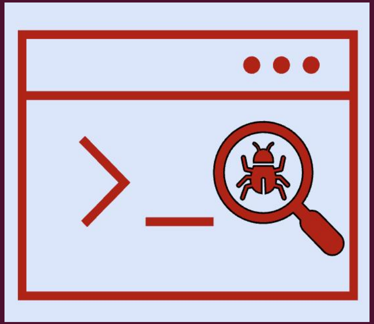
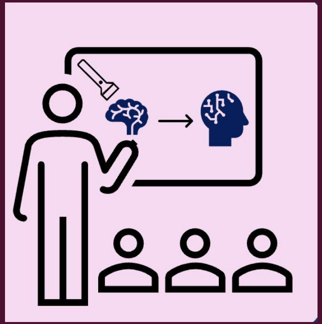
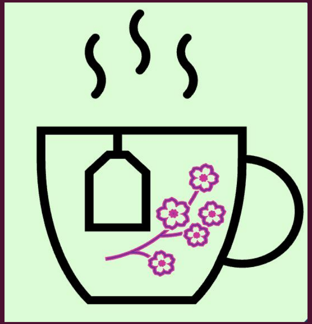




Three Lenses for Improving Programmer Productivity

From Anecdote to Evidence

Madeline Endres, PhD Defense, University of Michigan


MILAN
Birthplace of Thomas A. Edison, February 11, 1847.
A Moravian village, Pequotting. 1804-1809.
First permanent white settlers came in 1816.


2023
THUNDERHEAD

Why study human-focused programming productivity?

The Range of Individual Differences in Programming Performance

Sackman (et al.), 1968

Performance Measure	Slowest Coder	Fastest Coder	Ratio
Code Hours: Algebra Problem	111	7	16:1
Code Hours: Maze Problem	50	2	25:1
Debug Hours: Algebra Problem	170	6	28:1
Debug Hours: Maze Problem	26	1	26:1

Why study human-focused programming productivity?

The Range of Individual Differences in Programming Performance

Sackman (et al.), 1968

Performance Measure	Slowest Coder	Fastest Coder	
Code Hours: Algebra Problem	111	7	
Code Hours: Maze Problem	50	2	
Debug Hours: Algebra Problem	170	6	28:1
Debug Hours: Maze Problem	26	1	26:1

Novice Software Developers, All Over Again

Andrew Begel, Beth Simon

A Tale of Two Cities: Software Developers Working from Home during the COVID-19 Pandemic

DENAE FORD, Microsoft Research

Socioeconomic Status and Computer Science Achievement

Spatial Ability as a Mediating Variable in a Novel Model of Understanding

Miranda C. Parker, Amber Solomon, Brianna Pritchett

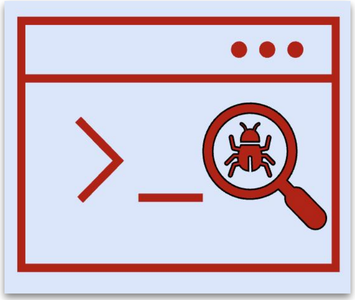
A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges

Jenny T. Liang, Chenyang Yang, Brad A. Myers

What Predicts Software Developers' Productivity?

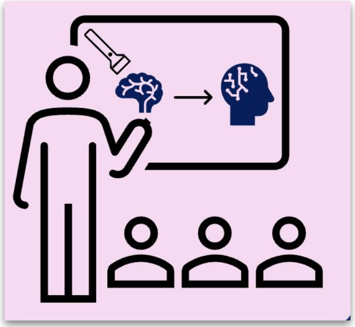
Emerson Murphy-Hill, Ciera Jaspan, Caitlin Sadowski, David Shepherd, Michael Phillips, Collin Winter, Andrea Knight, Edward Smith, and Matthew Jorde

Developing Efficient and Usable Programming Support



Can we support non-traditional novices in **writing more correct code faster?**

Designing Effective Developer Training



Can we use **cognitive insights** to inform training and **improve programming outcomes?**

Understanding External Productivity Factors

How does **psychoactive substance** use **impact software productivity?**

Improving Programming Productivity: My Human-Focused Approach

Desired Research Attribute	**Why I'm Excited** **(and you could be too!)**
Provide *Theoretically-Grounded* and *Actionable Insights*	Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact

Improving Programming Productivity: My Human-Focused Approach

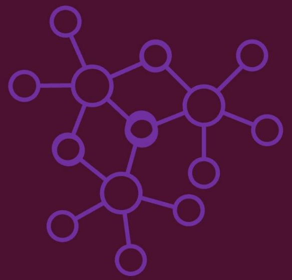
Desired Research Attribute	**Why I'm Excited** **(and you could be too!)**
Provide *Theoretically-Grounded* and *Actionable Insights*	Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact
Include *Empirical or Objective Measures* of Programmers	Captures aspects of programming beyond self-reporting alone, including unconscious behaviors and habits

Improving Programming Productivity: My Human-Focused Approach

Desired Research Attribute	**Why I'm Excited** (and you could be too!)
Provide *Theoretically-Grounded* and *Actionable Insights*	Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact
Include *Empirical or Objective Measures* of Programmers	Captures aspects of programming beyond self-reporting alone, including unconscious behaviors and habits
Minimize Scientific Bias to Support Generalizability	Controlled experimental design can capture a signal, even for complex human behavior

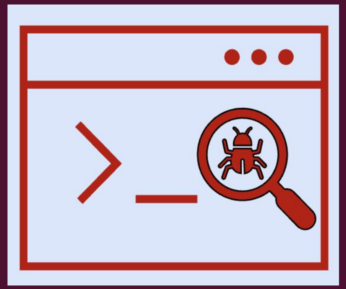
Improving Programming Productivity: My Human-Focused Approach

Desired Research Attribute	**Why I'm Excited** **(and you could be too!)**
Provide *Theoretically-Grounded* and *Actionable Insights*	Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact
Include *Empirical or Objective Measures* of Programmers	Captures aspects of programming beyond self-reporting alone, including unconscious behaviors and habits
Minimize Scientific Bias to Support Generalizability	Controlled experimental design can capture a signal, even for complex human behavior
Support *Diverse Developer Groups*	I prefer approaches that not only help programmers in general, but also help those who need the most support



InFix and Seq2Parse: *Developing Efficient and Usable Tools*

Supporting Non-traditional Programming Novices via a two novel forms of bug-fixing support



The online Python Tutor interpreter currently has 60,000 users per month

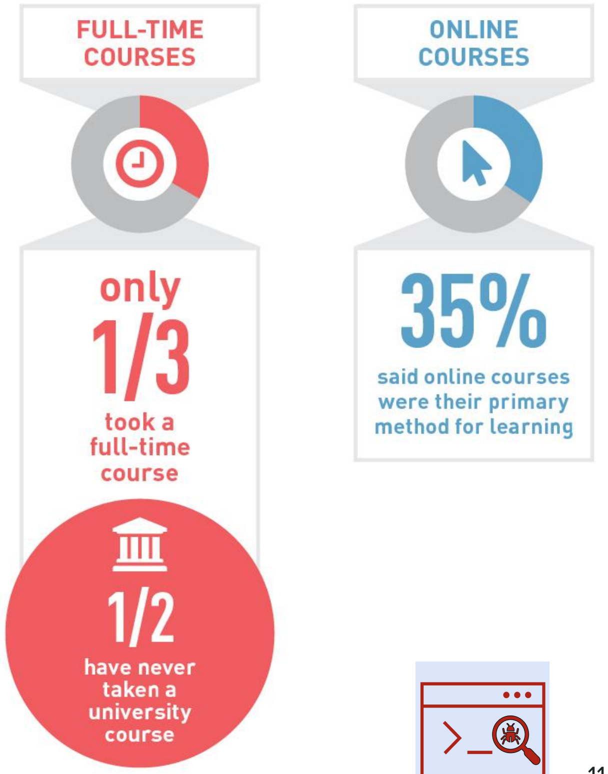
Many People Want to Learn to Code

Without traditional classroom support

GeekWire NEWS JOBS EVENTS RESOURCES ABOUT Search

Coding bootcamps see huge enrollment increase

How do Codecademy's
45 million users
learn to code?



One Such Platform: **Python Tutor**

Write code in Python 3.11 [newest version, latest features

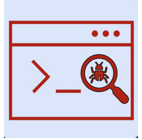
```python
def listSum(numbers):
  if not numbers:
    return 0
  else:
    (f, rest) = numbers
    return f + listSum(rest)

myList = (1, (2, (3, None)))
total = listSum(myList)
```

Visualize Execution   Get AI Help

Python Tutor is a free online **interpreter**. It helps novices **visualize arbitrary code execution.**

Users are primarily *Novice Programmers*

Started in 2010, it has had over **150 million users from 180 countries**



Parse Errors

- Syntax errors are, by far, the most common Python error type experienced by novice programmers (77%)

```
u = 42
x = 3.14
print(x * math.e / 2
```

```
SyntaxError: missing
parentheses in call to print
```

Input-Related Bugs

- We found that 6% of student errors are resolved by fixing the program input, not the source code

Example Code and Input

```
u = 42
x = float(input())
print(x * math.e / 2)
```

```
26,2
```

```
ValueError: could not convert
string to float: '26,2'
```

Parse Errors

- Syntax errors are, by far, the most common Python error type experienced by novice programmers (77%)

Proposed Approach:
Neurosymbolic technique,
Seq2Parse

Published in OOPSLA, 2022

Input-Related Bugs

- We found that 6% of student errors are resolved by fixing the program input, not the source code

Example Code and Input

Proposed Approach:
Template-repair approach,
InFix

Published in ASE, 2019

Parse Errors

- Syntax errors are, by far, the most common Python error type experienced by novice programmers (77%)

Proposed Approach: Neurosymbolic technique, **Seq2Parse**

Published in OOPSLA, 2022

Input-Related Bugs

- We found that 6% of student errors are resolved by fixing the program input, not the source code

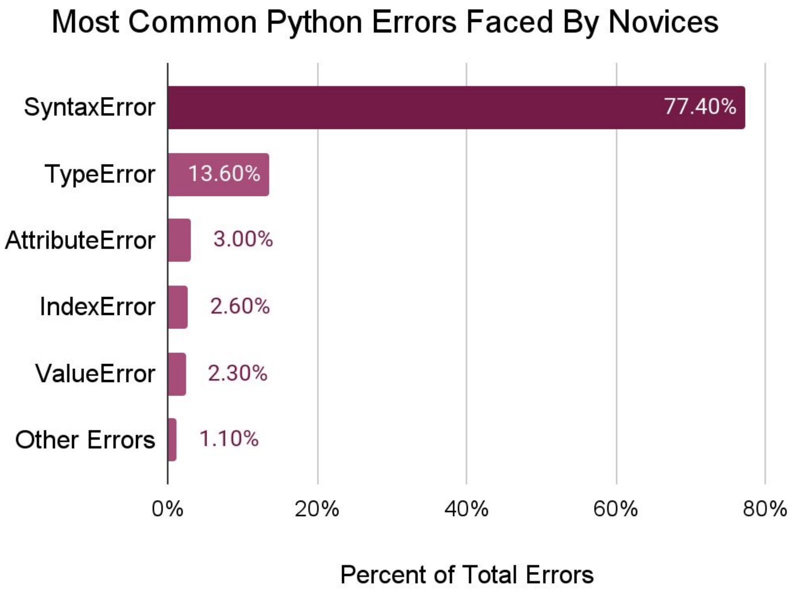
Example Code and Input

Proposed Approach: Template-repair approach, **InFix**

Published in ASE, 2019

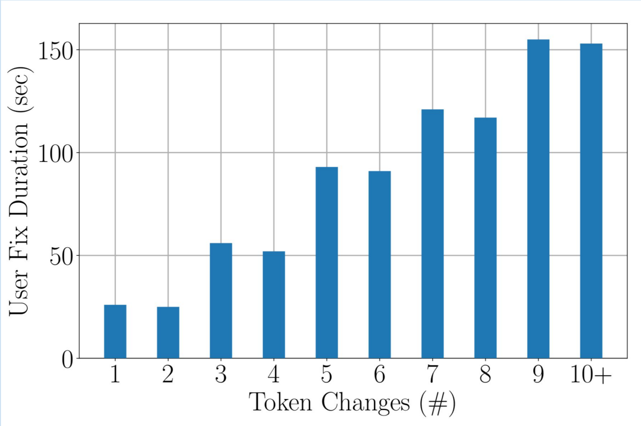
What do Non-Traditional Novices Struggle with? ***Parse Errors***

For Non-Traditional Novices, Parse Errors (Syntax Errors) are both **common** and **challenging**



37% of Parse Errors take over two minutes to resolve

More complex fixes take even longer:



Fixing Parse Errors: How can we support Novices?

Goal: We want support for fixing parse errors faced by non-traditional novices that is both:

- *Effective*: can provide **helpful repairs close to the user's intent** in the majority of cases

and

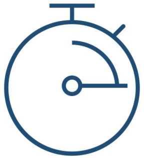
- *Efficient*: Fast enough to be computed in ***real time***

Fixing Parse Errors: How can we support Novices?

Goal: We want support for fixing parse errors faced by non-traditional novices that is both:

- *Effective*: can provide **helpful repairs close to the user's intent** in the majority of cases

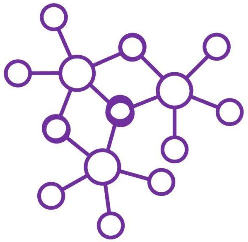
and



- *Efficient*: Fast enough to be computed in ***real time***

Symbolic Approach?

Neural Approach?



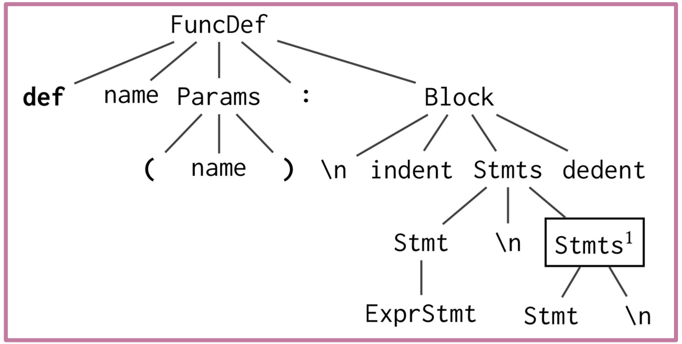
Parsing Overview

Program P

```
def foo(a):
  return a + 42

def bar(a):
  b = foo(a) + 17
  return b +
```

Grammar G

```
S        →  Stmts end_marker
Stmts    →  Stmt \n | Stmt \n Stmts
Stmt     →  FuncDef | ExprStmt
          | RetStmt | PassStmt | ...
FuncDef  →  def name Params : Block
Block    →  \n indent Stmts dedent
```



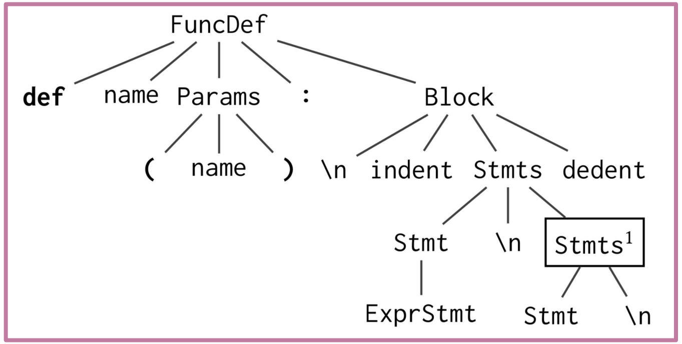
Finding Parse Errors: Fault Localization

Program P

```
def foo(a):
  return a + 42

def bar(a):
  b = foo(a) + 17
  return b +
```

Grammar G

```
S        → Stmts end_marker
Stmts    → Stmt \n | Stmt \n Stmts
Stmt     → FuncDef | ExprStmt
           | RetStmt | PassStmt | ...
FuncDef  → def name Params : Block
Block    → \n indent Stmts dedent
```



Fixing Parse Errors: Error Correcting Earley Parsers

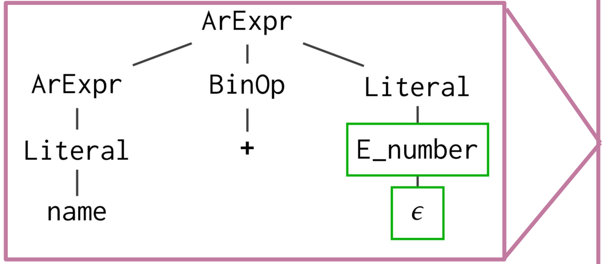
Program P

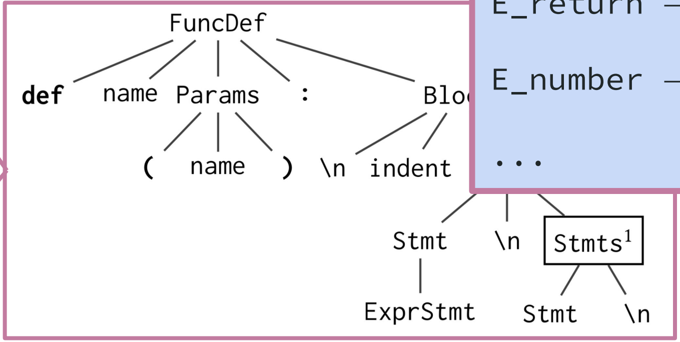
```
def foo(a):
  return a + 42

def bar(a):
  b = foo(a) + 17
  return b +
```

Grammar G'

```
S        → Stmts end_marker
Stmts    → Stmt \n | Stmt \n Stmts
Stmt     → FuncDef | ExprStmt
           | RetStmt | PassStmt | ...
FuncDef  → def name Params : Block
Block    → \n indent Stmts dedent
```

```
New_S    →  S | S Insert
RetStmt  → | E_return | E_return Args
E_return →  return | ϵ | Replace
           | Insert return
E_number →  number | ϵ | Replace
           | Insert number
...
```





Fixing Parse Errors: Error Correcting Earley Parsers

Program P

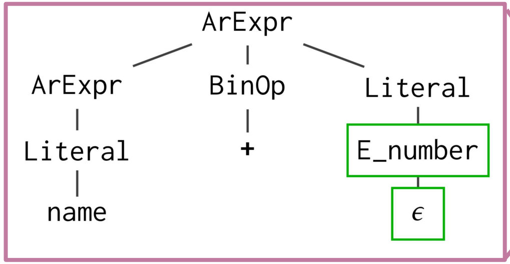
```python
def foo(a):
  return a + 42

def bar(a):
```

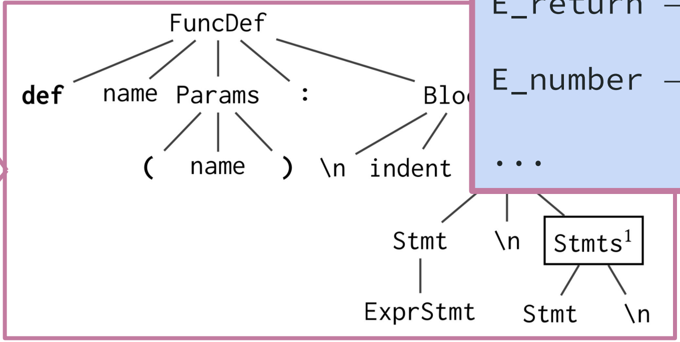
Too many rules!

Grammar G'

```
S        → Stmts end_marker
Stmts    → Stmt \n | Stmt \n Stmts
Stmt     → FuncDef | ExprStmt
           | RetStmt | PassStmt | ...
FuncDef  → def name Params : Block
Block    → \n indent Stmts dedent
```

```
New_S    →  S | S Insert
RetStmt  → |E_return | E_return Args
E_return →  return | ϵ | Replace
           | Insert return
E_number →  number | ϵ | Replace
           | Insert number
...
```





Fixing Parse Errors: Neural Approaches

Pros:

- Sequence classifiers can be **good at predicting edits** or repairs **similar to human behavior**
- Once trained, neural approaches **can be efficient**

Cons:

- Generally, **no guarantees** that the response will correct (e.g., actually parse), let alone be a minimal repair
- Neural approaches can be **confused by program context** not directly related to the parse error

```
def foo(a):
  return a + 42

def bar(a):
  b = foo(a) + 17
  return b +
```

SEQ2PARSE: Key Insight

- EC-Parsers guarantee a correct fix, but are slow because **they consider too many production rules**, *the vast majority of which are not needed to fix any given error.*
- In contrast, Neural approaches are fast and leverage user patterns, but **can be inaccurate or untrustworthy** if used alone

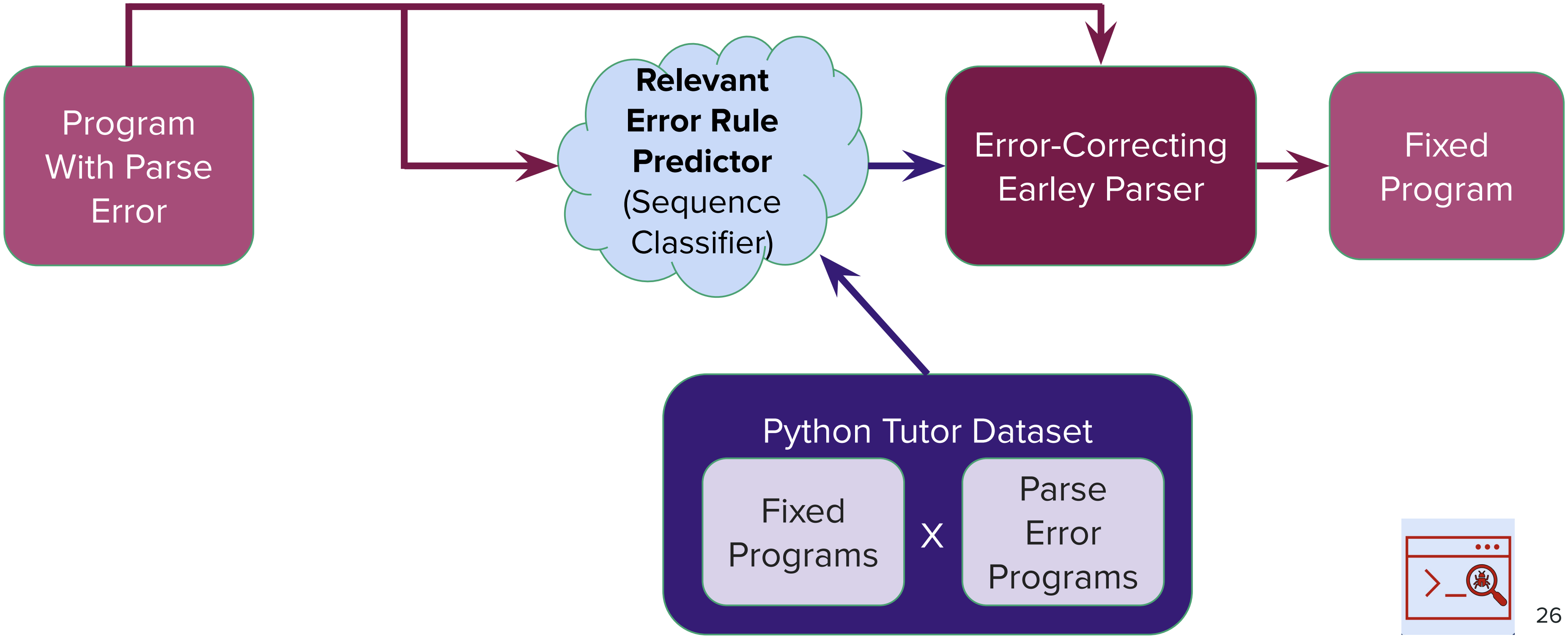
We propose to get the best of both worlds and *efficiently* and *accurately* suggest repairs in a **neurosymbolic fashion**:

1. Train sequence classifiers to **predict the relevant EC-rules for a given program, *instead of the next token or the full fix***

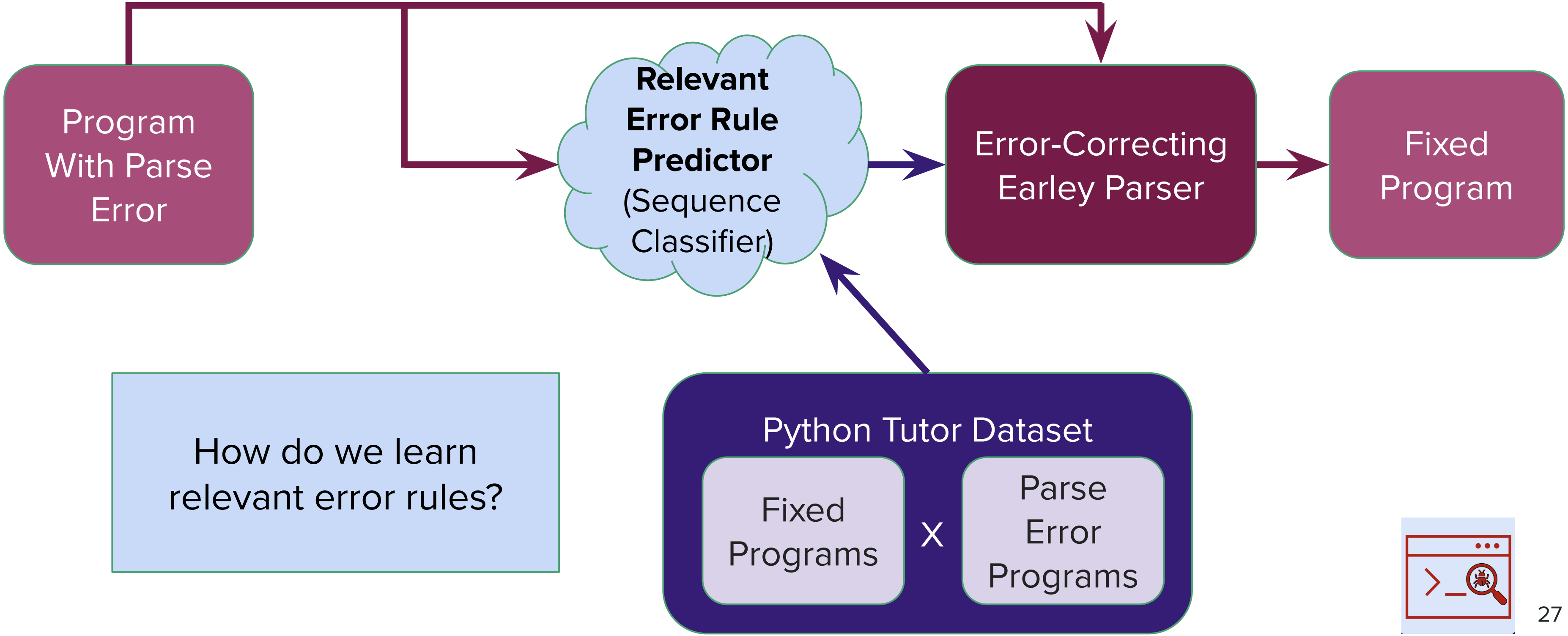
2. Use the predicted rules to synthesize a Parse Error repair via EC-Parsing

SEQ2PARSE: Efficient Fixes for Novice Parse Errors

SEQ2PARSE: Efficient Fixes for Novice Parse Errors



SEQ2PARSE: Efficient Fixes for Novice Parse Errors



How do we learn relevant error rules?

Additional Considerations for Learning EC-Production Rules

Ill-parsed Program Representation for Learning:

```python
def foo(a):
  return a + 42

def bar(a):
  b = foo(a) + 17
  return b +
```

```
Stmt \n

def name Params: \n
indent Stmt \n
return Expr BinOp \n
dedent end_marker
```

- *Problem*: Predicting relevant production rules using full buggy programs causes the model to be **confused by irrelevant program context**
- *Our Solution*: Instead of standard token strings, develop semantics for **Abstracted Token Sequences** that concentrate information relevant to a given parse error and remove confusing context

Mitigating Representational Ambiguity:

- *Problem:* While needed, this abstraction adds ambiguity into what parse tree should result from any given abstracted token sequence
- *Our Solution:* Use fixed Python Tutor programs to learn a **Probabilistic Context Free Grammar** and resolve parsing ambiguities

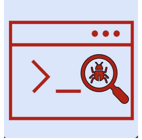
S	$\rightarrow$	Stmts end_marker ($p = \underline{100.0\%}$)
Stmts	$\rightarrow$	Stmt \n ($p = \underline{38.77\%}$) | Stmt \n Stmts ($p = \underline{61.23\%}$)
Stmt	$\rightarrow$	ExprStmt ($p = \underline{62.64\%}$) | RetStmt ($p = \underline{7.59\%}$) | ...
RetStmt	$\rightarrow$	**return** ($p = \underline{1.61\%}$) | **return** Args ($p = \underline{98.39\%}$)

Seq2Parse: Python Implementation

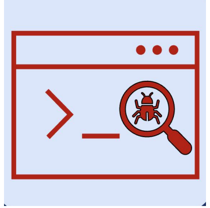
- Dataset: Over **One Million Buggy/Fixed Program Pairs** from Python Tutor
    - Average abstracted token sequence is 43 tokens long
    - 15,000 random programs used for evaluation, the rest for model training

- Error Rule Prediction **Model Structure**:
    - **Transformer classifier** with six blocks, each with a fully-connected hidden layer of 256 neurons and 12 attention heads, **connected to a DNN-based classifier** with two fully-connected hidden layers.
    - Trained using an Adam optimizer, a variant of stochastic gradient descent for 50 epochs.

- **Model Output: Top 20 most likely error production rules** for a given Buggy Program
    - These rules are then fed into the Error Correcting Earley Parser



Seq2Parse: Does it work? Yes!



Seq2Parse can fix most parse errors for non-traditional novices,

Repair Rate: Seq2Parse **can parse and repair up to 94.25% of programs** with syntax errors.

in real time

Efficiency: Seq2Parse can parse and repair the vast majority of the test set in under 20 seconds in a **median time of 2.1 seconds**

and with the same, or better, quality to the novices themselves!

Quality: Seq2Parse generates the exact fix as the historical user up to 35% of the time! Of the remainder, Seq2Parse repairs are equivalent to or more useful than historical repairs 52% and 15% of the time, respectively.

Seq2Parse: Does it work? Yes!

We assess repair quality via a study with 39 programmers

Captured 527 subjective quality ratings for a corpus of 50 SEQ2PARSE / historical fix pairs
Compared the two pairs using standard statistical tests

Buggy Python Program

```
shift = [(9, 2, 4), (7, 6, 9)]
for i in shift:
    for item in i:
    print(i[1])
```

Debugging Hint: Possible Fix

```
shift = [(9, 2, 4), (7, 6, 9)]
for i in shift:
    for item in i:
        print(i[1])
```

Python Error Message

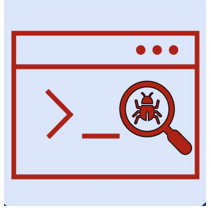
```
  File "program.py", line 4
    print(i[1])
    ^
IndentationError: expected an indented block after 'for'
statement on line 3
```

Questions To Answer:

1. Between 1 (not helpful) and 5 (very helpful), how **helpful** is the **Python Error Message** for debugging the program?

2. Between 1 (not helpful) and 5 (very helpful), how **helpful** is the provided **Possible Fix** for debugging the program?

Seq2Parse: Does it work? Yes!



SEQ2PARSE can fix most parse errors for non-traditional novices,

Repair Rate: SEQ2PARSE **can parse and repair up to 94.25% of programs** with syntax errors.

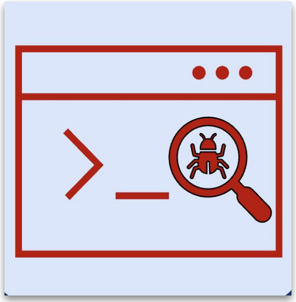
in real time

Efficiency: SEQ2PARSE can parse and repair the vast majority of the test set in under 20 seconds in a **median time of 2.1 seconds**

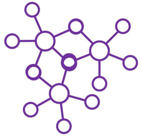
and with the same, or better, quality to the novices themselves!

Quality: SEQ2PARSE generates the exact fix as the historical user up to 35% of the time! Of the remainder, SEQ2PARSE repairs are equivalent to or more useful than historical repairs 52% and 15% of the time, respectively.

Lens 1 — Summary: Developing Better Bug Fixing Support



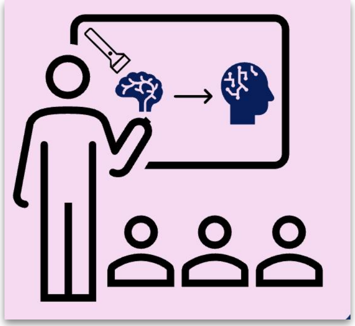
- We **identified parse errors and input-related bugs as a significant barrier** for non-traditional novices in practice
- We **propose Seq2Parse**, a neurosymbolic approach to fixing parse errors, and **InFix**, a template-based approach for fixing input-related bugs
- Our preliminary results show that both tools produce repairs that are **accurate, efficient, and of high quality,** as judged by humans.



Developing Efficient and Usable Programming Support

Can we support non-traditional novices in **writing more correct code faster?**

Designing Effective Developer Training



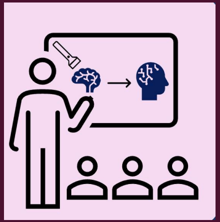
Can we use **cognitive insights** to inform training and **improve programming outcomes?**

Understanding External Productivity Factors

How does **psychoactive substance** use **impact software productivity?**

TO READ OR TO ROTATE?

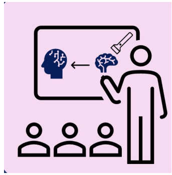
An example of how cognitive insights can inform effective programming interventions



ESEC/FSE, 2021,
ICSE 2021

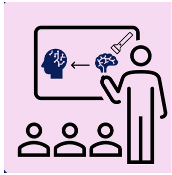
Novice programmers often struggle,
especially those students with weaker preparatory education

This struggle may result from **insufficient preparation in cognitive skills** necessary for programming



How can we help students?

Cognitive interventions (the supplemental training of a necessary cognitive skill) can **help underprepared students succeed in many fields**



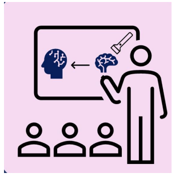
How can we help students?

Cognitive interventions (the supplemental training of a necessary cognitive skill)

can **help underprepared students succeed in many fields**

A writing-intensive course improves biology undergraduates' perception and confidence of their abilities to read scientific literature and communicate science

Sara E. Brownell,[1] Jordan V. Price,[2] and Lawrence Steinman[2,3]



How can we help students?

Cognitive interventions (the supplemental training of a necessary cognitive skill) can **help underprepared students succeed in many fields**

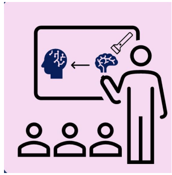
A writing-intensive course improves biology undergraduates' perception and confidence of their abilities to read scientific literature and communicate science

Sara E. Brownell,[1] Jordan V. Price,[2] and Lawre

THE EFFECTS OF ORIGAMI LESSONS ON STUDENTS' SPATIAL VISUALIZATION SKILLS AND ACHIEVEMENT LEVELS IN A SEVENTH-GRADE MATHEMATICS CLASSROOM

A Qualitative Inquiry into the Effects of Visualization on High School Chemistry Students' Learning Process of Molecular Structure

Susan Deratzou



How can we help students?

Cognitive interventions (the supplemental training of a necessary cognitive skill)

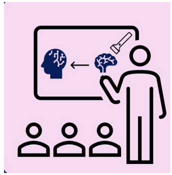
can **help underprepared students succeed in many fields**

A writing-intensive course improves biology undergraduates' perception and confidence of their abilities to read scientific literature and communicate science

Sara E. Brownell,[1] Jordan V. Price,[2] and Lawre

THE EFFECTS OF ORIGAMI LESSONS ON STUDENTS' SPATIAL VISUALIZATION SKILLS AND ACHIEVEMENT LEVELS IN A SEVENTH-GRADE MATHEMATICS CLASSROOM

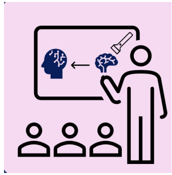
A Qualitative Inquiry into the Effects of Visualization on High School Chemistry Students' Learning



Does spatial skills instruction improve STEM outcomes? The answer is 'yes'

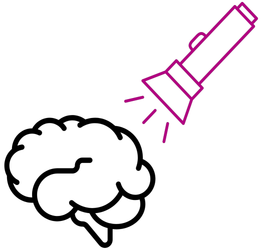
Sheryl Sorby[a,*], Norma Veurink[b], Scott Streiner[c]

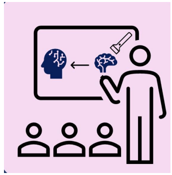
Cognitive interventions may also help improve programming ability for novices…



Cognitive interventions may also help improve programming ability for novices…

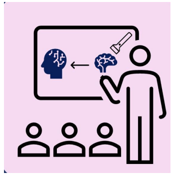
… but what cognitive skills should we target?



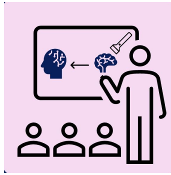


Neuroimaging and Software Engineering

- Understanding the **cognitive basis of software engineering** is important
- Neuroimaging allows us to **objectively measure** this cognitive basis by **directly observing brain activation** patterns while programming! (as opposed to self-reported data)
- Potential impact areas of neuroimaging include pedagogy, technology transfer, expertise, adult retraining



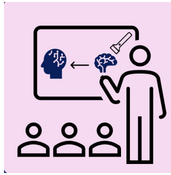
What do we know up to 2023?



- Neuroimaging uses **contrast-based experiments** to compare **programming** activities to **other cognitive tasks**

Neuroimaging Experiment	**Is programming like Reading?**	**Is programming like Spatial Reasoning?**
Siegmund *et al.*, (2014)	✔	
Siegmund *et al.*, (2017)	✔	
Floyd *et al.*, (2017)	✔	
Huang *et al.*, (2019)		✔

What do we know up to 2023?

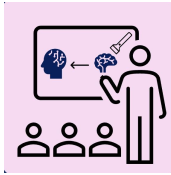


- Neuroimaging uses **contrast-based experiments** to compare **programming** activities to **other cognitive tasks**

Neuroimaging Experiment	**Is programming like Reading?**	**Is programming like Spatial Reasoning?**
Siegmund *et al.*, (2014)	✔	
Siegmund *et al.*, (2017)	✔	
Floyd *et al.*, (2017)	✔	
Huang *et al.*, (2019)		✔

Found connection with expertise

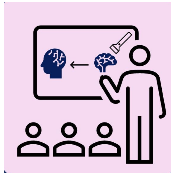
What do we know up to 2023?



- Neuroimaging uses **contrast-based experiments** to compare **programming** activities to **other cognitive tasks**

Neuroimaging Experiment	**Is programming like Reading?**	**Is programming like Spatial Reasoning?**	**What about with novices?**
Siegmund *et al.*, (2014)	✔		**?**
Siegmund *et al.*, (2017)	✔		**?**
Floyd *et al.*, (2017)	✔		**?**
Huang *et al.*, (2019)		✔	**?**

Lens 2 Study Overview



Phase 1: **Neuroimaging**

- We first build a model of novice programmer cognition using **the first neuroimaging study of true novice programmers** during code comprehension

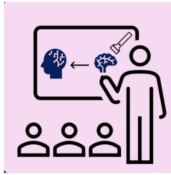
Published in ICSE, 2021

Phase 2: **Transfer Training**

- We then investigate the the usefulness of transfer training in computing **comparing the impact of two cognitive interventions on novice programming performance** in a controlled, longitudinal study

Published in FSE, 2021

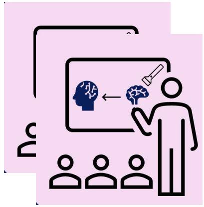
Phase 1: Neuroimaging Method



- We use **Functional Near Infrared Spectroscopy** (fNIRS) to capture the brain activation patterns of **novice programmers** (no prior programming experience)
  - **fNIRS** uses light to measure the oxygen levels in different parts of the brain
  - Supports studying the brain while doing natural programming tasks

- We compare programming-associated activations to **two well-understood cognitive tasks** commonly used in neuroimaging studies of expert developers: **spatial visualization** and **reading**

Experimental Timeline: A Semester of CS1



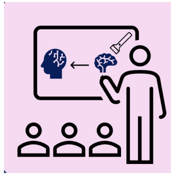
Week 1: Start of the CS1 (EECS 183) semester

Week 3: Participant recruitment from CS1

Week 4-5.5: Brain scans

Week 16: End of semester

Experimental Timeline: A Semester of CS1



Week 1: Start of the CS1 (EECS 183) semester

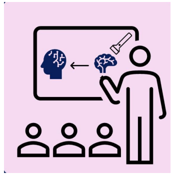
Week 3: Participant recruitment from CS1

Week 4-5.5: Brain scans

Week 16: End of semester

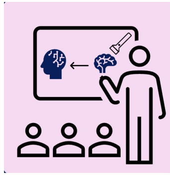
Neuroimaging Stimuli

We compare brain activation during three tasks:



Neuroimaging Stimuli

We compare brain activation during three tasks:



- **CS1-Level Programming**

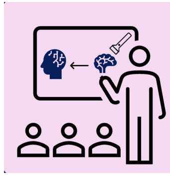
Please type the corresponding letter which best represents the **return value** of the **function call** below:

```
bool func(bool x, bool y) {
    return (x && y) || (x && !y);
}

func(true, false)
```

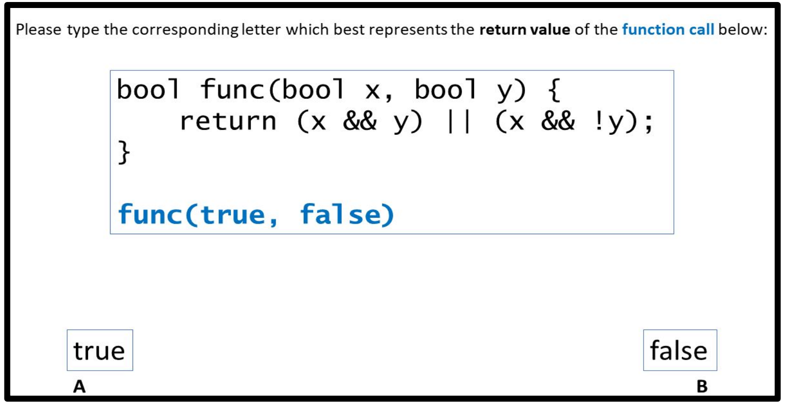
true	false
A	B

Neuroimaging Stimuli

We compare brain activation during three tasks:

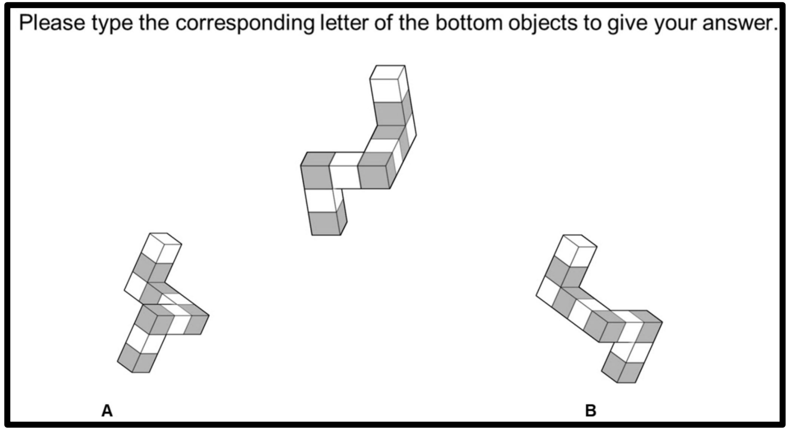


- CS1-Level Programming
- **Mental Rotation**



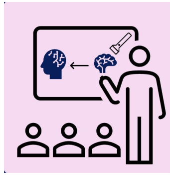
Please type the corresponding letter which best represents the **return value** of the **function call** below:

```
bool func(bool x, bool y) {
    return (x && y) || (x && !y);
}

func(true, false)
```

Neuroimaging Stimuli

We compare brain activation during three tasks:



- CS1-Level Programming
- Mental Rotation
- **Prose Fill in the Blank**

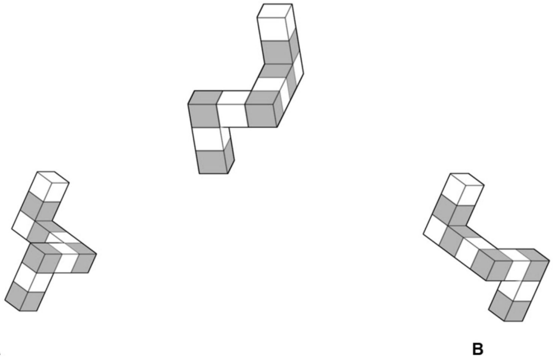
Please type the corresponding letter which best represents the **return value** of the **function call** below:

```
bool func(bool x, bool y) {
    return (x && y) || (x && !y);
}

func(true, false)
```

true — A

false — B

Please type the corresponding letter of the bottom objects to give your answer.



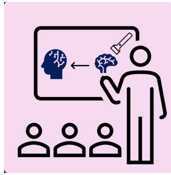
Please type the corresponding letter of the word which best fills in **BLANK** in the sentence below:

The author presents the life of Zane Grey with **BLANK** unusual in a biographer: he is not even convinced that Grey was a good writer.

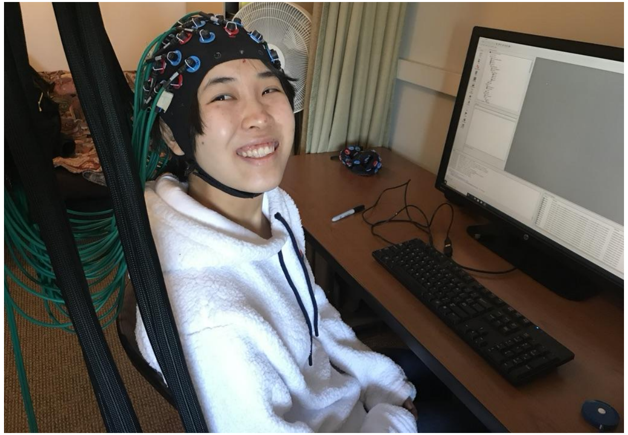
an eloquence — A

a detachment — B

fNIRS Scan Data Collection and Analysis

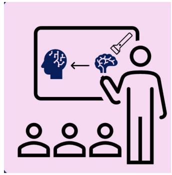


- Each scan session lasts **two hours** in a darkened room
  - 90 stimuli, 30 of each type (programming, mental rotation, reading)

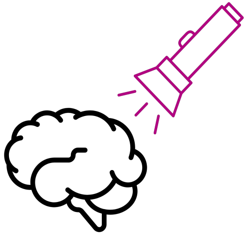


- 36 participants, **31 valid** (24 female, 7 male)
  - Recruited from EECS 183, here at Michigan!
- Data Analysis
  - Compare activation by task by brain area using best practices from psychology
  - Significance threshold: $p < 0.01$.
  - We used False Discovery Rate to correct for multiple comparisons: $q < 0.05$

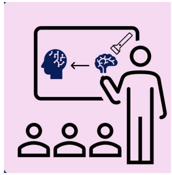
A Mathematical Model of Novice Cognition: Primary Research Questions



- *Comparative Activation*: How does novice program comprehension **compare to prose comprehension and spatial reasoning** at the cognitive level?
    - How do novices' brain activation patterns **compare to those of expert developers**?



Phase 1 Results: Comparative Brain Activation



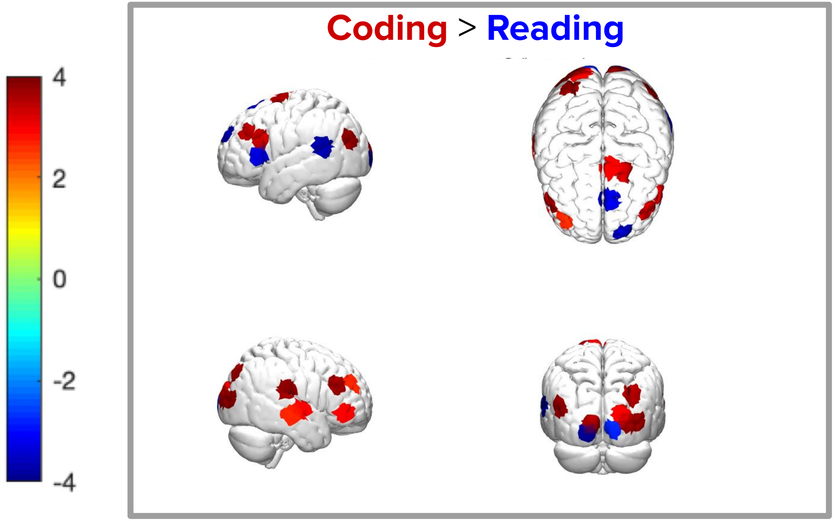
- Question: **Do novices use spatial and/or language areas while programming?**

- Result: While areas associated with both are activated, we find **more substantial differences between Coding and Reading** than between **Coding and Mental Rotation**

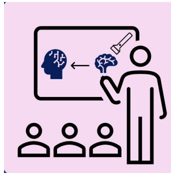
Phase 1 Results: Comparative Brain Activation

- Question: **Do novices use spatial and/or language areas while programming?**

- Result: While areas associated with both are activated, we find **more substantial differences between Coding and Reading** than between **Coding and Mental Rotation**

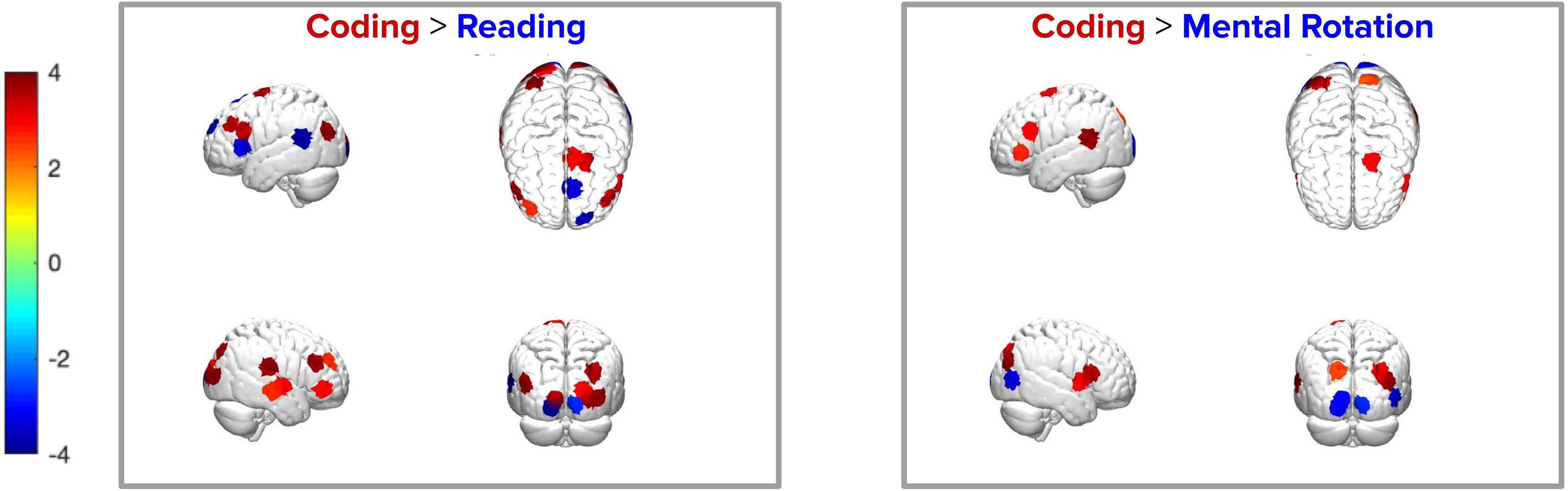


Phase 1 Results: Comparative Brain Activation

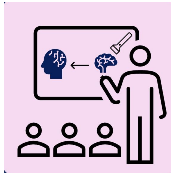


- Question: **Do novices use spatial and/or language areas while programming?**

- Result: While areas associated with both are activated, we find **more substantial differences between Coding and Reading** than between **Coding and Mental Rotation**

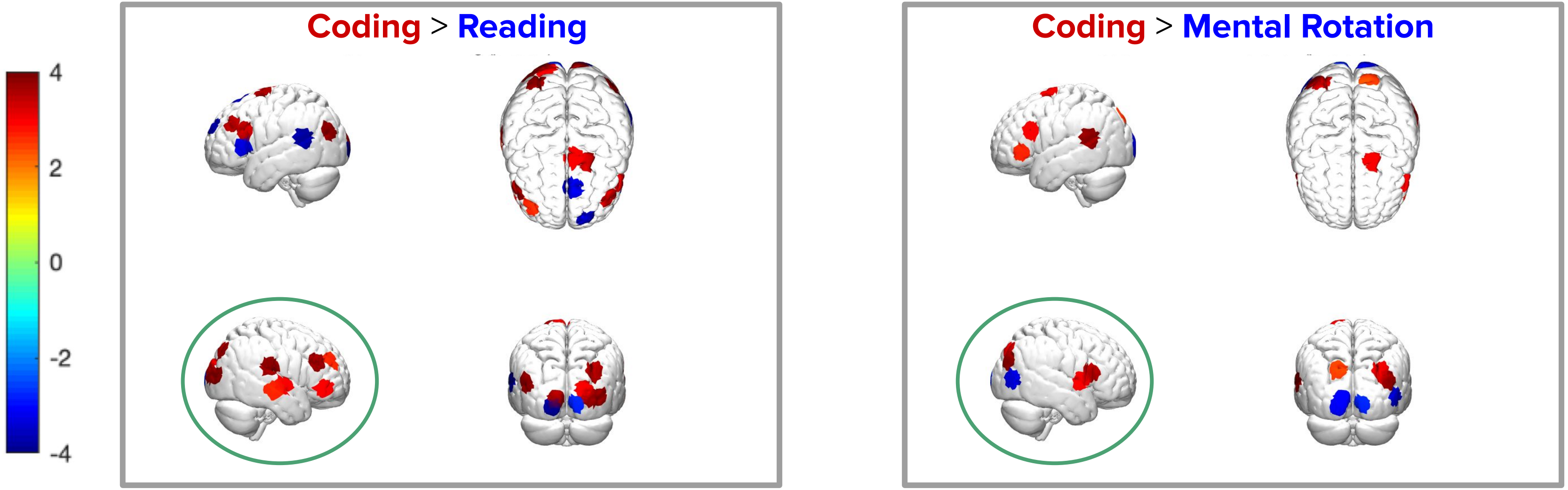


Phase 1 Results: Comparative Brain Activation

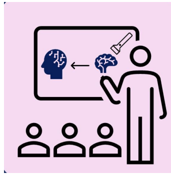


- Question: **Do novices use spatial and/or language areas while programming?**

- Result: While areas associated with both are activated, we find **more substantial differences between Coding and Reading** than between **Coding and Mental Rotation**

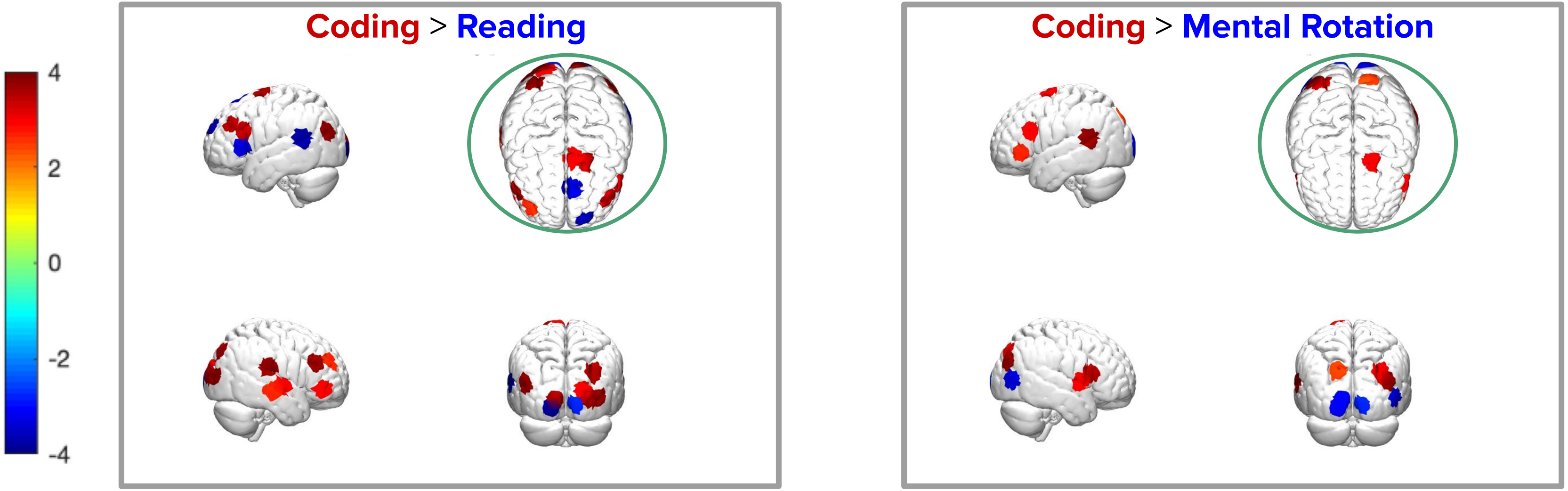


Phase 1 Results: Comparative Brain Activation

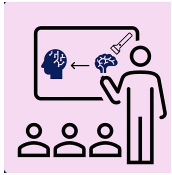


- Question: **Do novices use spatial and/or language areas while programming?**

- Result: While areas associated with both are activated, we find **more substantial differences between Coding and Reading** than between **Coding and Mental Rotation**



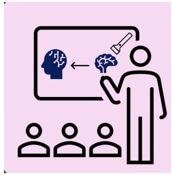
Phase 1 Results: Comparative Brain Activation



- Question: **Do novices use spatial and/or language areas while programming?**

- Result: While areas associated with both are activated, we find **more substantial differences between Coding and Reading** than between **Coding and Rotation**

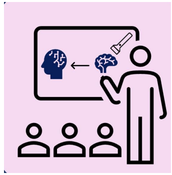
Phase 1 Results: Comparative Brain Activation



- Question: **Do novices use spatial and/or language areas while programming?**
- Result: While areas associated with both are activated, we find **more substantial differences between Coding and Reading** than between **Coding and Rotation**
- We also find that for novices coding engages more **working memory** and is more **cognitively challenging** than does either mental rotation or prose reading

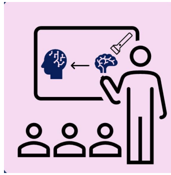
So for novices, we see more differences with reading that spatial ability. Now what?

Phase 1 Results: Comparing to Experts



- Question: **How does this finding compare to previous studies with experts?**
- Floyd *et al.* found that coding and prose tasks are more similar in terms of neural activity for senior undergraduate than for mid-level undergraduates
- Our results: **the pattern may continue to novices**. For less experienced programmers, **programming and reading show little cognitive similarity**
- Implications for developer training and pedagogy:
  - Perhaps **spatial skills enable general problem solving** for novices, but **domain-specific programming strategies** use **more reading-associated** cognitive processes
  - Directly training reading-based domain-specific strategies may help **novices become experts faster**

Phase 1 Summary

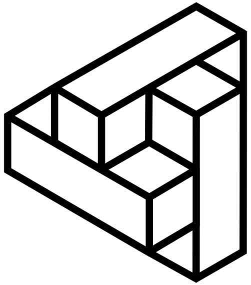


For novices, **spatial reasoning is less distinct to programming than reading** at a cognitive level.

This is **in contrast to results with expert developers**, and has **implications for** future programming **training** or **interventions**.

Phase 2: Transfer Training

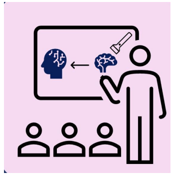
A Tale of Two Cognitive Interventions



vs.

Standardized and Validated Spatial Reasoning Training

Our Novel CS-focused Technical Reading Training

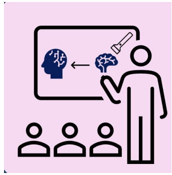


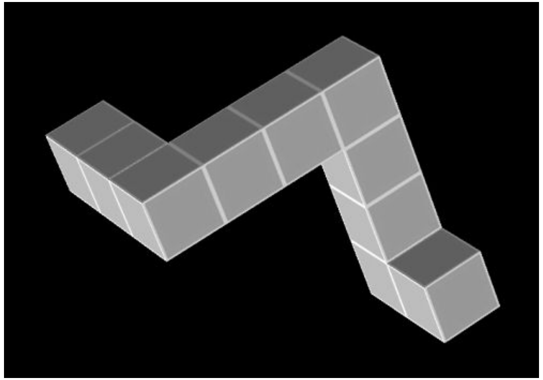
Intervention 1: Spatial Reasoning Training

- **Spatial Reasoning** is the ability to mentally manipulate 2D and 3D shapes

- We use a **validated pre-made Spatial Reasoning Training Curriculum** developed for engineering students
  - Developed by Sorby *et al.* (2000)

- Includes sketching practice of shape rotation projection, and folding

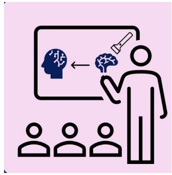




Intervention 2: Technical Reading Training

- We developed an intervention to teach **strategies** for **efficiently understanding scientific writing**

- Strategies focused on **using structural cues to scan academic papers** to retrieve and understand key points
  - Inspired by eye tracking findings: **experienced programmers tend to read code non-linearly**, focusing on high level features.



Novices *Reading Code*

```
import json
import os
import fixTheories.factory as theory_factory

class InFix:

    def __init__(self, inFix_config, inFix_log):
        self.config = inFix_config
        self.log = inFix_log

    def find_fixes(self, filt=None):
        """
        This function finds all the fixes required by the initializing configuration
        Filter is an optional filter that says which sessions to consider
        """

        # See if the config file is indicating that this run is resuming a previous partial run

        to_return = []

        # Load the partial file if there is one
        if self.config.partial_file is not None:
            with open(self.config.partial_file, 'r') as f:
                f = iter(f)
                to_return = []
                for line in f:
                    try:
                        if line.startswith('NEXT'):
                            general_config = json.loads(next(f))
                            experiment_results = json.loads(next(f))
                            if len(experiment_results) == 0:
                                print("This experiment was not actually completed")
                                continue
                            to_return.append((general_config, experiment_results))
                    except StopIteration:
                        pass

        # Make dictionary of values
        zzz = set()
        for config, result in to_return:
            zzz.add(config["UniqueId"])
        theories = self.config.get_theories()
        global_theory_config = [self.config.get_specific_theory_info(x) for x in theories]
        global_theory_factory = [theory_factory.get_theory_solver(i, global_theory_config[0], self.log) for i i
        theory_results = [[] * len(theories)]

        # Now, loop through all the files in path
        base_path = self.config.get_session_path()
        counter = 1
        ak = 0
        for folder in os.listdir(base_path):
            ak += 1
            print("Number: {}".format(ak))
            print(folder)
            # Check here if the folder has already been done by the thing in the config
            if folder in zzz:
```

Busjahn, et al., 2015

Experts *Reading Code*


```
import json
import os
import fixTheories.factory as theory_factory

class InFix:

    def __init__(self, inFix_config, inFix_log):
        self.config = inFix_config
        self.log = inFix_log

    def find_fixes(self, filt=None):
        """
        This function finds all the fixes required by the initializing configuration
        Filter is an optional filter that says which sessions to consider
        """

        # See if the config file is indicating that this run is resuming a previous partial run

        to_return = []

        # Load the partial file if there is one
        if self.config.partial_file is not None:
            with open(self.config.partial_file, 'r') as f:
                f = iter(f)
                to_return = []
                for line in f:
                    try:
                        if line.startswith('NEXT'):
                            general_config = json.loads(next(f))
                            experiment_results = json.loads(next(f))
                            if len(experiment_results) == 0:
                                print("This experiment was not actually completed")
                                continue
                            to_return.append((general_config, experiment_results))
                    except StopIteration:
                        pass

        # Make dictionary of values
        zzz = set()
        for config, result in to_return:
            zzz.add(config["UniqueId"])
        theories = self.config.get_theories()
        global_theory_config = [self.config.get_specific_theory_info(x) for x in theories]
        global_theory_factory = [theory_factory.get_theory_solver(i, global_theory_config[0], self.log) for i i
        theory_results = [[] * len(theories)]

        # Now, loop through all the files in path
        base_path = self.config.get_session_path()
        counter = 1
        ak = 0
        for folder in os.listdir(base_path):
            ak += 1
            print("Number: {}".format(ak))
            print(folder)
            # Check here if the folder has already been done by the thing in the config
            if folder in zzz:
```

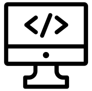
Novices *Reading Code*

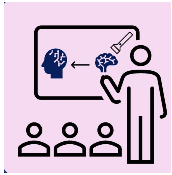
```python
import json
import os
import fixTheories.factory as theory_factory

class InFix:

    def __init__(self, inFix_config, inFix_log):
        self.config = inFix_config
        self.log = inFix_log

    def find_fixes(self, filt=None):
        """
        This function finds all the fixes required by the initializing configuration
        Filter is an optional filter that says which sessions to consider
        """

        # See if the config file is indicating that this run is resuming a previous partial run

        to_return = []

        # Load the partial file if there is one
        if self.config.partial_file is not None:
            with open(self.config.partial_file, 'r') as f:
                f = iter(f)
                to_return = []
                for line in f:
                    try:
                        if line.startswith('NEXT'):
                            general_config = json.loads(next(f))
                            experiment_results = json.loads(next(f))
                            if len(experiment_results) == 0:
                                print("This experiment was not actually completed")
                                continue
                            to_return.append((general_config, experiment_results))
                    except StopIteration:
                        pass

        # Make dictionary of values
        zzz = set()
        for config, result in to_return:
            zzz.add(config["UniqueId"])
        theories = self.config.get_theories()
        global_theory_config = [self.config.get_specific_theory_info(x) for x in theories]
        global_theory_factory = [theory_factory.get_theory_solver(i, global_theory_config[0], self.log) for i i
        theory_results = [[] * len(theories)]

        # Now, loop through all the files in path
        base_path = self.config.get_session_path()
        counter = 1
        ak = 0
        for folder in os.listdir(base_path):
            ak += 1
            print("Number: {}".format(ak))
            print(folder)
            # Check here if the folder has already been done by the thing in the config
            if folder in zzz:
```

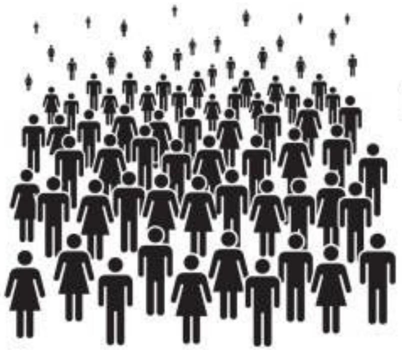
Experts *Reading Code*

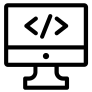

```python
import json
import os
import fixTheories.factory as theory_factory

class InFix:

    def __init__(self, inFix_config, inFix_log):
        self.config = inFix_config
        self.log = inFix_log

    def find_fixes(self, filt=None):
        """
        This function finds all the fixes required by the initializing configuration
        Filter is an optional filter that says which sessions to consider
        """

        # See if the config file is indicating that this run is resuming a previous partial run

        to_return = []

        # Load the partial file if there is one
        if self.config.partial_file is not None:
            with open(self.config.partial_file, 'r') as f:
                f = iter(f)
                to_return = []
                for line in f:
                    try:
                        if line.startswith('NEXT'):
                            general_config = json.loads(next(f))
                            experiment_results = json.loads(next(f))
                            if len(experiment_results) == 0:
                                print("This experiment was not actually completed")
                                continue
                            to_return.append((general_config, experiment_results))
                    except StopIteration:
                        pass

        # Make dictionary of values
        zzz = set()
        for config, result in to_return:
            zzz.add(config["UniqueId"])
        theories = self.config.get_theories()
        global_theory_config = [self.config.get_specific_theory_info(x) for x in theories]
        global_theory_factory = [theory_factory.get_theory_solver(i, global_theory_config[0], self.log) for i i
        theory_results = [[] * len(theories)]

        # Now, loop through all the files in path
        base_path = self.config.get_session_path()
        counter = 1
        ak = 0
        for folder in os.listdir(base_path):
            ak += 1
            print("Number: {}".format(ak))
            print(folder)
            # Check here if the folder has already been done by the thing in the config
            if folder in zzz:
```


Busjahn, et al., 2015

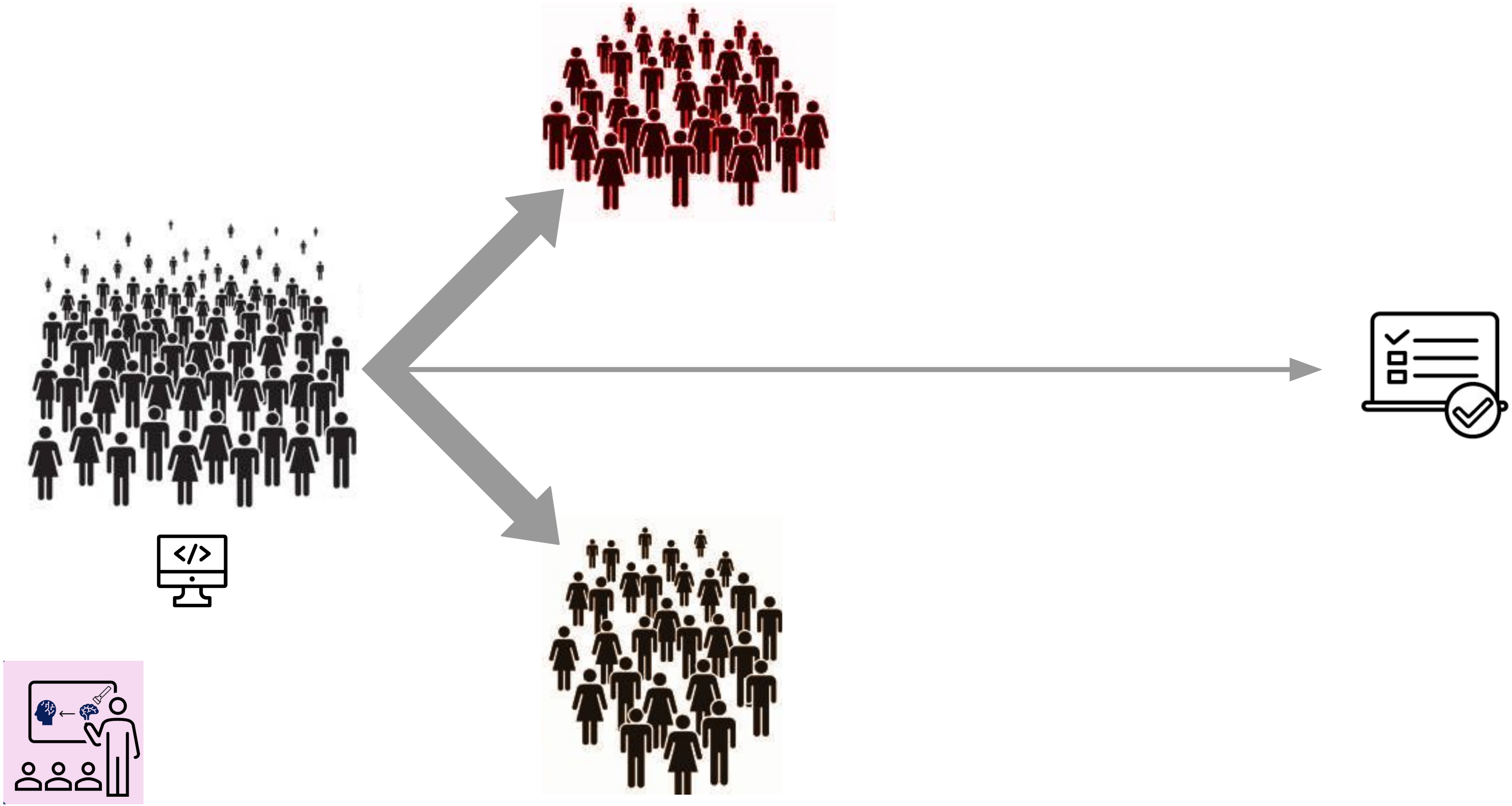


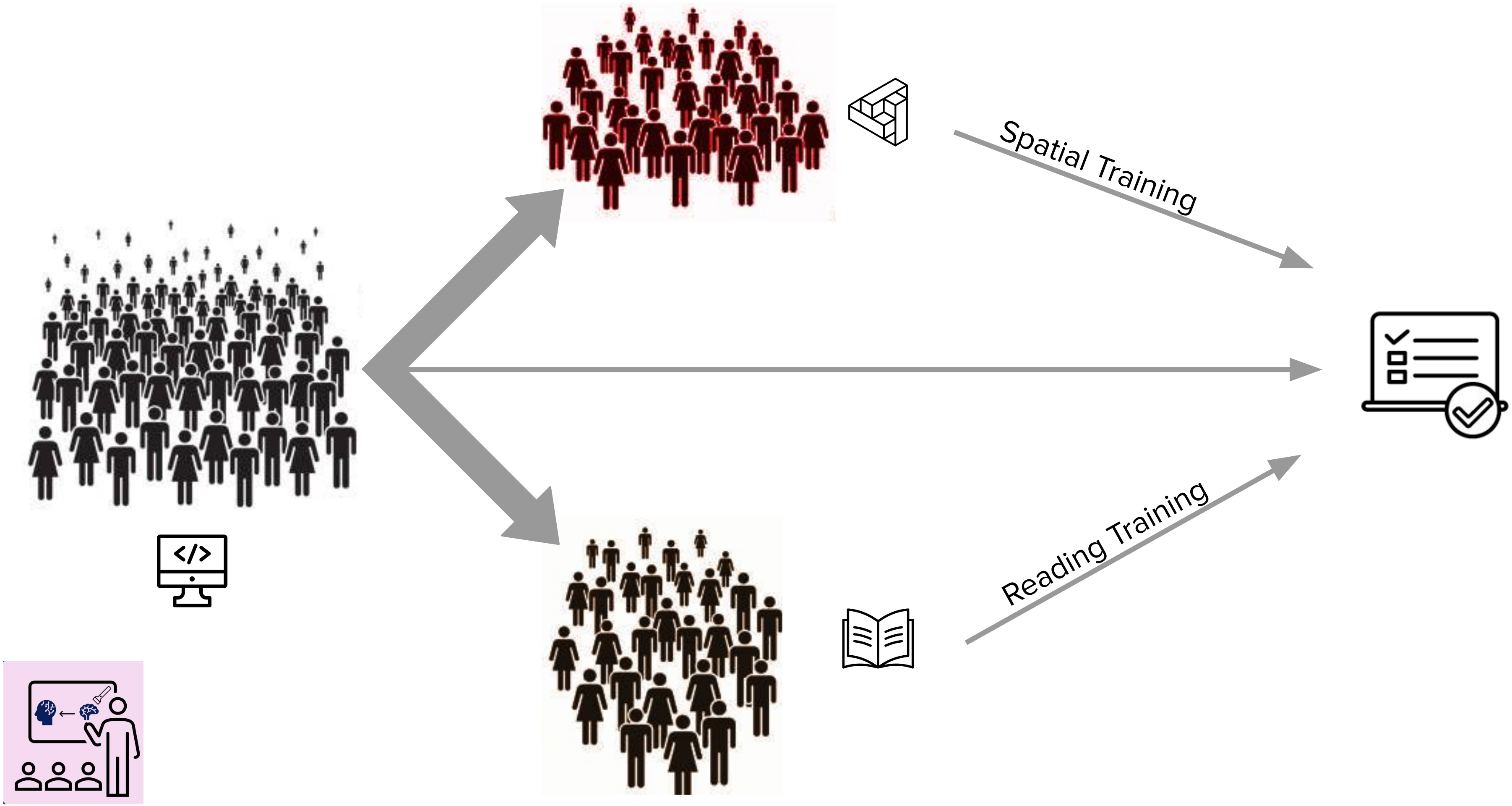




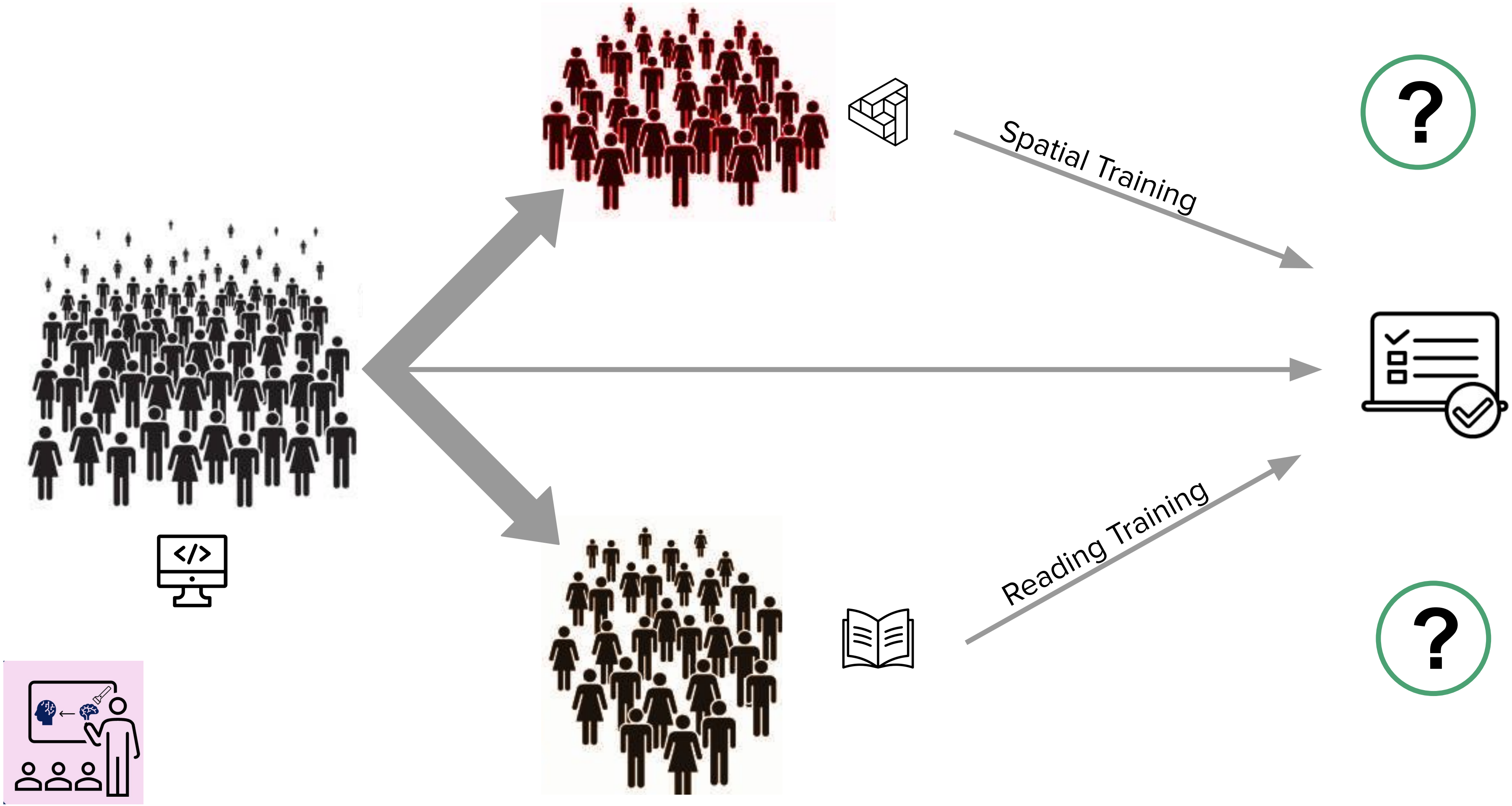


Semester CS1 Course With Final Exam



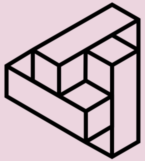


Spatial Training
Reading Training

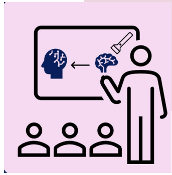

Spatial Training
Reading Training
?
?

Transfer Training Results: Which Group Did Better?

Spatial Reasoning Training

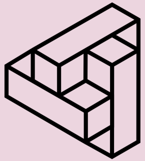


Technical Reading Training



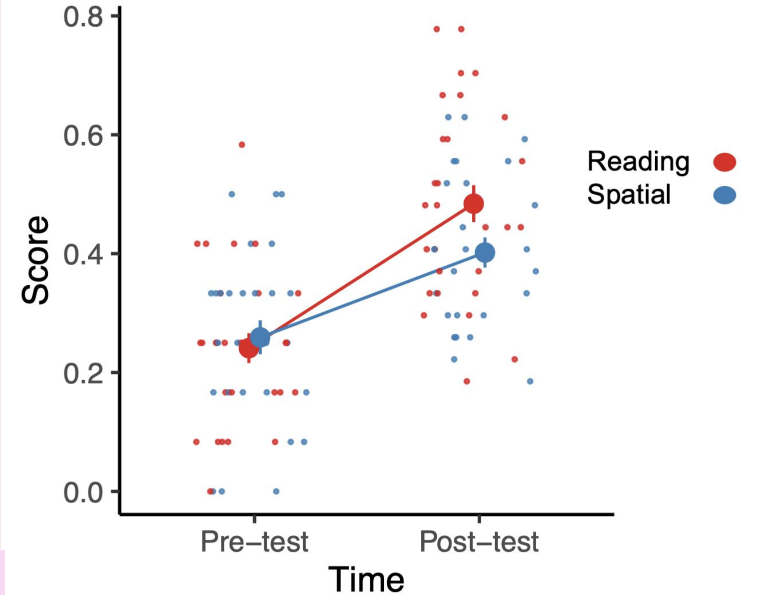
Transfer Training Results: Which Group Did Better?

Spatial Reasoning Training

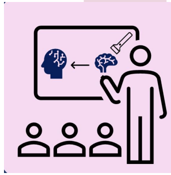


Technical Reading Training

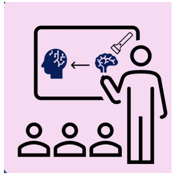
Transfer Training Results: Which Group Did Better?



Technical Reading Training

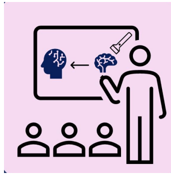


Now that we know that our Reading Training *transferred* to CS1, **what programming skill** did it help?

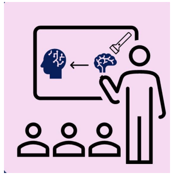


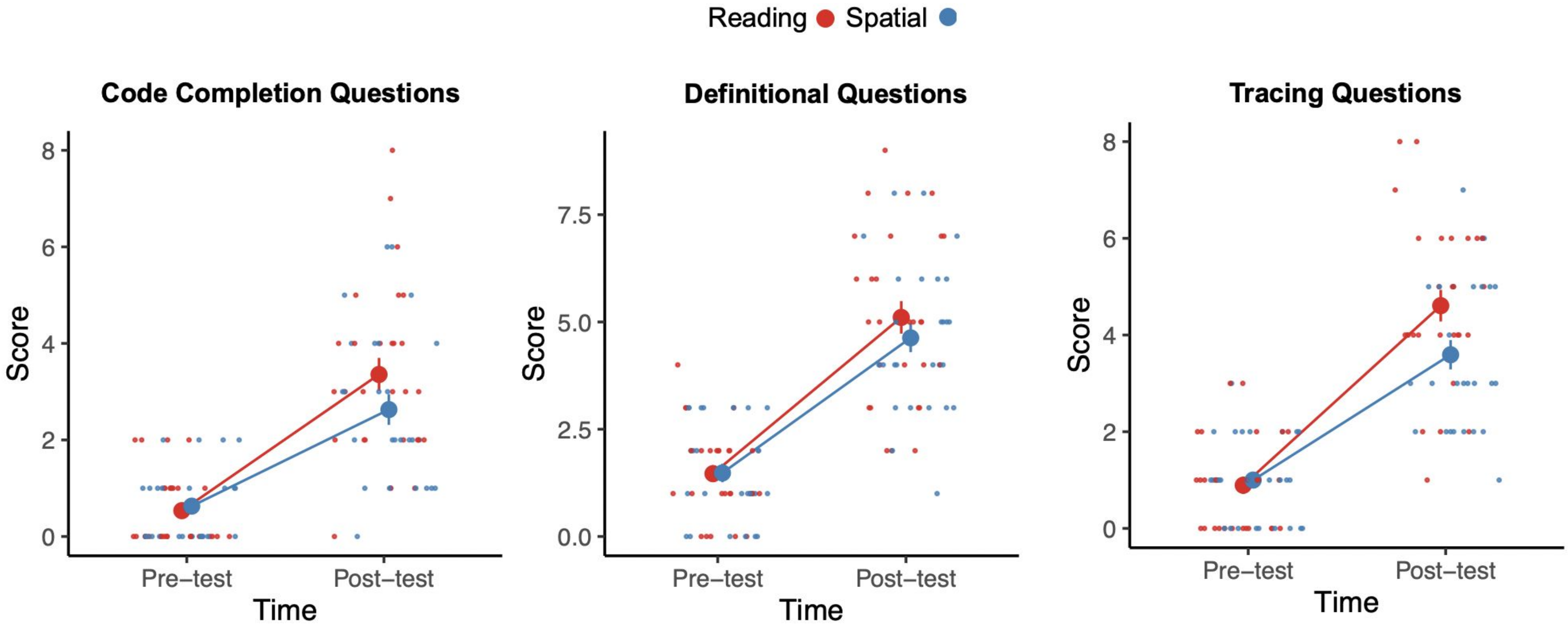
Now that we know that our Reading Training *transferred* to CS1, **what programming skill** did it help?

Our final programming assessment (the SCS1) had three types of questions: **code completion**, **definitional**, and **code tracing**

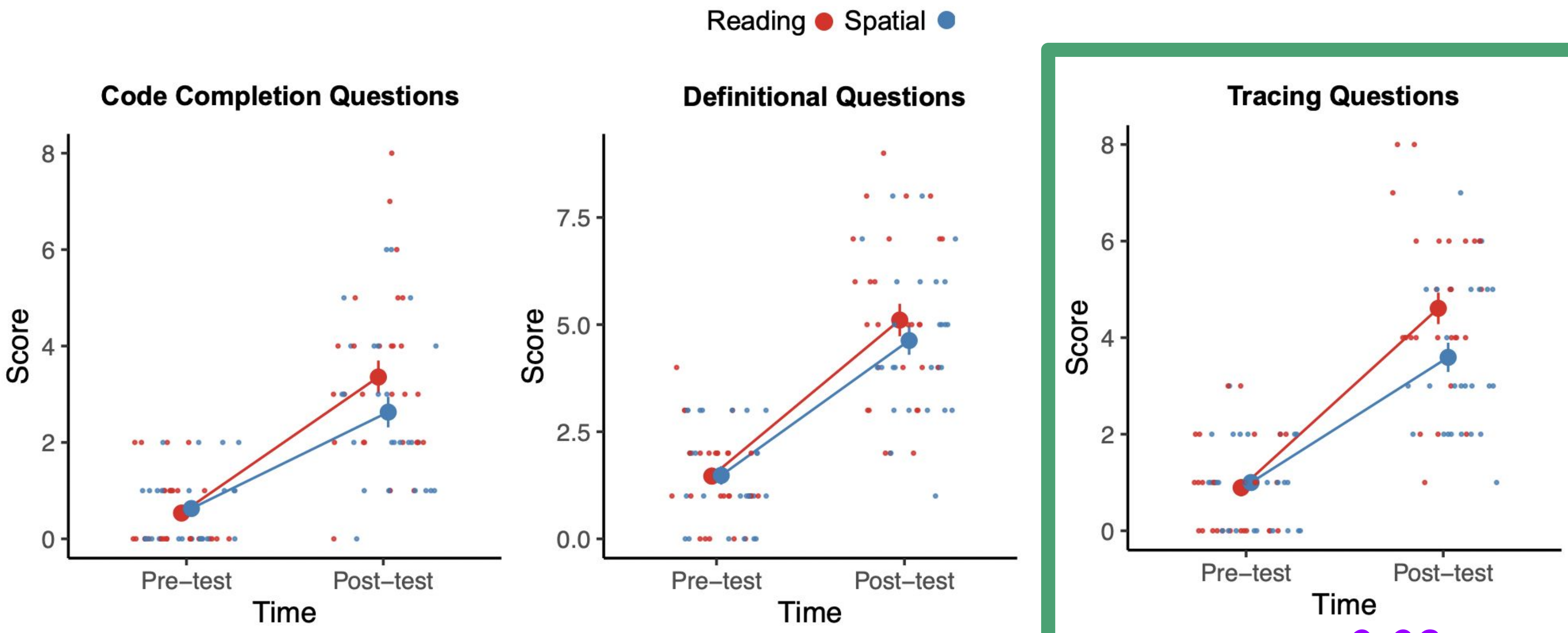


How did the Reading Training Help?



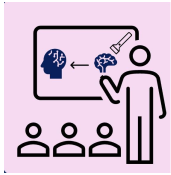


How did the Reading Training Help?

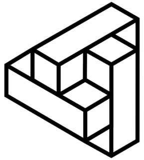


p = 0.03

Transfer Training Results: Summary

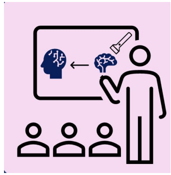


We compared the effects of **Spatial Reasoning Training** and our novel **CS-focused Technical Reading Training** on CS1 students.

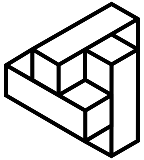


We found that **our Technical Reading Training helped programming ability more**, especially **helping novices trace through code**

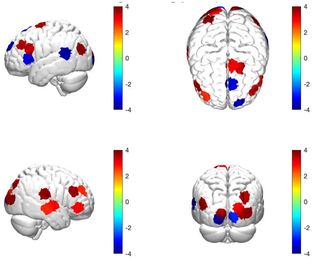
Transfer Training Results: Summary

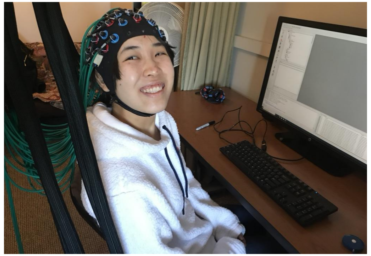


We compared the effects of **Spatial Reasoning Training** and our novel **CS-focused Technical Reading Training** on CS1 students.



We found that **our Technical Reading Training helped programming ability more**, especially **helping novices trace through code**





Developing Efficient and Usable Programming Support

Can we support non-traditional novices in **writing more correct code faster?**

Designing Effective Developer Training

Can we use **cognitive insights** to inform training and **improve programming outcomes?**

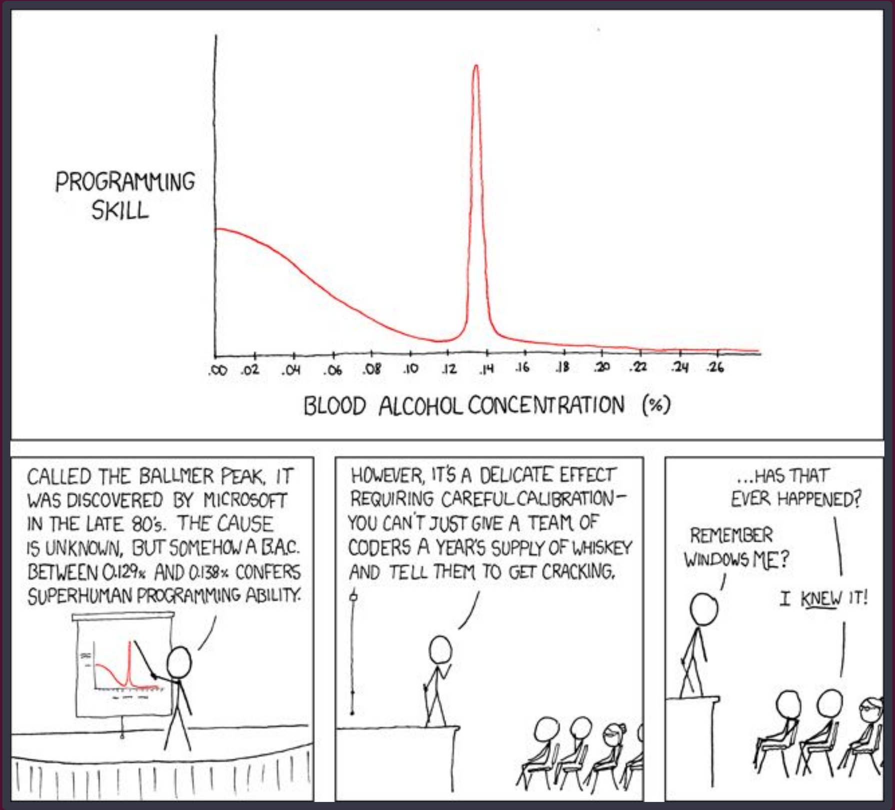
Understanding External Productivity Factors

How does **psychoactive substance** use **impact software productivity?**

Psychoactive Substances and Programming?

A case study on how understudied external factors can impact software productivity

ICSE 2022, 2024



Credit: XKCD Comic, https://xkcd.com/323/

CULTURE OF PSYCHOACTIVE SUBSTANCE USE AND SOFTWARE

Based upon my experiences and observations:

- caffeine
- nicotine
- alcohol
- ritalin
- modafinil

I've never met a developer that didn't use one of the aforementioned drugs *during work*.

Coder's High

Programming is just like drugs, except the dealer pays you.

BY DAVID AUERBACH JUNE 17, 2014 • 12:02 PM

"Taking LSD was a profound experience, one of the most important things in my life"

\- Steve Jobs

OLIVIA SOLON LONG READS 24.08.2016 08:28 AM

Under pressure, Silicon Valley workers turn to LSD microdosing

However, this culture may conflict with some organizational structures –

Take cannabis-related policies as an example:

We have a strict drug and alcohol policy. Employees are not permitted to use, possess, sell, transfer, manufacture, distribute, or be under the influence of illegal drugs on Cisco-owned or leased property, during working hours, while on company business, or while using company property.

Although certain jurisdictions may allow the prescription or other use of marijuana, this policy also applies to marijuana, which remains illegal under U.S. Federal law. Employees are not permitted to use, possess, sell, transfer, manufacture, distribute or be under the influence of these drugs while on Cisco owned or leased property, during working hours, while on company business, or while using company property. In additi
on du
or the

29% of software developers have taken a drug test for a programming-related job. (Endres et al, 2022)

MOTHERBOARD
TECH BY VICE

The FBI Says It Can't Find Hackers to Hire Because They All Smoke Pot

The FBI is struggling to find good hackers because of marijuana rules

By MARY SCHUMACHER
THE FRESH TOAST | APR 23, 2018 AT 11:52 AM

Despite this conflict, **little empirical research has been conducted** on psychoactive substance use in software development

We want to know if, when, or why developers use substances while programming: **tech companies cannot effectively evaluate existing anti-drug policies**

And we want to build a mathematical model of **how such substances actually impact programmers**

We present results from the:

First large-scale empirical study of psychoactive substance use in software engineering

(800+ programmers, 450 full-time devs)

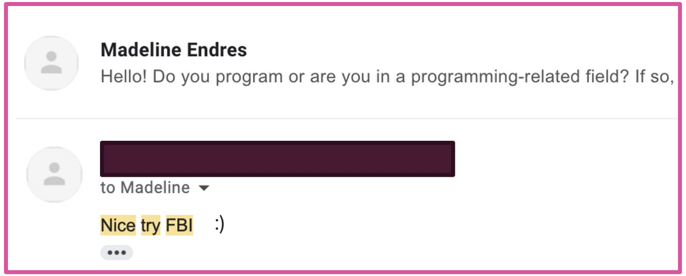
First controlled observational study of cannabis and programming

(70+ programmers, pre-registered hypotheses)

Psychoactive Substances Exploratory Survey: **Summary**

803 survey responses:

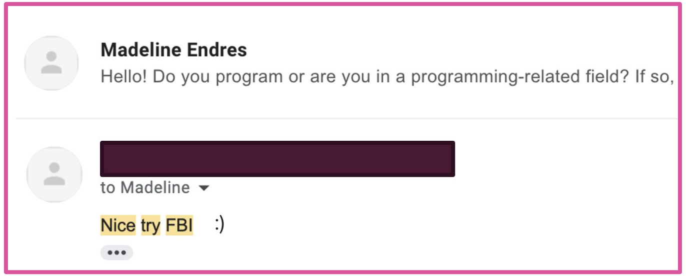
- 440 from GitHub Emails
- 339 from University of Michigan
- 24 from Social Media
- 56% Have full-time programming jobs, 36% are students



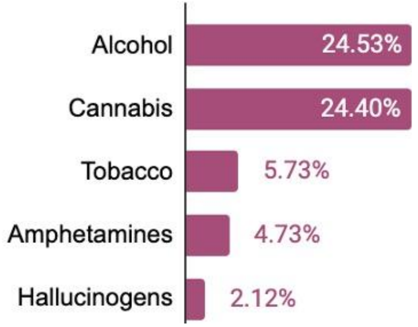
Psychoactive Substances Exploratory Survey: **Summary**

803 survey responses:

- 440 from GitHub Emails
- 339 from University of Michigan
- 24 from Social Media
- 56% Have full-time programming jobs, 36% are students



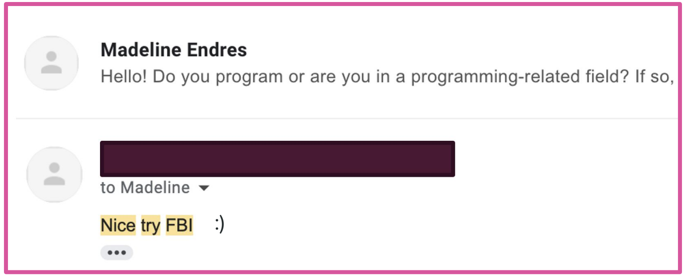
Usage While Programming in Last Year



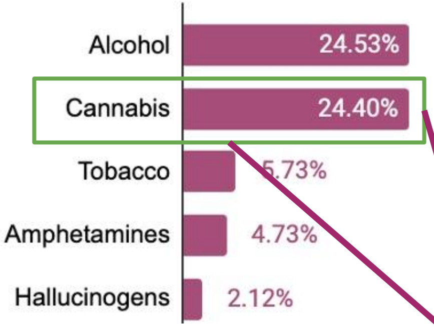
Psychoactive Substances Exploratory Survey: **Summary**

803 survey responses:

- 440 from GitHub Emails
- 339 from University of Michigan
- 24 from Social Media
- 56% Have full-time programming jobs, 36% are students



Usage While Programming in Last Year



- 33% use for **work-related tasks**
- 11% use at a **frequency likely to be caught by a drug test**

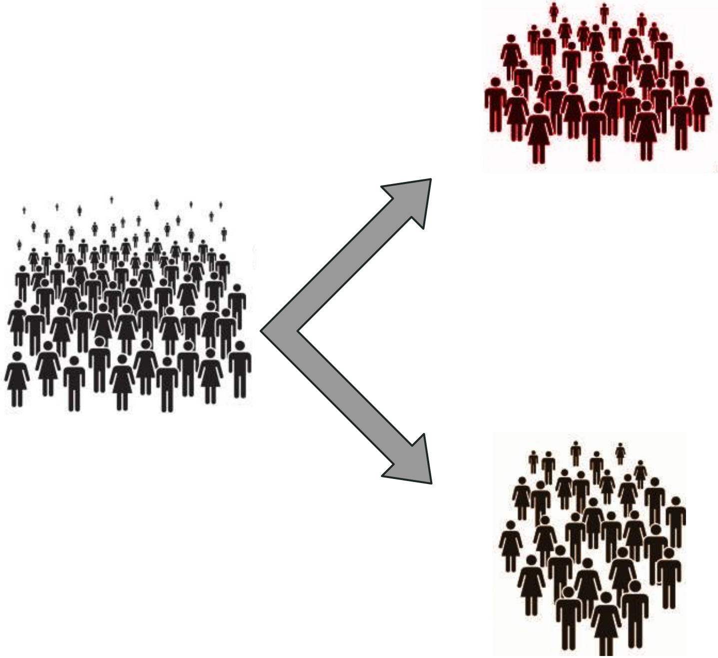
A Controlled Observational Study: Cannabis

- Goal: To **build a mathematical model of how cannabis use impacts programming.**
  - We want our model to be rigorous enough to be used by individual developers and policy makers alike in making more informed cannabis and programming decisions.
  - We **pre-registered our hypotheses** to facilitate future replication.

A Controlled Observational Study: Cannabis

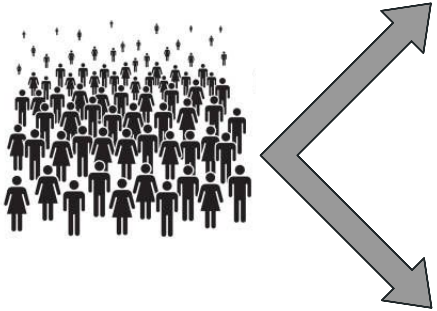
- Goal: To **build a mathematical model of how cannabis use impacts programming.**
  - We want our model to be rigorous enough to be used by individual developers and policy makers alike in making more informed cannabis and programming decisions.
  - We **pre-registered our hypotheses** to facilitate future replication.

- Design Considerations:
  - Achieving **sufficient statistical power** to answer our **pre-registered research questions**
  - Balancing **ecological validity** with **experimental control**
  - Maximizing participant **privacy and safety**

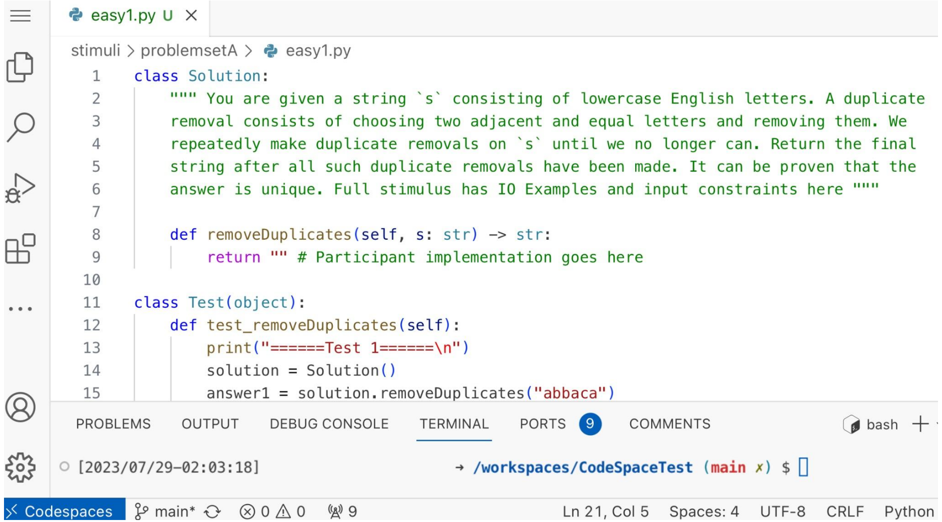


Remote Programming
Session 1

Programming While High

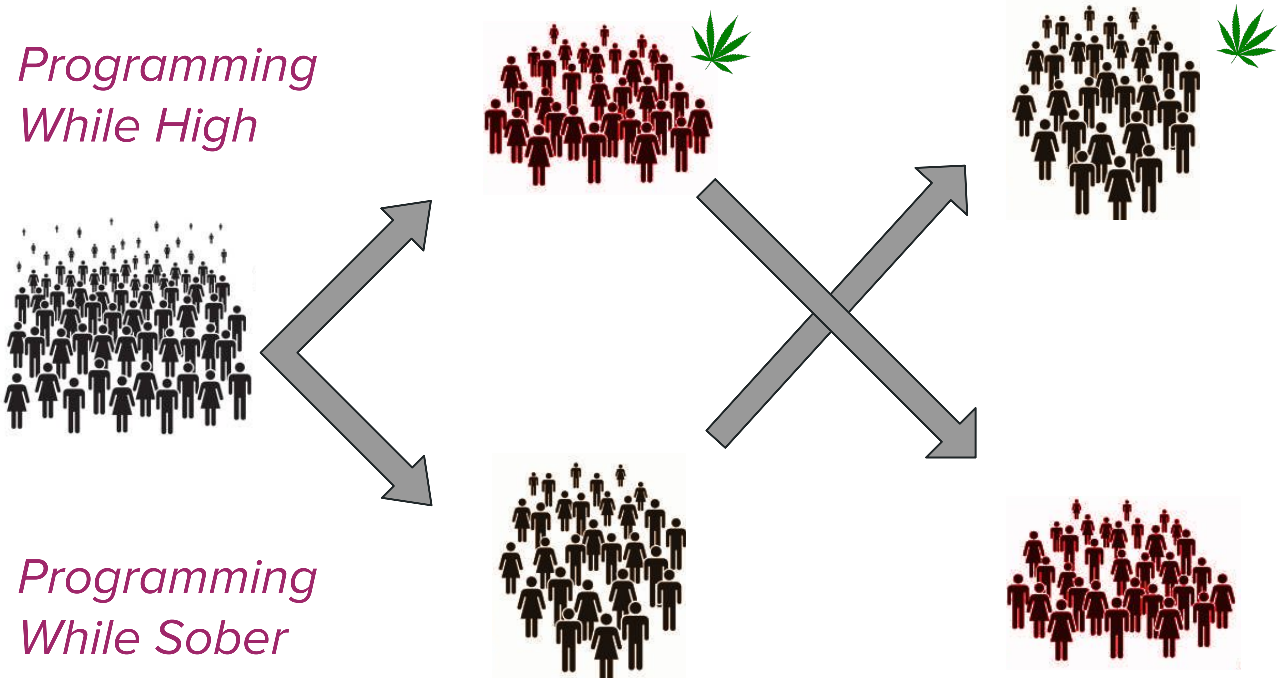


Programming While Sober



Remote Programming Session 1

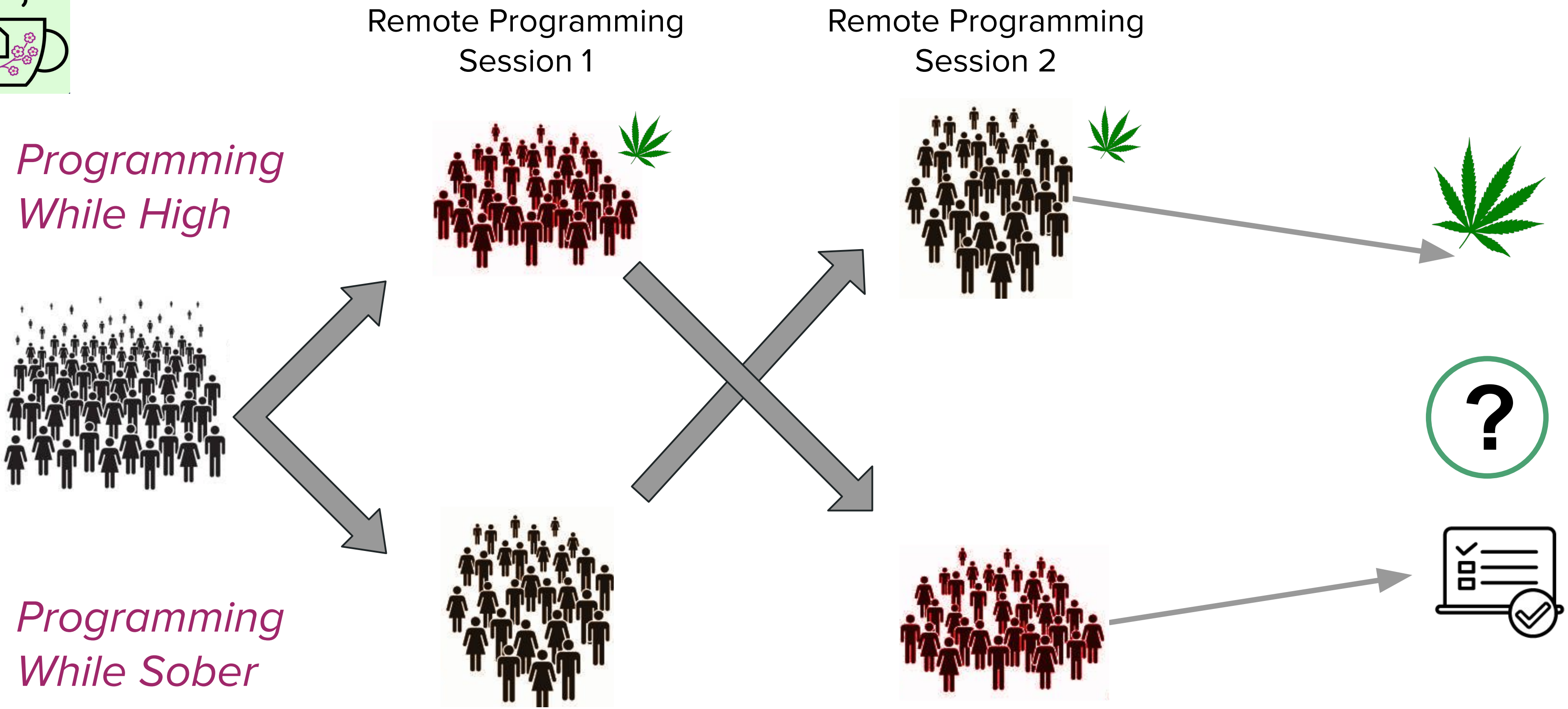
Remote Programming Session 2



easy1.py U

stimuli > problemsetA > easy1.py

```python
class Solution:
    """ You are given a string `s` consisting of lowercase English letters. A duplicate
    removal consists of choosing two adjacent and equal letters and removing them. We
    repeatedly make duplicate removals on `s` until we no longer can. Return the final
    string after all such duplicate removals have been made. It can be proven that the
    answer is unique. Full stimulus has IO Examples and input constraints here """

    def removeDuplicates(self, s: str) -> str:
        return "" # Participant implementation goes here

```



easy1.py U ×

stimuli > problemsetA > easy1.py

```python
class Solution:
    """ You are given a string `s` consisting of lowercase English letters. A duplicate
    removal consists of choosing two adjacent and equal letters and removing them. We
    repeatedly make duplicate removals on `s` until we no longer can. Return the final
    string after all such duplicate removals have been made. It can be proven that the
    answer is unique. Full stimulus has IO Examples and input constraints here """

    def removeDuplicates(self, s: str) -> str:
        return "" # Participant implementation goes here

```

Results: **Pre-registered Hypotheses**

RQ1: How does cannabis intoxication while programming **impact program correctness**?

- Hypothesis: Programs will be **less correct** when written by cannabis-intoxicated programmers.

RQ2: How does cannabis intoxication while programming **impact programming speed**?

- Hypothesis: Cannabis-intoxicated programmers **will take longer to write** programs.

).

Results: Pre-registered Hypotheses

RQ1: How does cannabis intoxication while programming **impact program correctness**?

- Hypothesis: Programs will be **less correct** when written by cannabis-intoxicated programmers.
- Finding: **Cannabis use decreases program correctness** ($0.0005 < p < 0.05$, $0.28 < d$ 0.44, 10-14% fewer passed tests). In particular, cannabis impairs the ability to write and trace through programs.

RQ2: How does cannabis intoxication while programming **impact programming speed**?

- Hypothesis: Cannabis-intoxicated programmers **will take longer to write** programs.
- Finding: **Cannabis use impairs programming speed** ($p < 0.04$, $d = 0.3$, 10-14% slower). This decrease in speed is associated with typing slower, deleting more characters, and more time spent not typing.

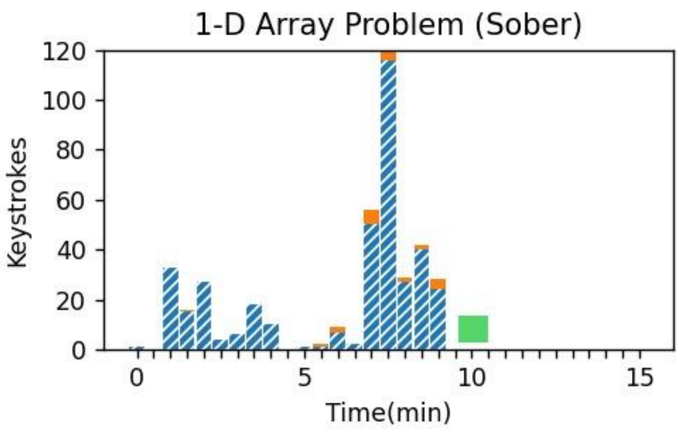
High vs. Sober: How does Cannabis Impair Programming?

Programming While Sober

Normal Keystrokes

Delete Keystrokes

Finished with Correct Solution



High vs. Sober: How does Cannabis Impair Programming?

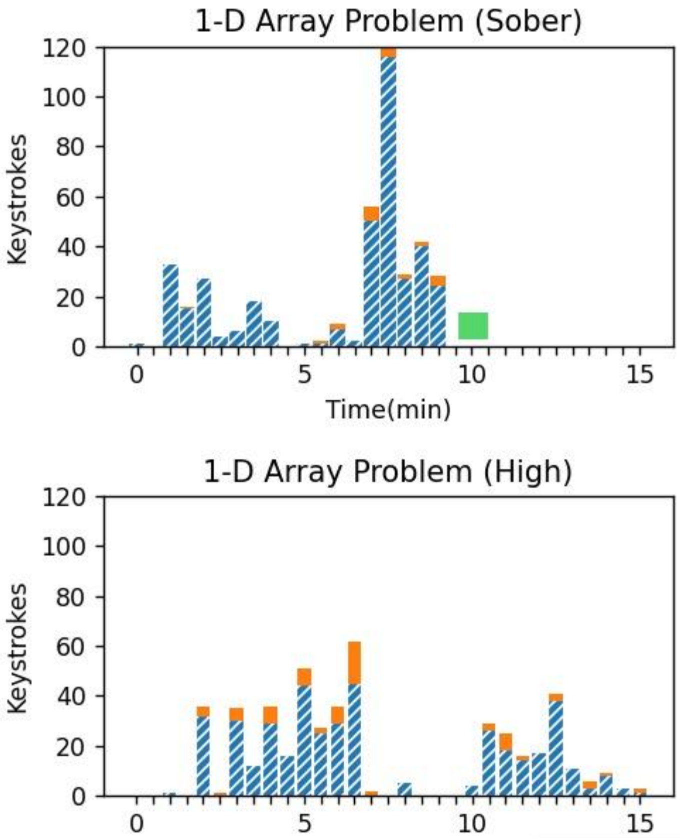
Programming While Sober

Normal Keystrokes

Delete Keystrokes

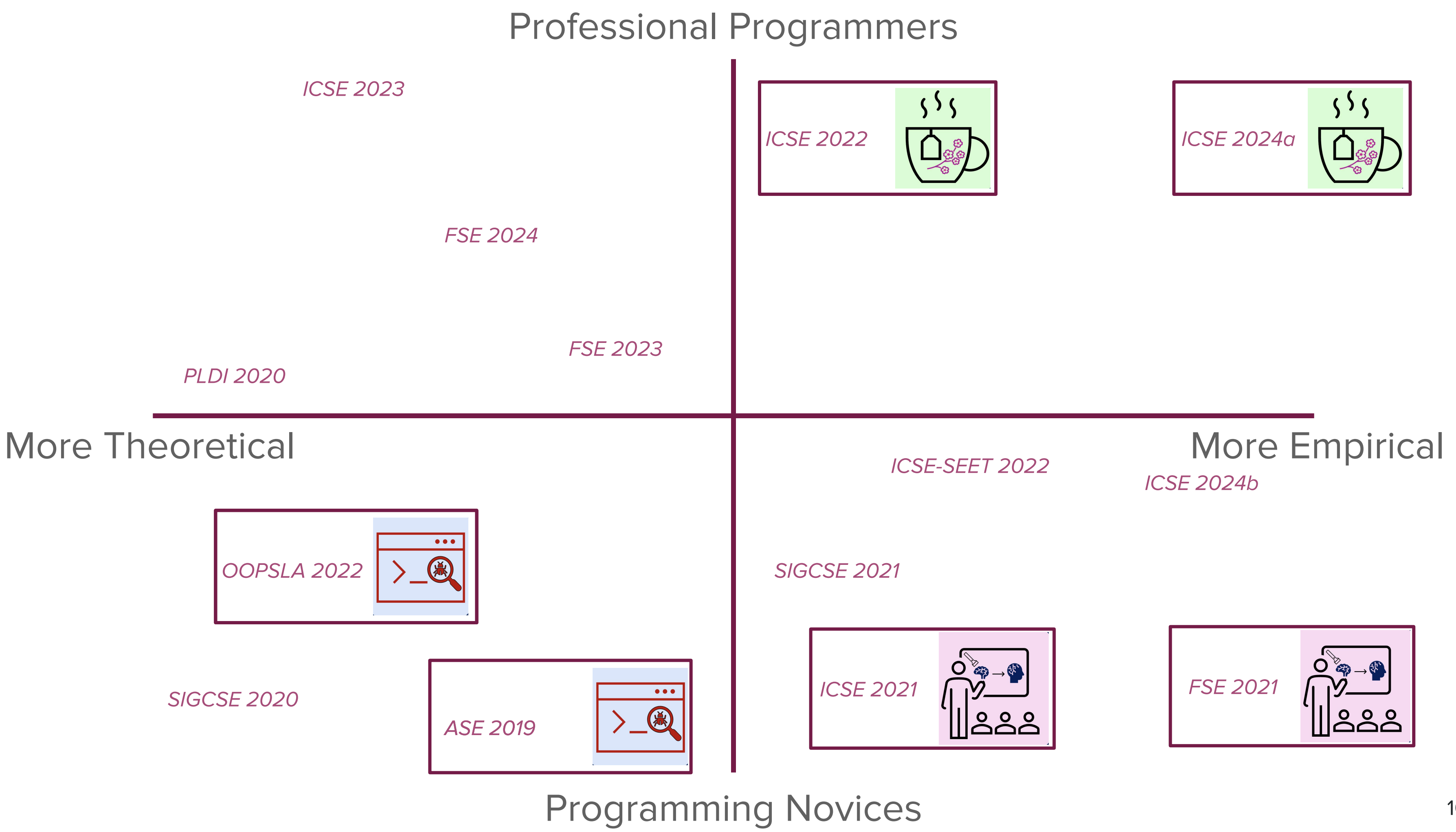
Finished with Correct Solution

Programming While High



Lens 3 - Summary: Psychoactive Substances and Programming

- By surveying 800 programmers, we found that **psychoactive substance use is common in software**
- Despite anecdotes to the contrary, **we only observed evidence of cannabis impairing productivity**
- This work demonstrates the **usefulness of objective measures** and **careful controlled experimental design** to come to evidence-based conclusions on anecdotal software productivity factors



Professional Programmers
ICSE 2023
ICSE 2022
ICSE 2024a
FSE 2024
FSE 2023
PLDI 2020
More Theoretical
More Empirical
ICSE-SEET 2022
ICSE 2024b
OOPSLA 2022
SIGCSE 2021
SIGCSE 2020
ASE 2019
ICSE 2021
FSE 2021
Programming Novices

1. **FSE, 2024** *Can LLMs Transform Natural Language Intent into Formal Methods Postconditions?*
   Endres, M., Fakhoury, S., Chakraborty, S., Lahiri, S.
2. **ICSE, 2024a** *Causal Relationships and Programming Outcomes: A Transcranial Magnetic Stimulation Experiment,*
   Ahmad, H., **Endres, M.**, Newman, K., Santiesteban, P., Shedden, E., Weimer, W. **(Distinguished Paper)**
3. **ICSE, 2024b** *High Expectations: An Observational Study of Programming and Cannabis Intoxication,*
   He, W., Parikh, M., Weimer, W., **Endres, M.**
4. **FSE, 2023** *A Four-Year Study of Student Contributions to OSS with a Lightweight Intervention,*
   Fang, Z., **Endres, M.,** Zimmermann, T., Ford, D., Weimer, W., Leach., K., Huang, Y **(Distinguished Paper)**
5. **ICSE, 2023** *From Organizations to Individuals: Psychoactive Substance Use By Professional Programmers,*
   Newman, K., **Endres, M.,** Weimer, W., Johnson, B.
6. **OOPSLA, 2022** *Seq2Parse: Neurosymbolic Parse Error Repair,*
   Sakkas, G., **Endres, M.,** Guo, P., Weimer, W., Jhala, R.
7. **ICSE, 2022** *Hashing It Out: A Survey of Programmers' Cannabis Usage, Perception, and Motivation,*
   Endres, M., Boehnke, K., Weimer, W.
8. **ICSE-SEET, 2022** *Debugging with Stack Overflow: Web Search Behavior in Novice and Expert Programmers,*
   Li, A., **Endres, M.,** Weimer,
9. **FSE, 2021** *To Read or To Rotate? Comparing the Effects of Technical Reading Training and Spatial Skills Training...*
   Endres, M., Fansher, M., Shah, P., Weimer, W.
10. **ICSE, 2021** *Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study*
    Endres, M., Karas, Z., Hu, Z., Kovelman, I., Weimer, W
11. **SIGCSE, 2021** *An Analysis of Iterative and Recursive Problem Performance,*
    Endres, M., Weimer, W., Kamil, A.
12. **PLDI, 2020** *Type Error Feedback via Analytic Program Repair*
    Sakkas, G., **Endres, M.,**Cosman, B.,Weimer, W.,Jhala, R.
13. **SIGCSE, 2020** *Pablo: Helping Novices Debug Python Code Through Data-Driven Fault Localization*
    Cosman, B., **Endres, M.,** Sakkas, G., Medvinsky, L., Yao-Yuan,Y.,Jhala, R.,Chaudhuri, K.,Weimer, W.
14. **ASE, 2019** *InFix: Automatically Repairing Novice Program Inputs*
    Endres, M., Cosman, B., Sakkas, G., Jhala, R., Weimer, W.

Student Advisees

Yiannos Demetriou
BS CS, 2023

Wenxin He
MS CS, 2024

Annie Li
BS CS, 2022

Kaia Newman
BS CS, 2023

Manasvi Parikh
BS CS, 2023

Interdisciplinary Collaborators

Madison Fansher
Cognitive Psychology

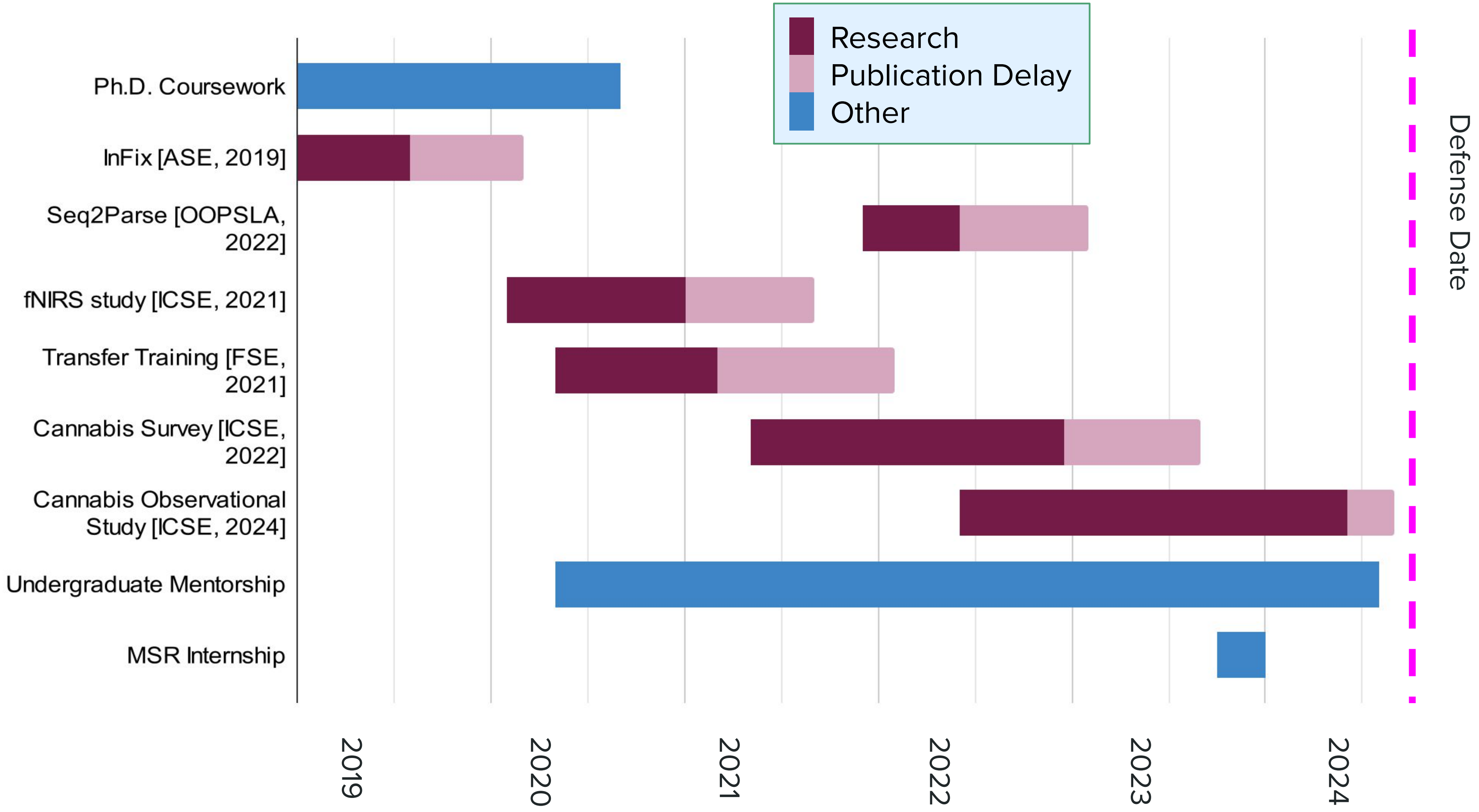
Xiaosu (Frank) Hu
Developmental Psychology

Priti Shah
Cognitive Psychology

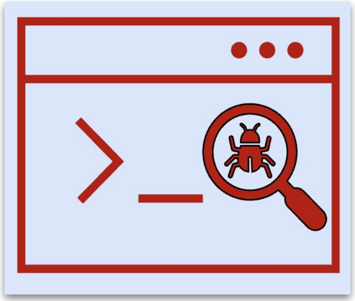
Kevin Bohenke
Anesthesiology and Chronic Pain

Ioulia Kovelman
Developmental Psychology

Ph.D. Timeline

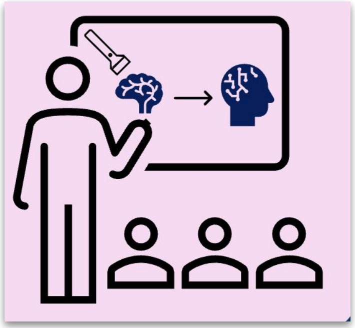


Developing Efficient and Usable Programming Support



Supporting non-traditional novices in **writing more correct code faster**

Designing Effective Developer Training



Use **cognitive insights** to inform training and **improve programming outcomes**

Understanding External Productivity Factors

Exploring how **substance use impacts software productivity**

A Human-Focused Approach to Improving Programmer Productivity

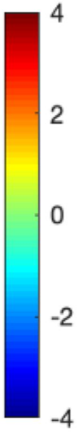
Madeline Endres, PhD Candidate, University of Michigan

Bonus Slides

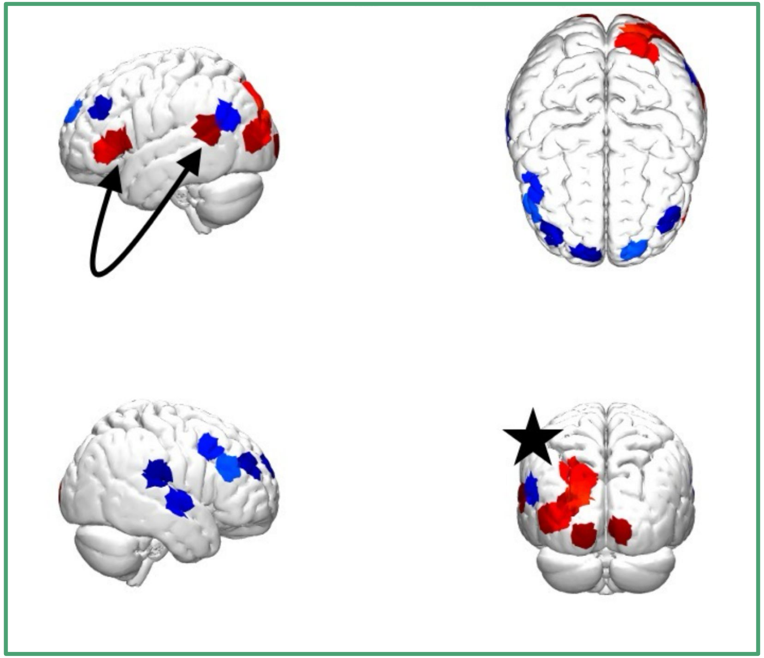
Data Analysis Pipeline

- **Step 1: Preprocessing**
  - Raw fNIRs Data (light intensity levels) → optical density data (how much is being absorbed?)
  - Optical density data → HbO/HbR signal (oxygenated vs deoxygenated blood)
- **Step 2: Individual Modeling**
  - We model the hemodynamic response for each subject individually using a GLM
  - Quality control checks are used to filter noisy data (e.g., signal to noise ratio, anticorrelation of HbO and HbR, visual activation spot checking)
- **Step 3: Group Modeling**
  - Used a linear mixed effects models, contrasting Task > Baseline activations
  - We applied a false-discovery rate (FDR) correction ($q < 0.05$) to account for multiple comparisons

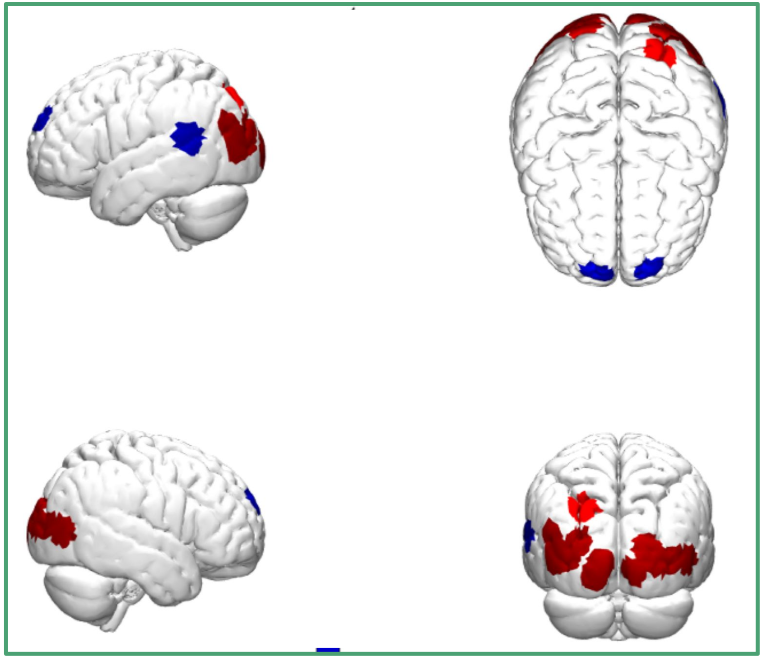
Results: Baseline Activation



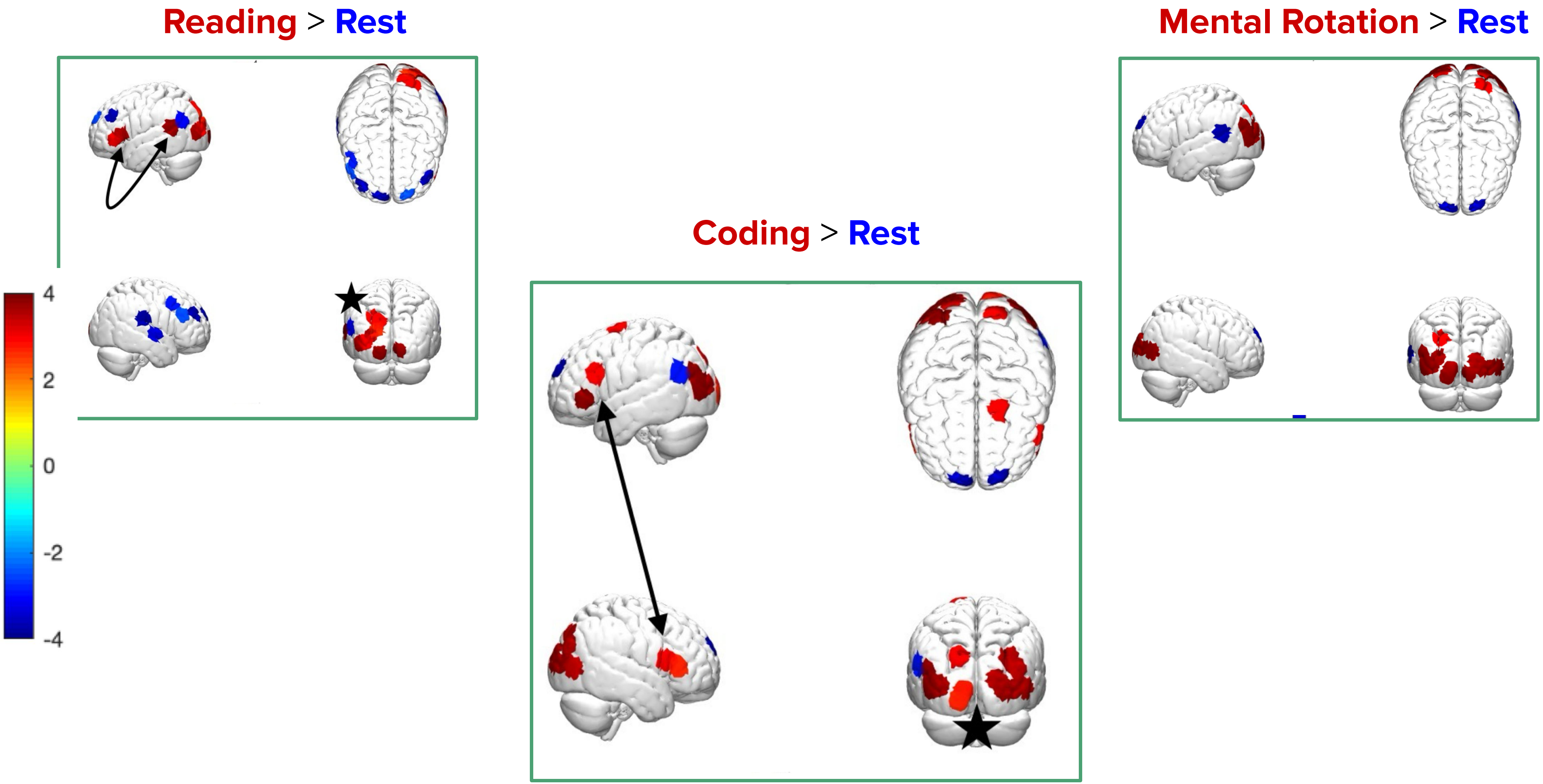
Reading > **Rest**



Mental Rotation > **Rest**



Results: Baseline Activation



Supporting Publications

1. **ICSE, 2024** *Causal Relationships and Programming Outcomes: A Transcranial Magnetic Stimulation Experiment,* Ahmad, H., **Endres, M.**, Newman, K., Santiesteban, P., Shedden, E., Weimer, W.
2. **ICSE, 2024** *High Expectations: An Observational Study of Programming and Cannabis Intoxication,* He, W., Parikh, M., Weimer, W., **Endres, M.**
3. **FSE, 2023** *A Four-Year Study of Student Contributions to OSS with a Lightweight Intervention,* Fang, Z., **Endres, M.,** Zimmermann, T., Ford, D., Weimer, W., Leach., K., Huang, Y
4. **ICSE, 2023** *From Organizations to Individuals: Psychoactive Substance Use By Professional Programmers,* Newman, K., **Endres, M.,** Weimer, W., Johnson, B.
5. **OOPSLA, 2022** *Seq2Parse: Neurosymbolic Parse Error Repair,* Sakkas, G., **Endres, M.,** Guo, P., Weimer, W., Jhala, R.
6. **ICSE, 2022** *Hashing It Out: A Survey of Programmers' Cannabis Usage, Perception, and Motivation,* **Endres, M.,** Boehnke, K., Weimer, W.
7. **ICSE-SEET, 2022** *Debugging with Stack Overflow: Web Search Behavior in Novice and Expert Programmers,* Li, A., **Endres, M.,** Weimer,
8. **FSE, 2021** *To Read or To Rotate? Comparing the Effects of Technical Reading Training and Spatial Skills Training…* **Endres, M.,** Fansher, M., Shah, P., Weimer, W.
9. **ICSE, 2021** *Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study* **Endres, M.,** Karas, Z., Hu, Z., Kovelman, I., Weimer, W
10. **SIGCSE, 2021** *An Analysis of Iterative and Recursive Problem Performance,* **Endres, M.,** Weimer, W., Kamil, A.
11. **PLDI, 2020** *Type Error Feedback via Analytic Program Repair* Sakkas, G.,**Endres, M.,**Cosman, B.,Weimer, W.,Jhala, R.
12. **SIGCSE, 2020** *Pablo: Helping Novices Debug Python Code Through Data-Driven Fault Localization* Cosman, B., **Endres, M.,** Sakkas, G., Medvinsky, L., Yao-Yuan,Y.,Jhala, R.,Chaudhuri, K.,Weimer, W.
13. **ASE, 2019** *InFix: Automatically Repairing Novice Program Inputs* **Endres, M.,** Cosman, B., Sakkas, G., Jhala, R., Weimer, W.

Visualize Python, Java, JavaSc

pythontutor.com/visualize.html#mode=edit

Write code in Python 3.6

```
u = 42
x = float(input())
print(x * math.e / 2)
```

Help improve this tool by completing a **short user survey**

Please wait ... executing (takes up to 10 seconds)

Live Programming Mode

Anecdotal evidence abounds:

Many programmers use cannabis while programming

Posted by u/SmartTest 5 months ago

31

Coding + Cannabis Use

Hey fellow programmers!

I wanted to see what your
and working in the field.

Do you use cannabis and
career?

[redacted] Software Engineer at WeedMaps (2017-present)

Updated Aug 17, 2019 · Upvoted by Hasib Al Muhaimin, IOI participant '13, '14, '15, ACM-ICPC World Finalist 2016 and Stanley Munson, AAS Computer Programming & Network Administrator, Helena College (2019)

At WeedMaps we smoke and code everyday, all day. Some of the smartest, most insightful people I know smoke pot daily.

g thoughts might get in
ally isn't much of an
o really dive deep into

[redacted] Blockchain Engineer at Parallelcoin (2018-present)

Answered Oct 18, 2019

I believe that there is something about the type of brains that are common among programmers that get along well with extra THC. I also want to point out that until the 20th century, people were drinking milk and eating meat fed on hemp seed, and

negative_epsilon · 8 yr. ago

I have attempted to program while both high and drunk, and neither works well. I might think I write good code, but the next morning I look and there are ridiculous errors that I never would have made sober.

coding-on-marijuana · 8 yr. ago

A fair amount of professional experience here. If you're a responsible, mature, self-respecting adult then it's (mj) a great tool to use for software development. YMMV, this is my experience as reflected by my own strengths and weaknesses as a developer.

Stress: None. Bugs won't bug you, defects won't stress you out, inexplicable side-effects become fun logic puzzles.

Cannabis use **can conflict with corporate anti-drug policies**

This conflict **can lead to hiring shortages!**

We have a strict drug and alcohol policy. Employees are not permitted to use, possess, sell, transfer, manufacture, distribute, or be under the influence of illegal drugs on Cisco-owned or leased property, during working hours, while on company business, or while using company property.

Although certain jurisdictions may allow the prescription or other use of marijuana, this policy also applies to marijuana, which remains illegal under U.S. Federal law. Employees are not permitted to use, possess, sell, transfer, manufacture, distribute or be under the influence of these drugs while on Cisco owned or leased property, during working hours, while on company business, or while using company property. In addition, no employee may report for work, go on or remain on duty while under the influence of, or impaired by, alcohol, or these drugs or substances.

We find that 29% of software developers have taken a drug test for a programming-related job.

MOTHERBOARD
TECH BY VICE

The FBI Says It Can't Find Hackers to Hire Because They All Smoke Pot

The FBI is struggling to find good hackers because of marijuana rules

By MARY SCHUMACHER
THE FRESH TOAST | APR 23, 2018 AT 11:52 AM

How can we represent ill-parsed programs when **training** our classifier?

Buggy Program

```python
def foo(a):
  return a + 42

def bar(a):
  b = foo(a) + 17
  return b +
```

Token Sequence

```
def name(name): \n
indent return name + number \n
dedent \n

def name(name): \n
indent name = name(name) + number \n
return name + \n
dedent end_marker
```

How can we represent ill-parsed programs when **training** our classifier?

Buggy Program

```
def foo(a):
  return a + 42

def bar(a):
  b = foo(a) + 17
  return b +
```

Token Sequence

```
def name(name): \n
indent return name + number \n
dedent \n

def name(name): \n
indent name = name(name) + number \n
return name + \n
dedent end_marker
```

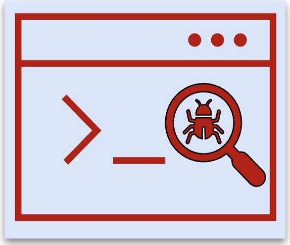
Abstracted Token Sequence

```
Stmt \n

def name Params: \n
indent Stmt \n
return Expr BinOp \n
dedent end_marker
```

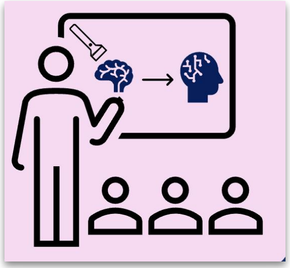
Great! But we have a new problem: **Ambiguity**

each abstracted token sequence can lead to multiple different ECE parse trees!

My Approach To Programming Productivity: What's Next?



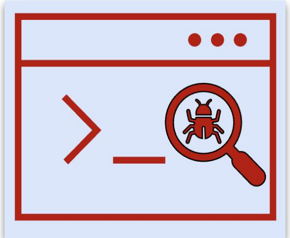
The next generation of
neurosymbolic productivity support

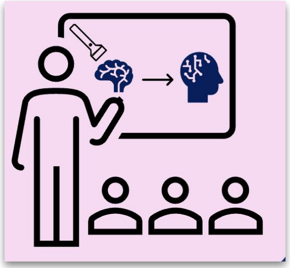


The continued use of medical imaging
to inform programming practice

Using controlled experimental
design and objective measures to
turn anecdote into evidence

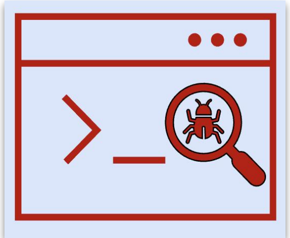
My Approach To Programming Productivity: What's Next?

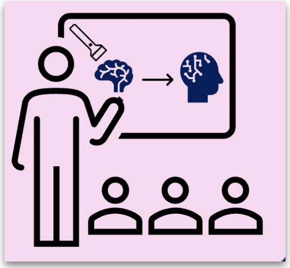




Using **Technical Reading Ability** as a lens for facilitating the ability to understand and communicate complex technical ideas at varying levels of abstraction.

My Approach To Programming Productivity: What's Next?





Using **Technical Reading Ability** as a lens for facilitating the ability to understand and communicate complex technical ideas at varying levels of abstraction.

Increasing participation and retention in computing for diverse programmer groups through **improving developer wellbeing**

How can we represent ill-parsed programs when **training** our classifier?

Buggy Program

```
def foo(a):
  return a + 42

def bar(a):
  b = foo(a) + 17
  return b +
```

Token Sequence

```
def name(name): \n
indent return name + number \n
dedent \n

def name(name): \n
indent name = name(name) + number \n
return name + \n
dedent end_marker
```

Abstracted Token Sequence

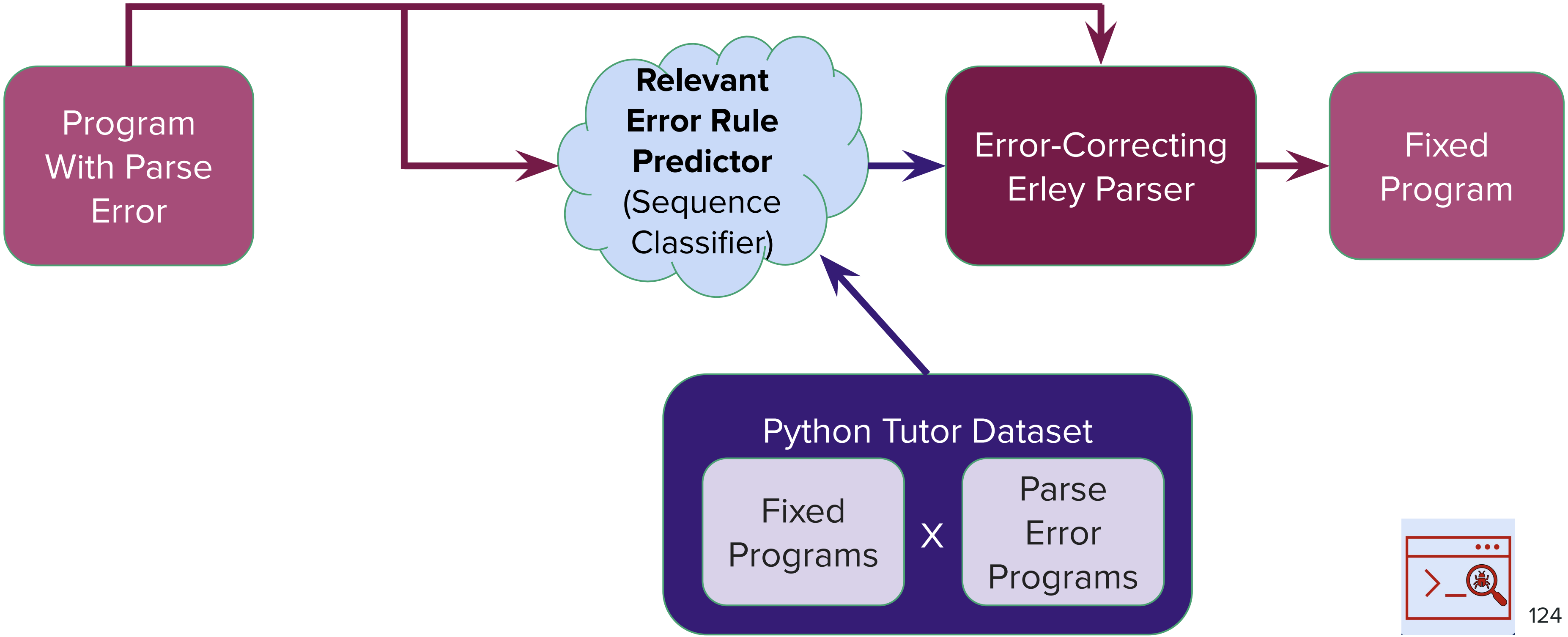
```
Stmt \n

def name Params: \n
indent Stmt \n
return Expr BinOp \n
dedent end_marker
```

Great! But we have a new problem: **Ambiguity**

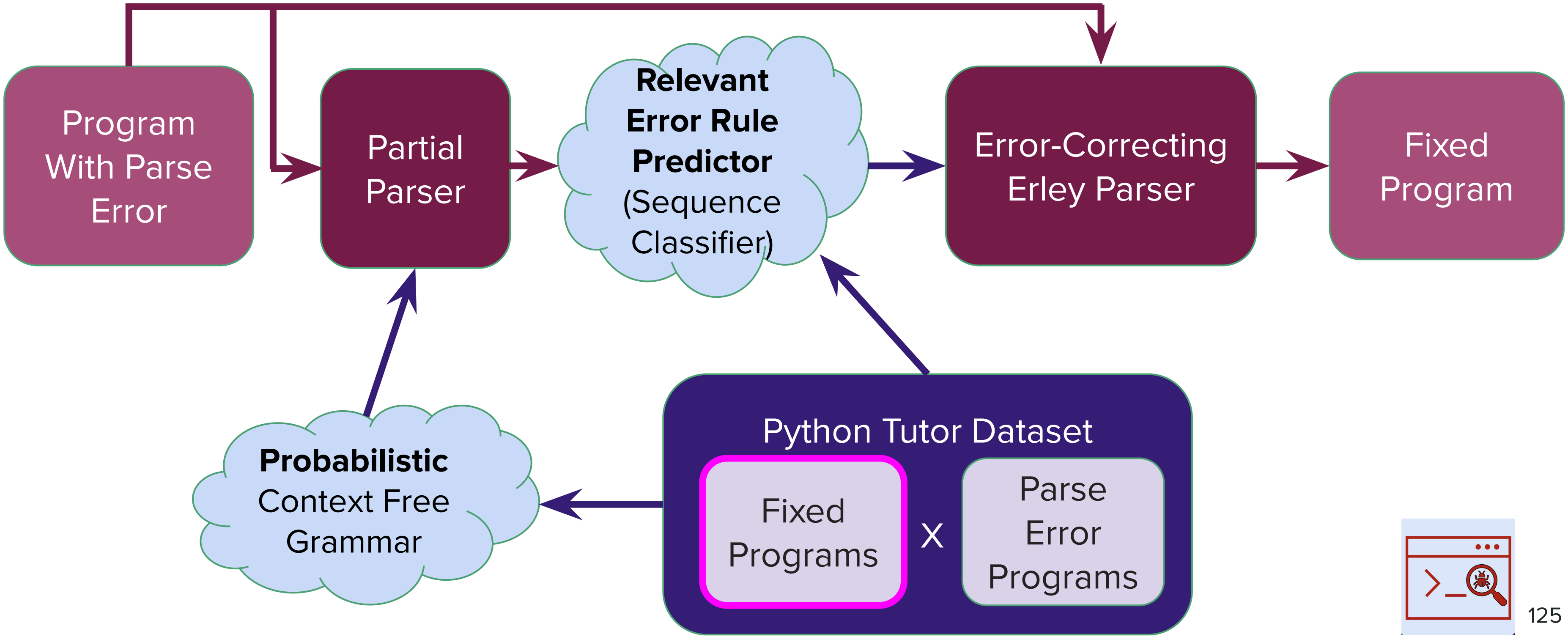
Solution: Learn a *Probabilistic Context Free Grammar* to Pick the Right One

S	$\rightarrow$	Stmts end_marker ($p = \underline{100.0\%}$)
Stmts	$\rightarrow$	Stmt \n ($p = \underline{38.77\%}$) | Stmt \n Stmts ($p = \underline{61.23\%}$)
Stmt	$\rightarrow$	ExprStmt ($p = \underline{62.64\%}$) | RetStmt ($p = \underline{7.59\%}$) | ...
RetStmt	$\rightarrow$	**return** ($p = \underline{1.61\%}$) | **return** Args ($p = \underline{98.39\%}$)

Seq2Parse: Efficient Fixes for Novice Parse Errors



Seq2Parse: Efficient Fixes for Novice Parse Errors



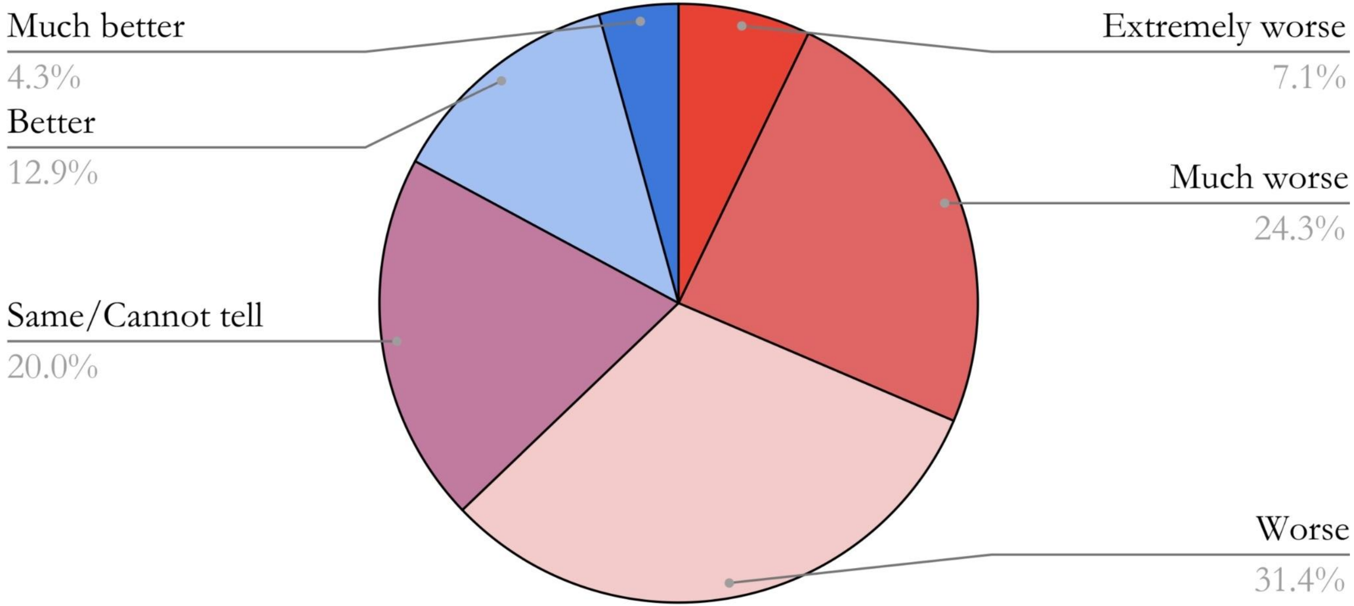
Cannabis sativa is the **world's most commonly used illicit substance**, used by more than 192 million people in 2018

Cannabis is used for many reasons both **medical** (e.g., pain relief) and **recreational** (e.g., altered consciousness)

126

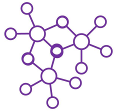
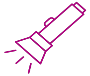
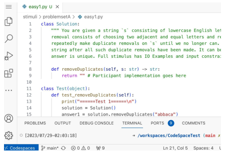
Cannabis's **legality is changing rapidly** with many countries (e.g., UK, Colombia, Canada, Malawi) recently taking **steps towards legalization**

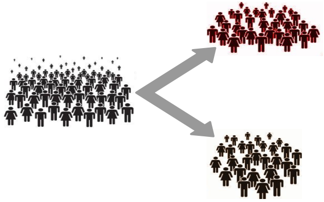
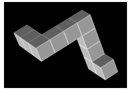
Self-reported subjective programming performance when high (compared to when sober)



My Approach To Programming Productivity: What's Next?

Desired Research Attribute	Why I'm Excited
Provide *Theoretically-Grounded* and *Actionable Insights*	Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact
Include *Empirical or Objective Measures* of Programmers	Captures aspects of programming beyond self-reporting alone, including unconscious behaviors and habits
Minimize Scientific Bias to Support Generalizability	Controlled experimental design can capture a signal, even for complex human behavior
Support *Diverse Developers*	I prefer approaches that not only help programmers in general, but also help those who need the most support

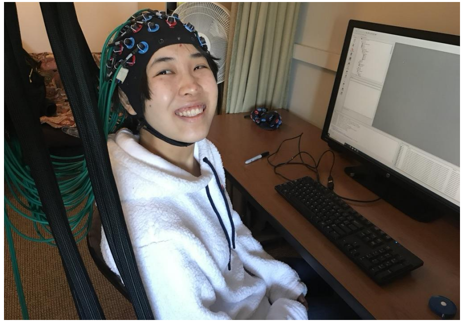





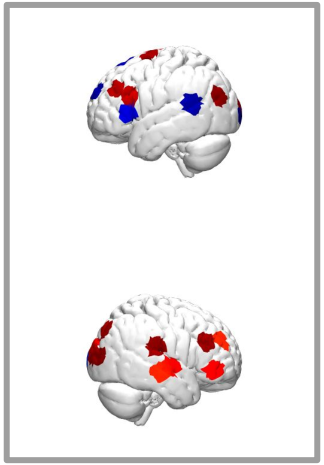



Objective measures of cognition

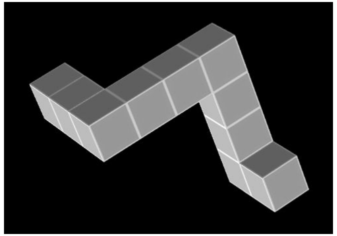
Models of Programming Cognition

Identify Relevant Cognitive Skills





Spatial Visualization



Technical Reading