" H-H channel

m = meaning
c = code

$$argmax_m P(m \mid c)$$

Bayes Rule

$$= argmax_m \frac{P(c \mid m)\ P(m)}{P(c)}$$

For a given c

$$\approx argmax_m P(c \mid m)\ P(m)$$

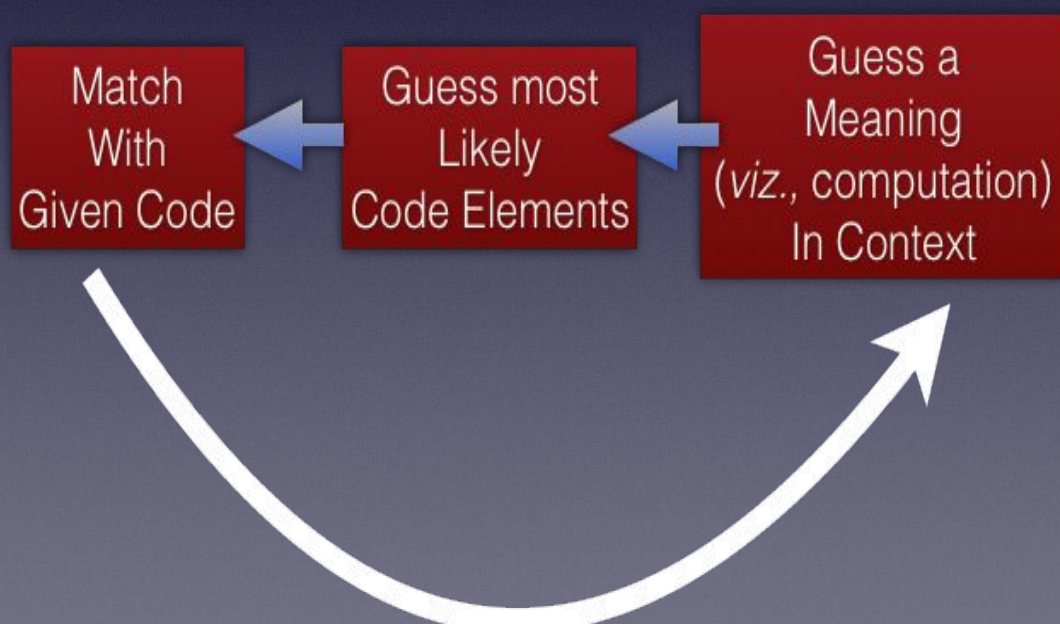
Informed by coding and domain knowledge

$$\approx argmax_m P(c \mid m)\ P(m \mid context)$$

The “noisy” H-H channel

$$argmax_{meaning} P(meaning \mid code)$$

$$\approx argmax_{meaning} P(code \mid meaning)\ P(meaning \mid context)$$

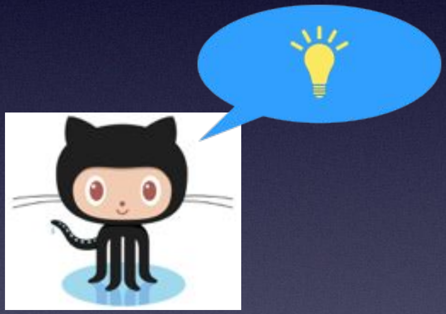


The “noisy” H-H channel

Guess a
Meaning
(*viz.*, computation)
In Context

$$argmax_m P(m \mid c)$$

$$\approx argmax_m P(c \mid m) \ P(m \mid context)$$



m = meaning
c = code

The “noisy” H-H channel

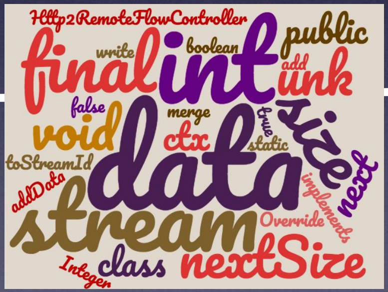
Guess a
Meaning
(*viz.*, computation)
In Context

$$argmax_m P(m \mid c)$$

$$\approx argmax_m P(c \mid m)\ P(m \mid context)$$

Guess most
Likely
Code Elements

m = meaning
c = code

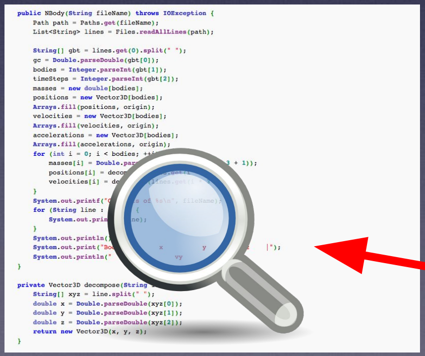


The “noisy” H-H channel

$$argmax_m P(m \mid c)$$

$$\approx argmax_m P(c \mid m)\ P(m \mid context)$$

m = meaning
c = code

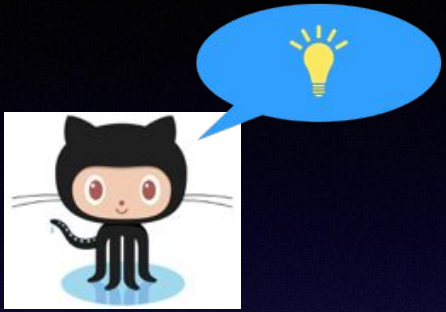


Guess a
Meaning
(*viz.*, computation)
In Context

Guess most
Likely
Code Elements

Match
With
Given Code

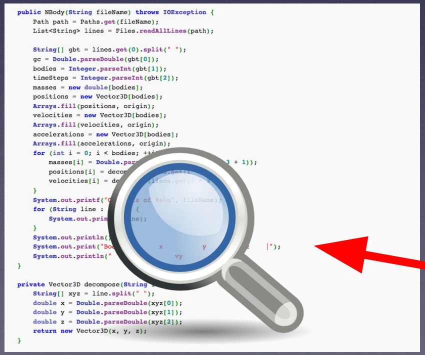
The "noisy" H-H channel



$$argmax_m P(m \mid c)$$

$$\approx argmax_m P(c \mid m)\ P(m \mid context)$$

m = meaning
c = code



Guess a Meaning (*viz.*, computation) In Context

Guess most Likely Code Elements

Match With Given Code

The “noisy” H-H channel

Guess a Meaning (*viz.*, computation) In Context

Guess most Likely Code Elements

Match With ven Code

$$argmax_m P(m \mid c)$$

$$\approx argmax_m P(c \mid m)\ P(m \mid context)$$

m = meani
c = code

This ‘guessing’ will be more effective if p(c|m) is ‘skewed’.

So what?

Given a computation to implement:

Programmers tend to favor one implementation over others.

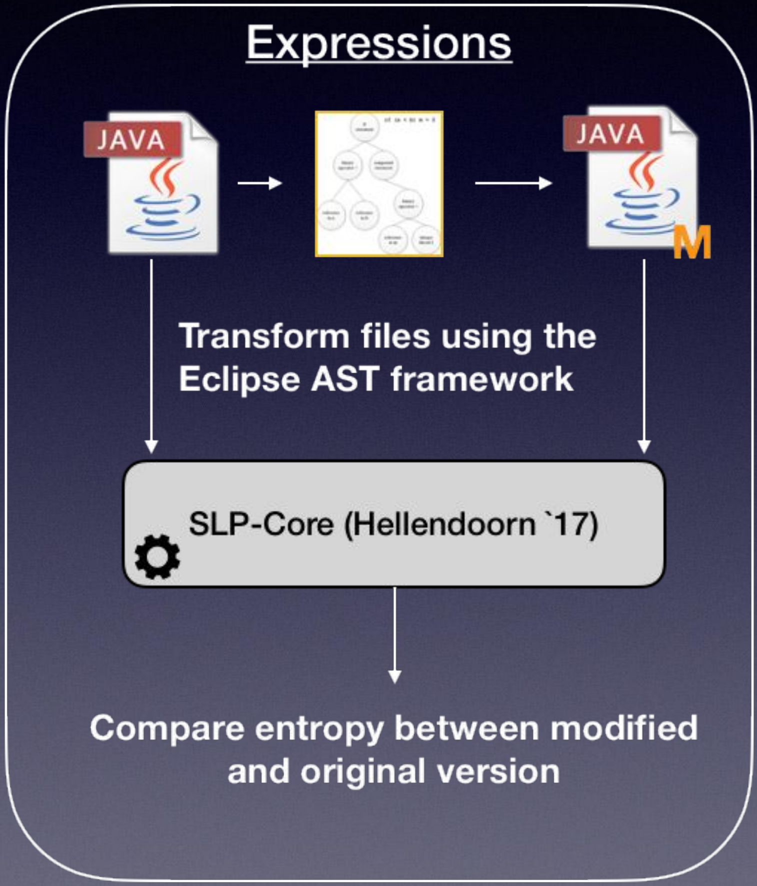
So if $C_1, C_2, \ldots C_n$ *are different, equivalent implementations of the same computation M, usually:*

$$\exists j : p(C_j \,|\, M) \gg p(C_i \,|\, M) \quad \forall i = 1 \ldots n, i \neq j$$

General Scheme

- Estimate a language model *LM* over a large corpus
- Using *LM*, Measure the entropy of “natural” program ***S*** (not in training set)
- Apply meaning preserving transform to ***S*** yielding
- Using *LM, measure entropy of S and*
- ***Null hypothesis:*** there is no difference.
  Alternative hypothesis: S is lower in entropy, than since programmers have a definite preference.

How to test?



Estimate language model *LM* over a corpus (12 Java projects/~16-17 million tokens)

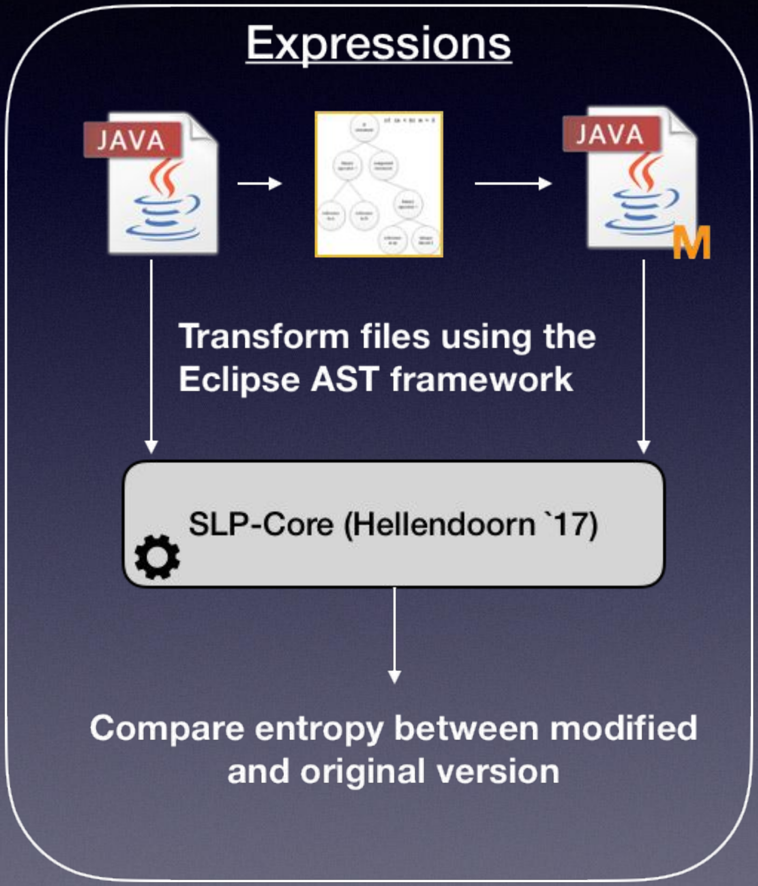
Using *LM*, Measure the entropy of "natural" program *S* (not in training set)

Apply **meaning preserving transforms** to *S* to create $\hat{S}_1, \hat{S}_2, \hat{S}_3 \ldots .$

Using *LM, measure entropy of S and*

$\hat{S}_1, \hat{S}_2, \hat{S}_3 \ldots .$

How to test?



Null hypothesis: There is no difference in entropy between S and $\hat{S}_1, \hat{S}_2, \hat{S}_3 \ldots.$

Alternative hypothesis: S is lower in entropy, than $\hat{S}_1, \hat{S}_2, \hat{S}_3 \ldots.$ since programmers have a definite preference.

Meaning Preserving Transform?

Focus on Expressions

Operator Commutation

A + B → B + A

A * B → B * A

Variable Name Swap

Int a
Int b

→

Int b
Int a

Adding Parenthesis

A + B * C

↓

A + (B * C)

Removing Parentheses

A + (B * C)

↓

A + B * C

Limit to operations without side effects to avoid changing meaning.

Future direction: *statement level transformations* (line, if block shuffling, etc)

Meaning Preserving Transform?

- Operator Commutation
- Superfluous parentheses removal
- Superfluous parentheses insertion
- Operator associativity (Pending)
- Renaming Variables within and across types
- Independent statement reordering (Pending)

Training/Testing

- Training:
  - 12 popular (Most Starred) Github projects
  - Manually selected to cover a diverse set of domains

- Testing (Projects with large number of numeric expressions)
  - Apache Commons Math Library, Biojava, Spring Framework by Pivotal

Language Models

- 6-gram with Jelinek-Mercer smoothing (‘global’)
- 6-gram-cache
- Above models with types from Pygments syntax highlighter
- Implemented using SLP-Core framework

Language Models

<u>Ordinary Ngram Models</u>

**return this . objectDepth = = 0 & &
((token = = JsonToken . START_ARRAY ...)) ;**

<u>Identifiers Replaced with Types</u>

**return this . Token_Name_Attribute = = 0 & & ((
Token_Name = = Token_Name .
Token_Name_Attribute ...)) ;**

Implemented in SLP-Core (Hellendoorn)

- Fast and easy to quickly update

Used Pygments to generate types for variables.

- Uncover structural patterns separate from identifiers.

Language Models

Ordinary Ngram Models

6-gram model

6-gram model
+ cache

Identifiers Replaced with Types

6-gram model

6-gram model
+ cache

Implemented in SLP-Core (Hellendoorn)

- Fast and easy to quickly update

Used Pygments to generate types for variables.

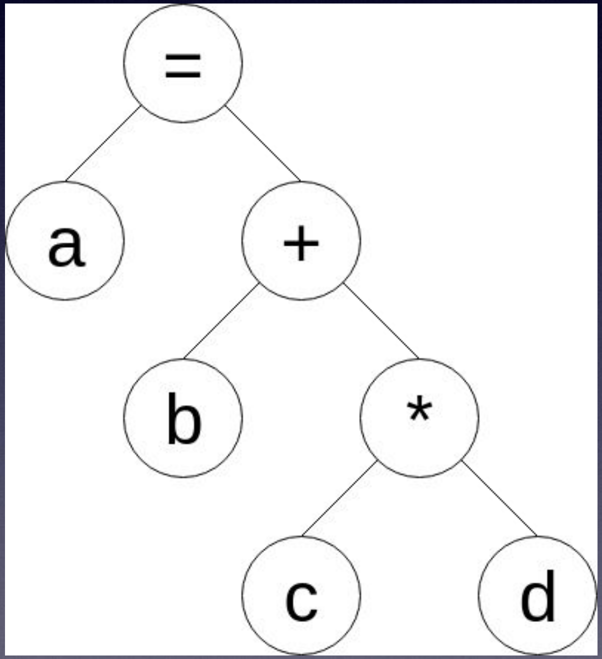
- Uncover structural patterns separate from identifiers.

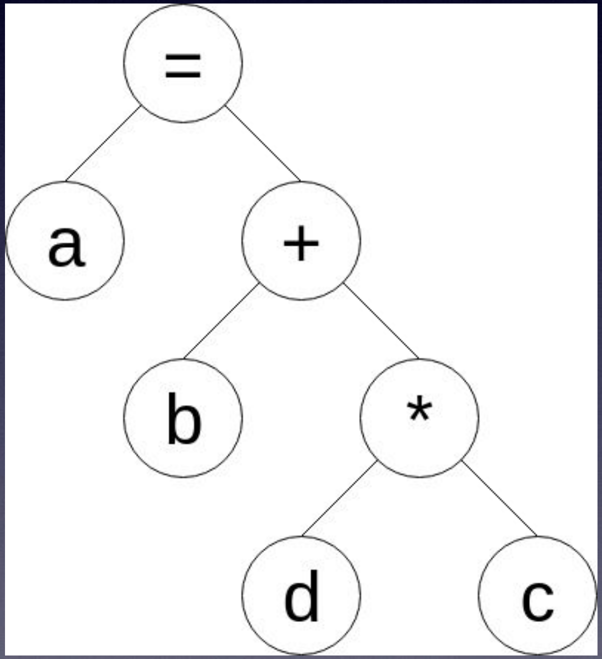
AST Transformations

- Abstract Syntax Tree
- Parser from Eclipse Foundation’s Java Development Tools (JDT) API

Commutation

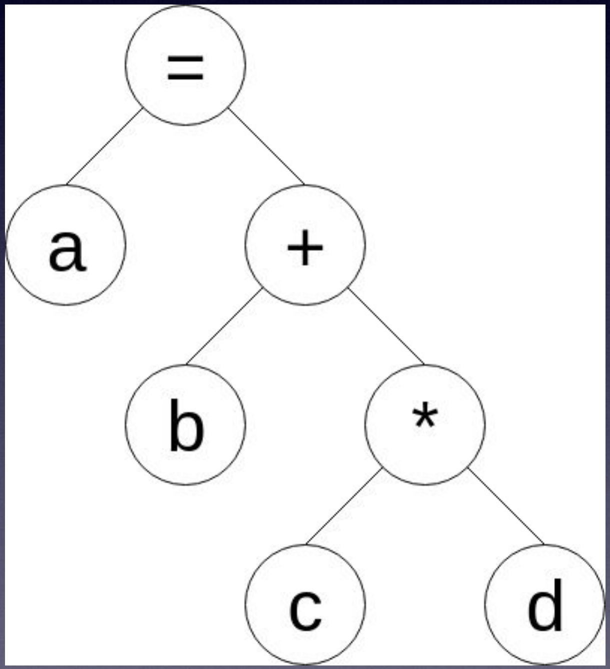
a = b + c * d -> a = b + d * c

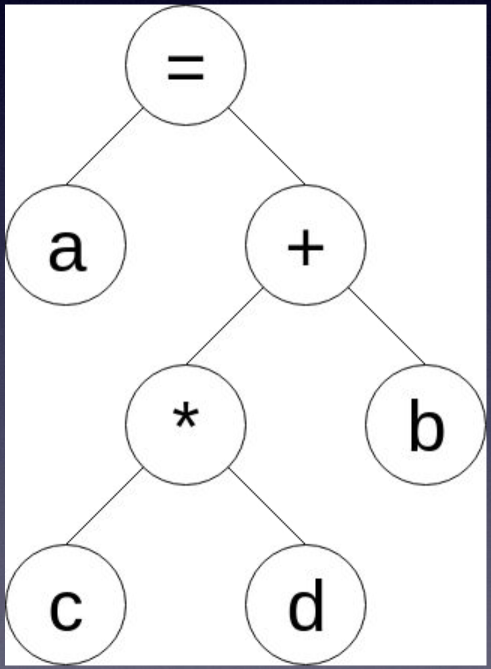




Commutation

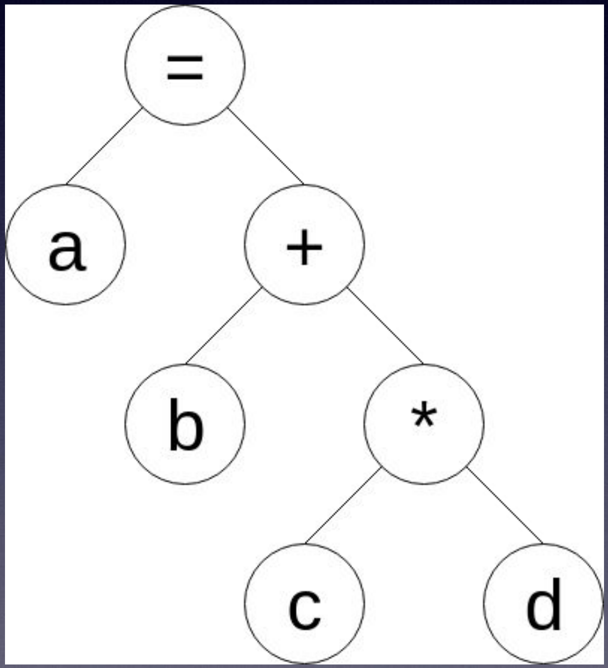
a = b + c * d -> a = d * c + b

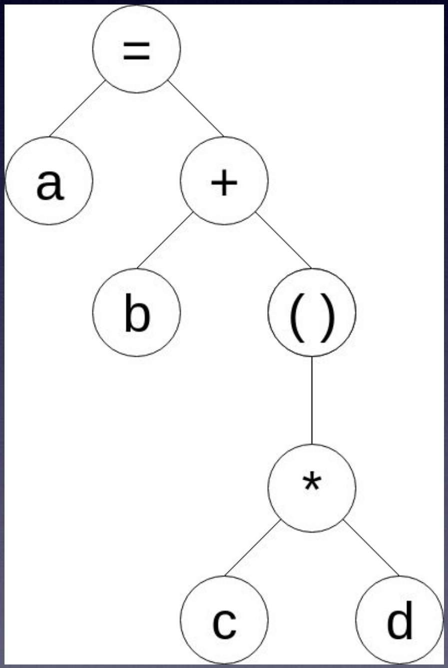




Parentheses Addition

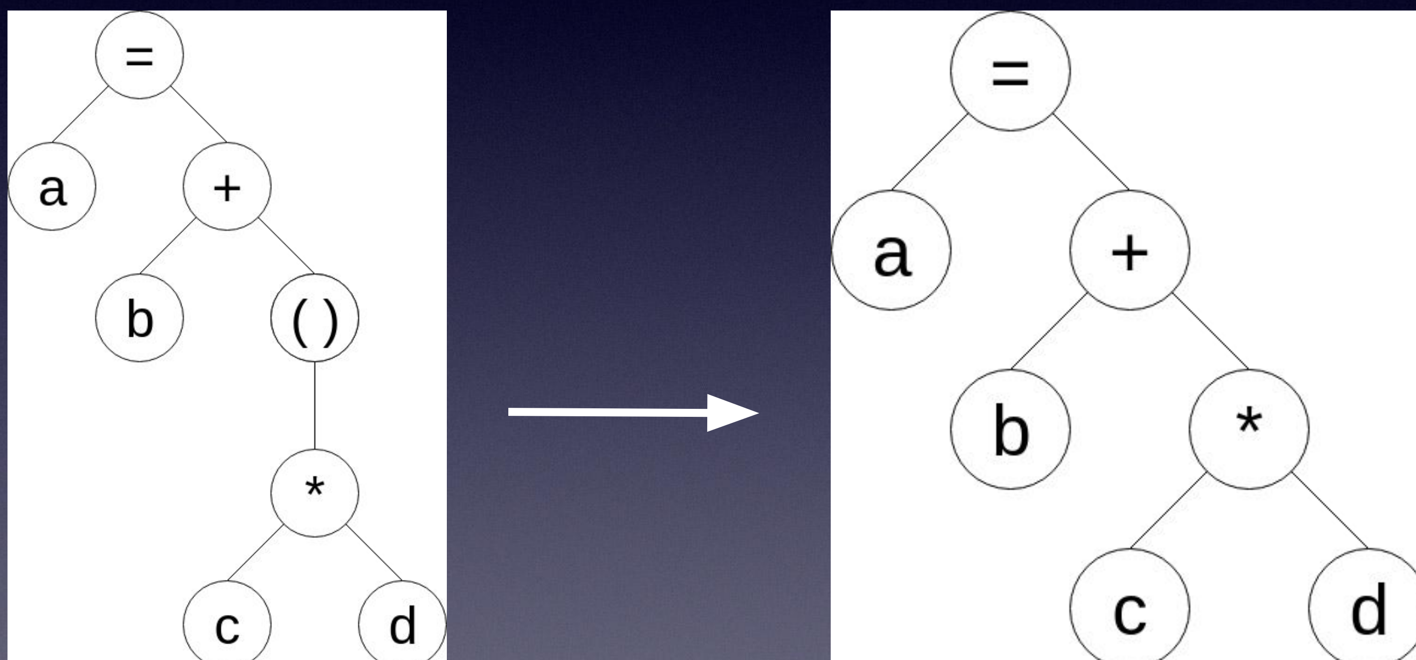
a = b + c * d -> a = b + (c * d)





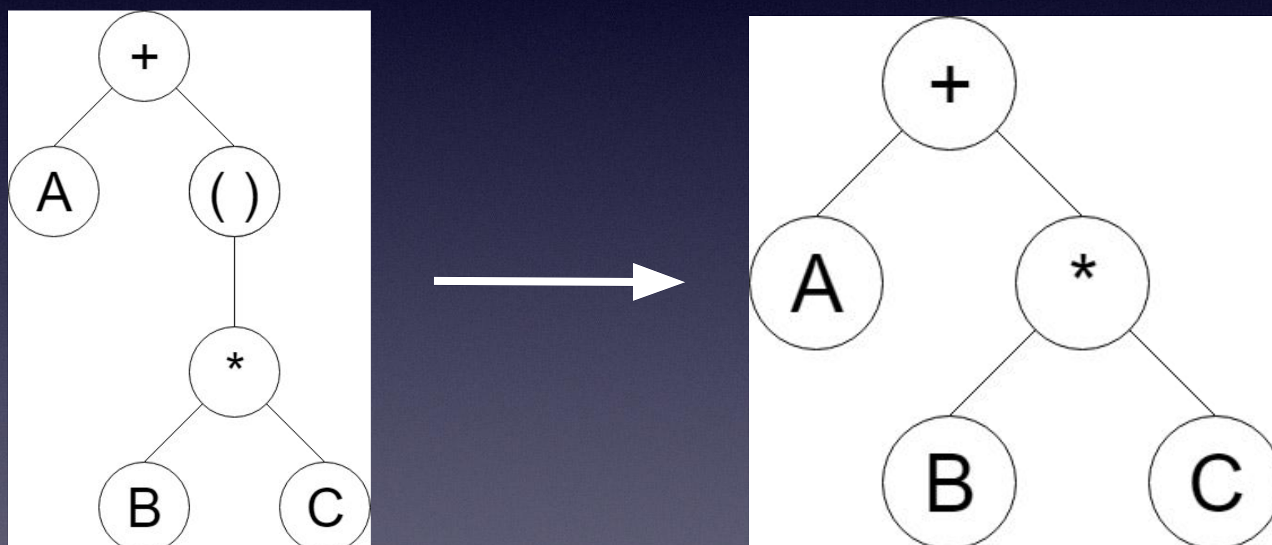
Parentheses Removal

a = b + (c * d) -> a = b + (c * d)



Parentheses Removal

A + (B * C) → A + B * C



Data Collected

- File Path
- Line Number
- Number of Tokens (Java and Pygments Type)
- Number of Transformations
- Average depth of transformed AST nodes
- Entropies
- Common Parent of Transformed Nodes
- Most and Least common operator involved

Number of Data Points

- 8,611 commutations
- 20,844 parentheses additions
- 2,670 parentheses removals

Results

What do we want to convey?

These transformations generally make code harder to predict (matching our theory) (Show Cohens D table?

Transformations relationship to the original entropy of the expression.

Certain ‘odd’ data out (add global model) + interesting interpretations of combined models.

Some ideas for future directions/gather feedback/ideas from audience.

A Note on Results

- Present results focusing on the change in average expression entropy.
  - Looked at line entropy as well (similar)

`X = A + ( B * C ) / D ;` → `H(B * C)`

- Include tokens that only appear in both the original and the changed lines (e.g. parenthesis) in average.

Do the models prefer the original?

- Paired T-tests comparing line before and after the change (all significant $p<.001$)

- Directed paired Cohen's D effect size.

	Global	Cache	Global Type	Cache Type
Operator Swap	0.313	0.859	0.360	0.706
Addition	-0.858	0.142	0.366	0.869
Removal	1.034	0.940	0.087	0.361

Summary

- When employing cache models, the negative impact on entropy is usually stronger...
- Interpretation: Local style tends to add additional restrictions?

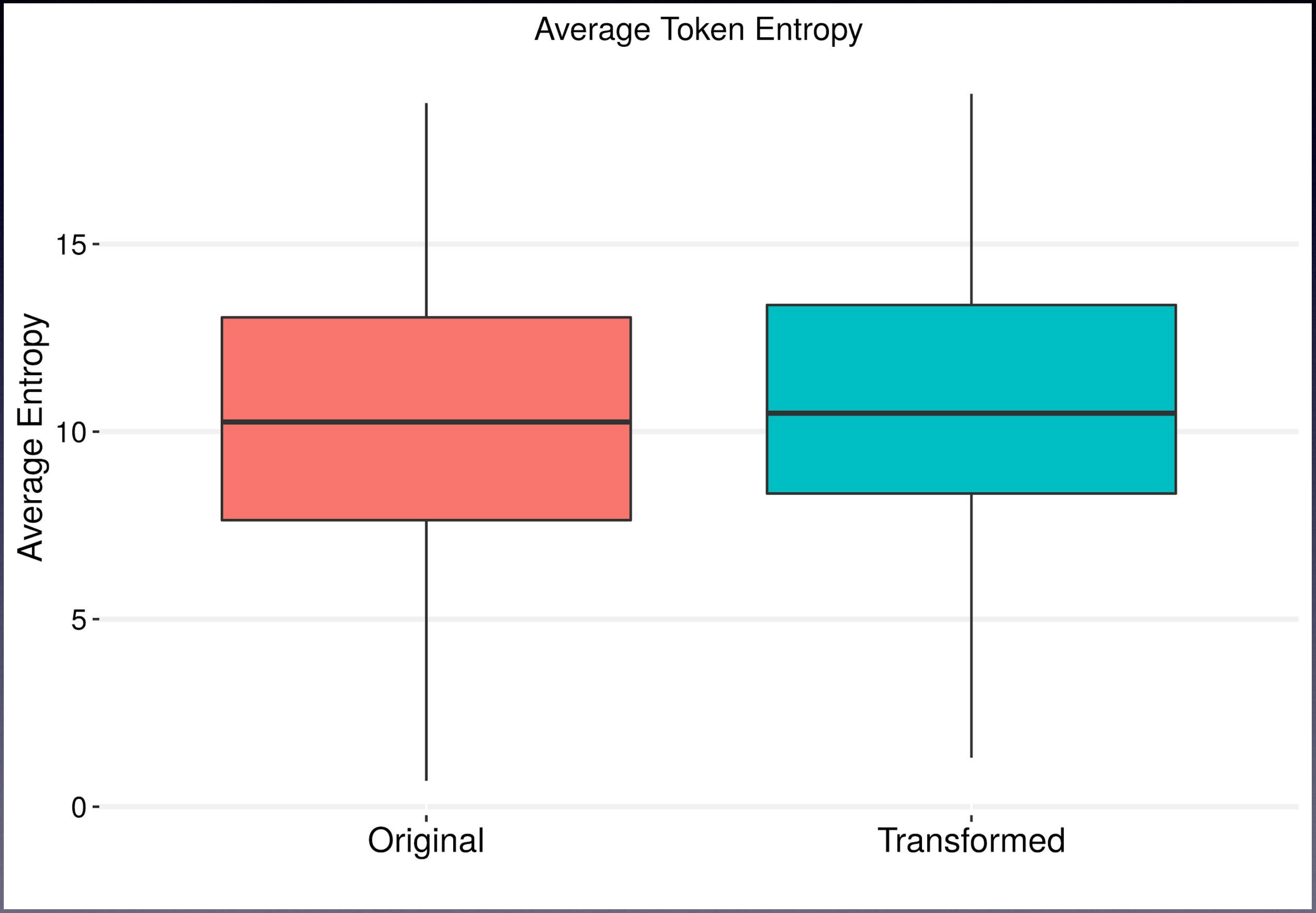
	Global	Cache	Global Type	Cache Type
Operator Swap	0.313	0.859	0.360	0.706
Addition	-0.858	0.142	0.366	0.869
Removal	1.034	0.940	0.087	0.361

Local Style Stronger

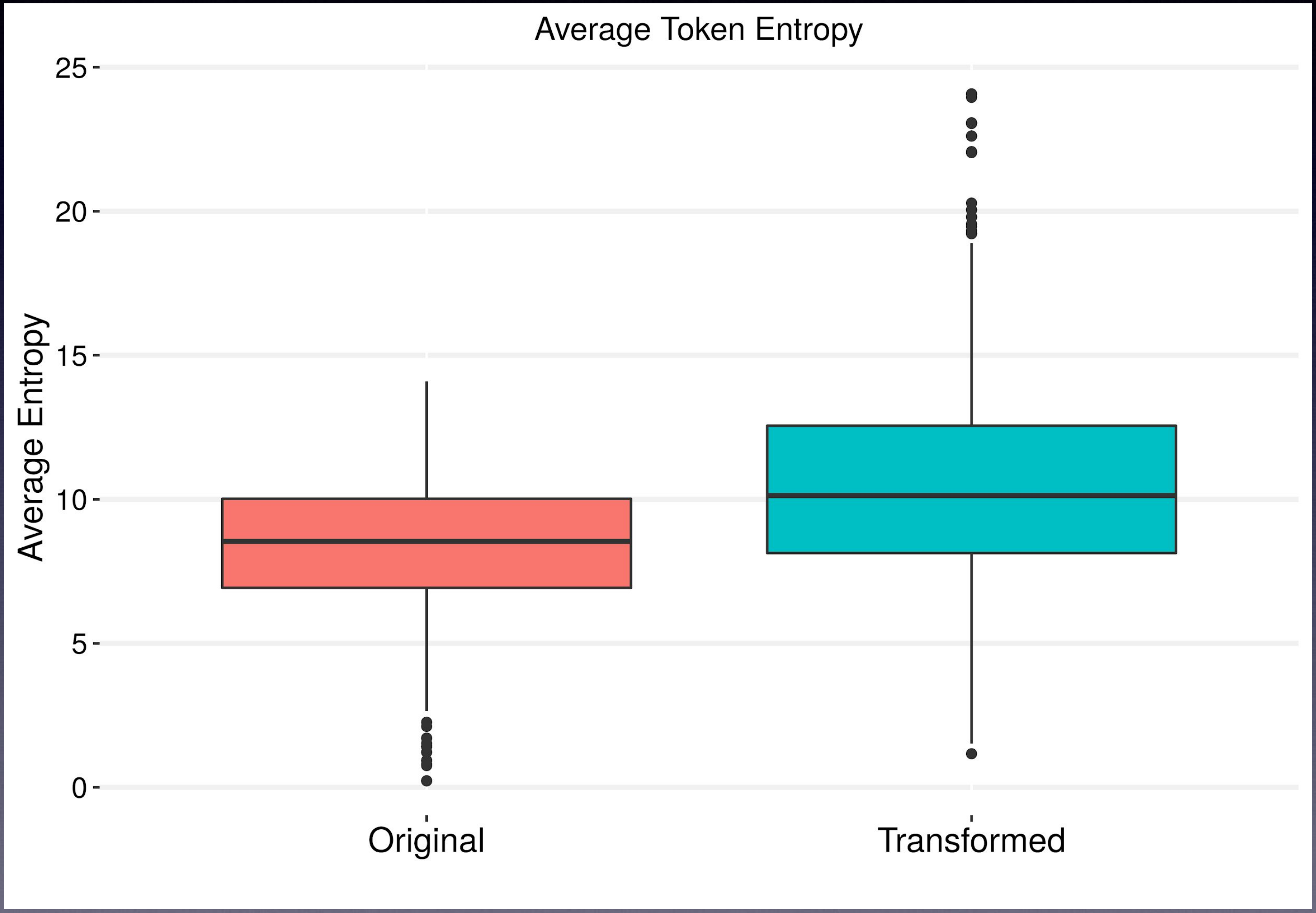
- When employing cache models, the negative impact on entropy is usually stronger...
- => Local style tends to be even more consistent.

	Global	Cache	Global Type	Cache Type
Operator Swap	0.313	0.859	0.360	0.706
Addition	-0.858	0.142	0.366	0.869
Removal	1.034	0.940	0.087	0.361

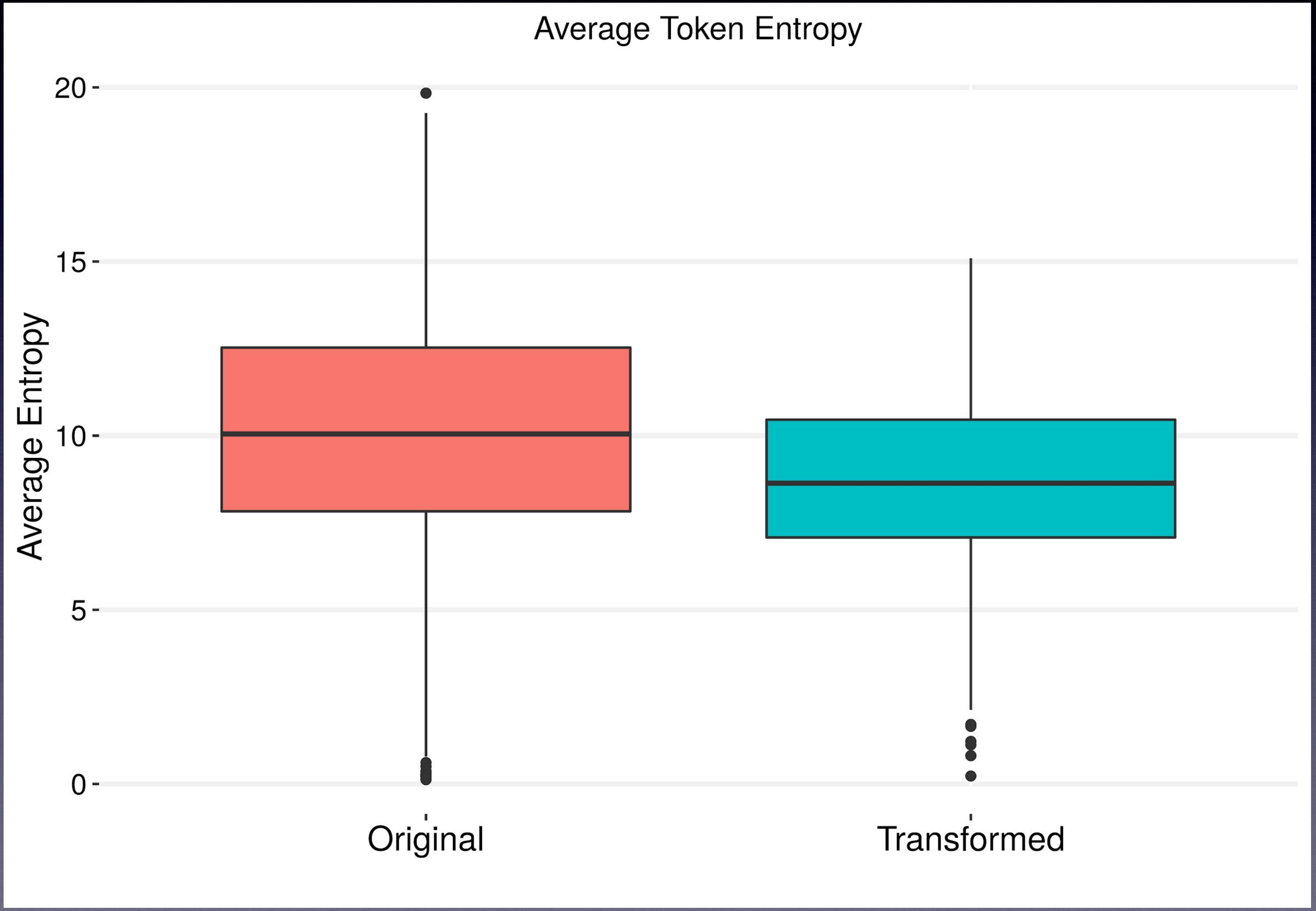
Operand Swapping (Global)



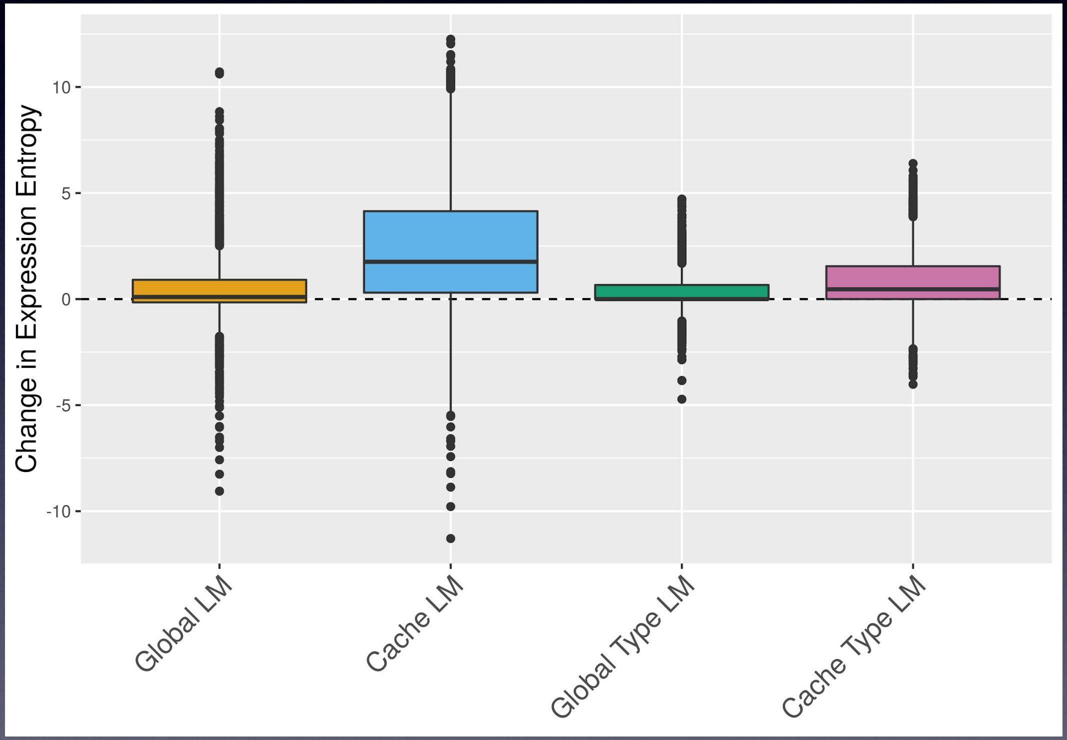
Parentheses Removal (Global)



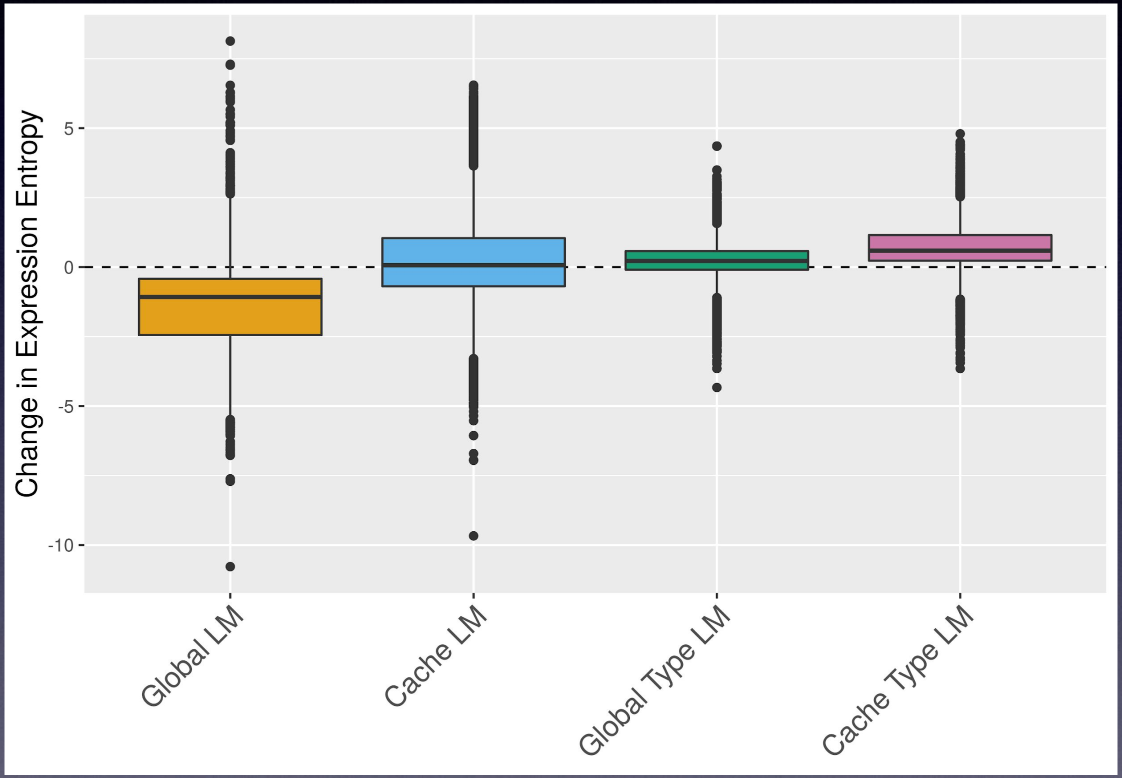
Parentheses Addition (Global)



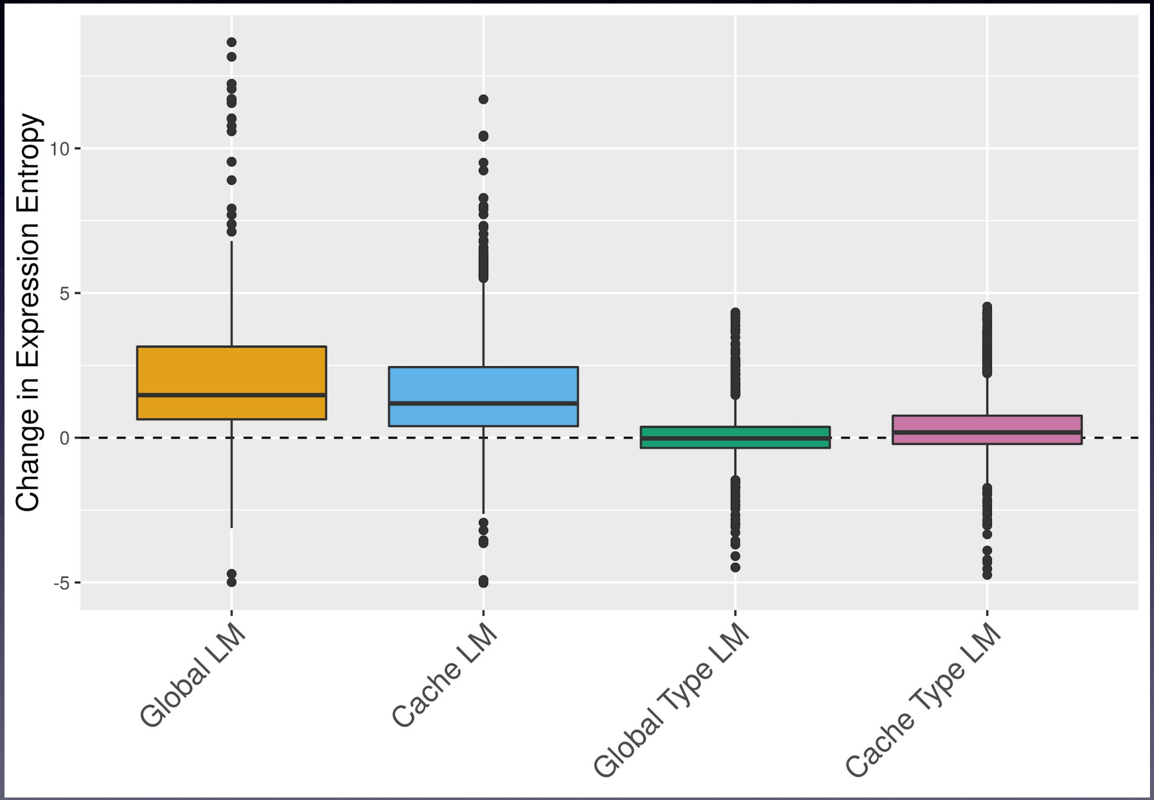
Operator Swapping



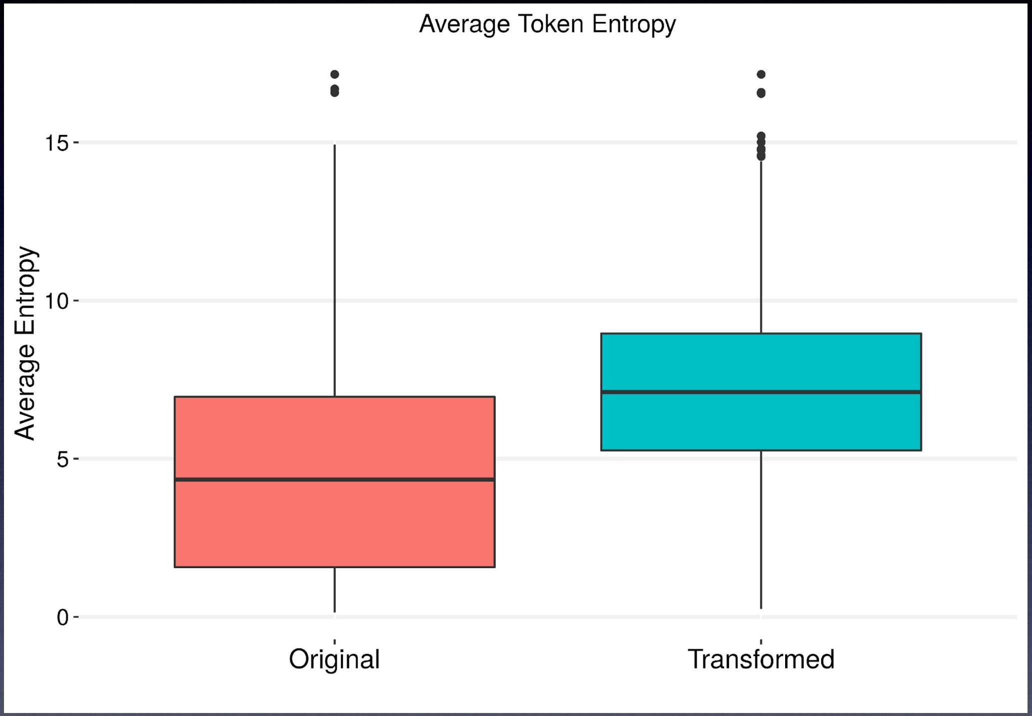
Adding Parenthesis



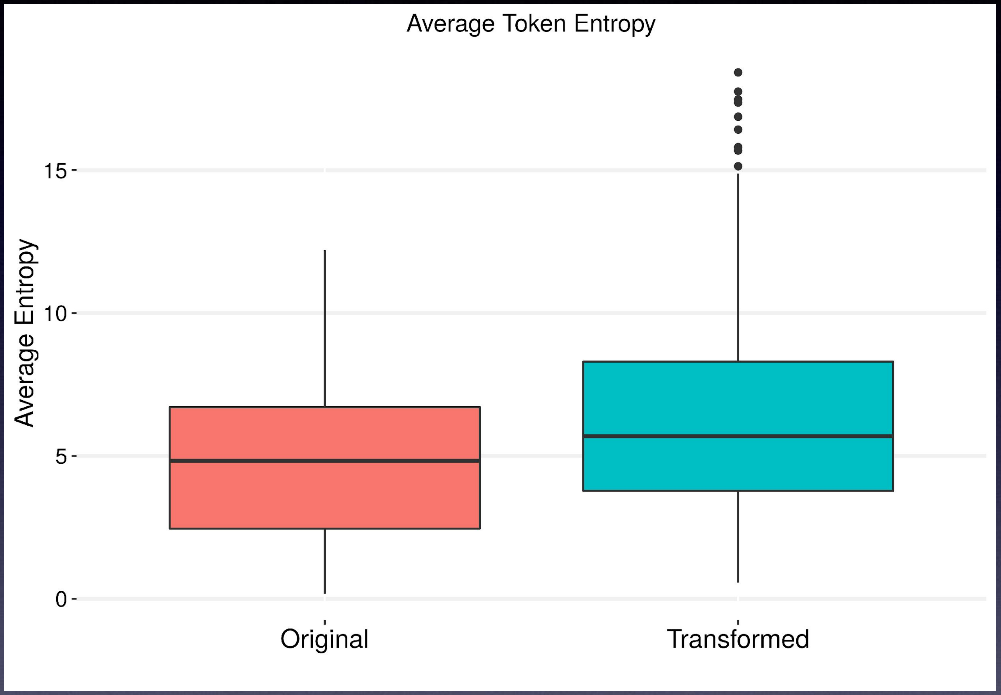
Removing Parenthesis



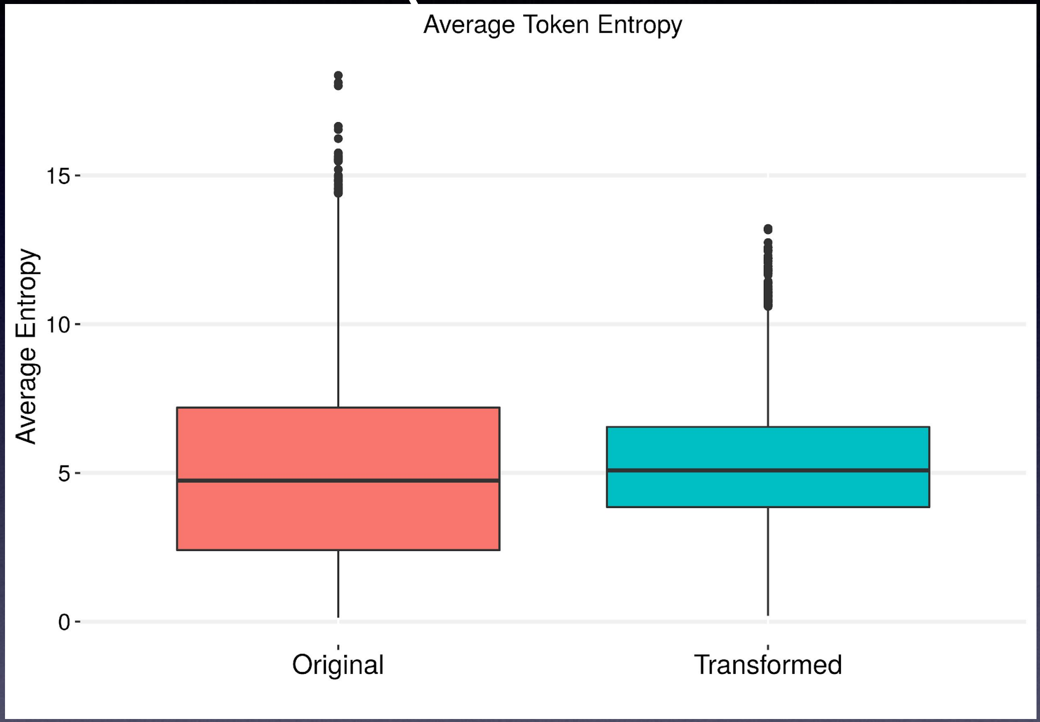
Commutation (Line)



Parentheses Removal (Line)



Parentheses Addition (Line_

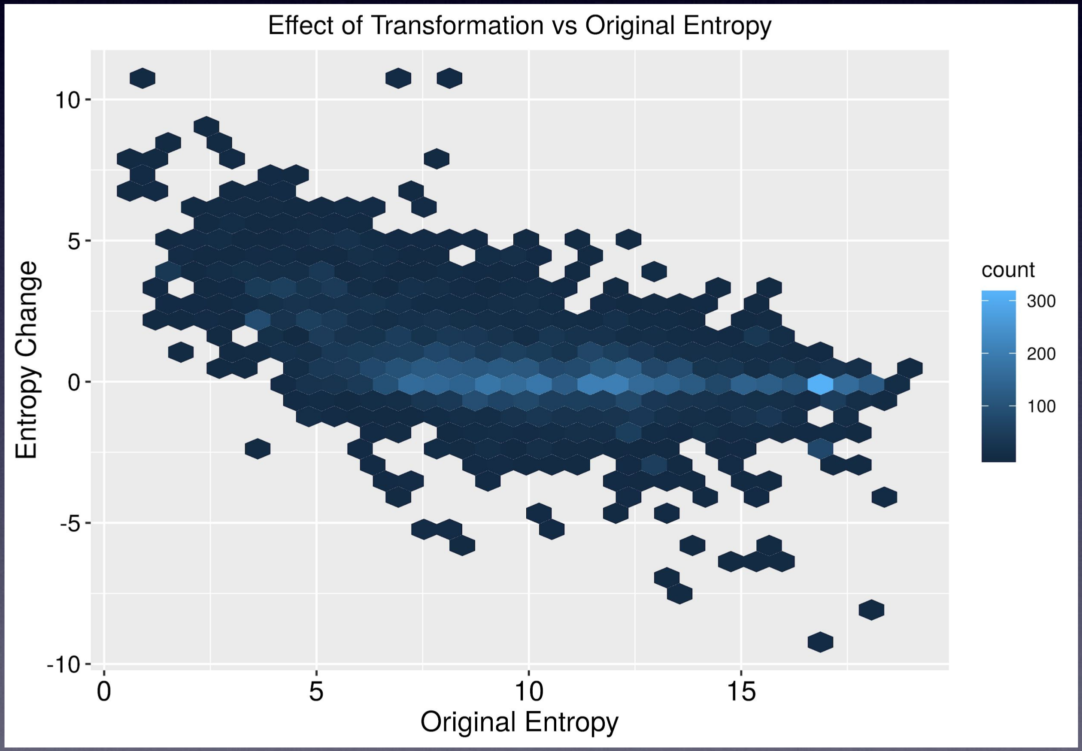


What's up with adding ()?

Code Examples	(Global) Ent Change
result = 31 * result + (int)(temp ^ (temp >>> 32));	-10.78
if (strict && (lowerBound == upperBound)) {	-7.62
Num = (floor*den) + num;	-6.78
final int minIndex = (binIndex * numRows) / numComponents;	-6.59
if (k > (n / 2)) {	-6.58

Effects Vs Original Entropy

Operand Swap (Global Model)



Higher Original Entropy = Greater Potential to reduce entropy

True for most, but not all of the transformations.

<u>Exception</u>: **Global models on () remove transformation**.

- Models capturing preference for including () in complex expressions?

Summary

- Programmers ***do*** generally show preference in choice of computations over the same meaning.
- But, not always true...
  - Opportunity for transforming code?
  - How well can LMs (and entropy measures) correspond with human understanding of code?

Future Directions

- Larger transformations (line swapping)
  - May move to C#/Roslyn
- Human evaluation of entropy score vs understanding/readability metrics
- Variation in human written programs (student code/multiple solutions to Rosetta Code)