ORACLE

Bug Fixes via Latent Program Sketches

Ben Rozonoyer

Labs East

August 5th, 2024

ORACLE

Bug Fixes via Latent Program Sketches

Ben Rozonoyer

Labs East

August 5th, 2024

ORACLE

Bug Fixes via Latent Program Sketches

BuFaLoS

Ben Rozonoyer

Labs East

August 5th, 2024

Why is Debugging Important?

This is what your developers are doing 75% of the time, and this is the cost you pay

(Assaraf, 2015)

- On average, a developer creates **70 bugs per 1000 lines** of code
- **15 bugs per 1000 lines** of code find their way to the customers
- **Fixing a bug takes 30x longer** than writing a line of code
- **75%** of a developer's time is spent on debugging
- **$113B** is spent annually on identifying and fixing product defects in US alone

Why is Debugging Important?

Reversible Debugging Software

(Britton et al., 2020)

- global cost of software development is **US$1.25 trillion / year**
- Wages-only estimated cost of debugging (50% time spent on debugging): **US$156 billion / year**

Why is Debugging Important?

An Exploratory Study of Debugging Episodes

(Alaboudi and LaToza, 2021)

- "…debugging was frequent, even in programming work, occurring **once every eight minutes.**"
- "…episodes vary greatly in time, with most being less than a few minutes and a few as **more than 100 minutes**."
- "…most debugging time is spent in **long debugging episodes**."

Why is Debugging Hard?

Why is Debugging Hard?

- Coding happens in teams — large, complex codebases with interlocking dependencies
  - You don’t know all the code, but still need to deduce original developer’s intent
  - Generating a bugfix can introduce new bugs

Why is Debugging Hard?

- Coding happens in teams — large, complex codebases with interlocking dependencies
  - You don’t know all the code, but still need to deduce original developer’s intent
  - Generating a bugfix can introduce new bugs

- Fixes should respect the buggy code
  - Dependencies
  - Naming conventions
  - Modularization & design

Why is Debugging Hard?

- Coding happens in teams — large, complex codebases with interlocking dependencies
  - You don’t know all the code, but still need to deduce original developer’s intent
  - Generating a bugfix can introduce new bugs

- Fixes should respect the buggy code
  - Dependencies
  - Naming conventions
  - Modularization & design

- Huge landscape of possible errors
  - Logical
  - Algorithmic
  - Syntactic
  - “Lack-of-thought”
  - Etc.

What Makes a Good Debugged Solution?

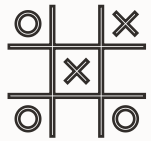
What Makes a Good Debugged Solution?

- Code should not be rewritten from scratch

What Makes a Good Debugged Solution?

- Code should not be rewritten from scratch

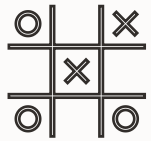
- Same approach and conventions should be followed as in buggy version
  - Algorithm idea (if correct)
  - Naming scheme
  - Modularization
  - Etc.



What Makes a Good Debugged Solution?

- Code should not be rewritten from scratch

- Same approach and conventions should be followed as in buggy version
  - Algorithm idea (if correct)
  - Naming scheme
  - Modularization
  - Etc.

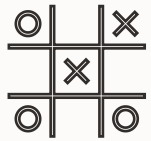


- Must fix bugs / improve functionality

What Makes a Good Debugged Solution?

- Code should not be rewritten from scratch

- Same approach and conventions should be followed as in buggy version
  - Algorithm idea (if correct)
  - Naming scheme
  - Modularization
  - Etc.



- Must fix bugs / improve functionality

- Should not be too slow

This Talk

In this talk, we develop a generative debugging approach which first tries to infer the developer's intent for the buggy code, resulting in increased transparency, and producing fixes that stay close to the original code.

Outline

- Debugging motivation
- Latent Program Sketches

How Are Programs Written?

How Are Programs Written?

Specification

Write a Python function
`flip_case(string: str) -> str`
to solve the following problem: For a given string, flip lowercase characters to uppercase and uppercase to lowercase.

```
>>> flip_case('Hello’)
'hELLO’
```

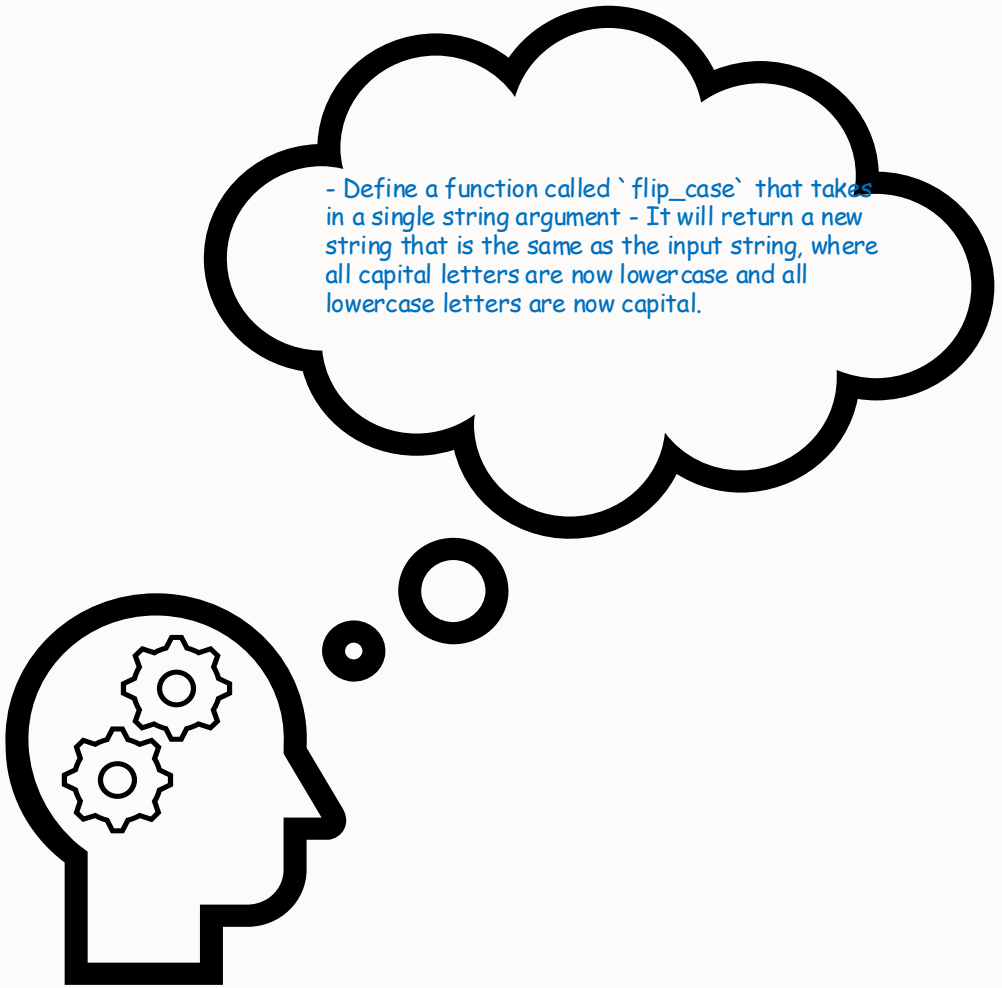
How Are Programs Written?

Specification

Write a Python function
`flip_case(string: str) -> str`
to solve the following problem: For a given string, flip lowercase characters to uppercase and uppercase to lowercase.

```
>>> flip_case('Hello')
'hELLO'
```

Sketch

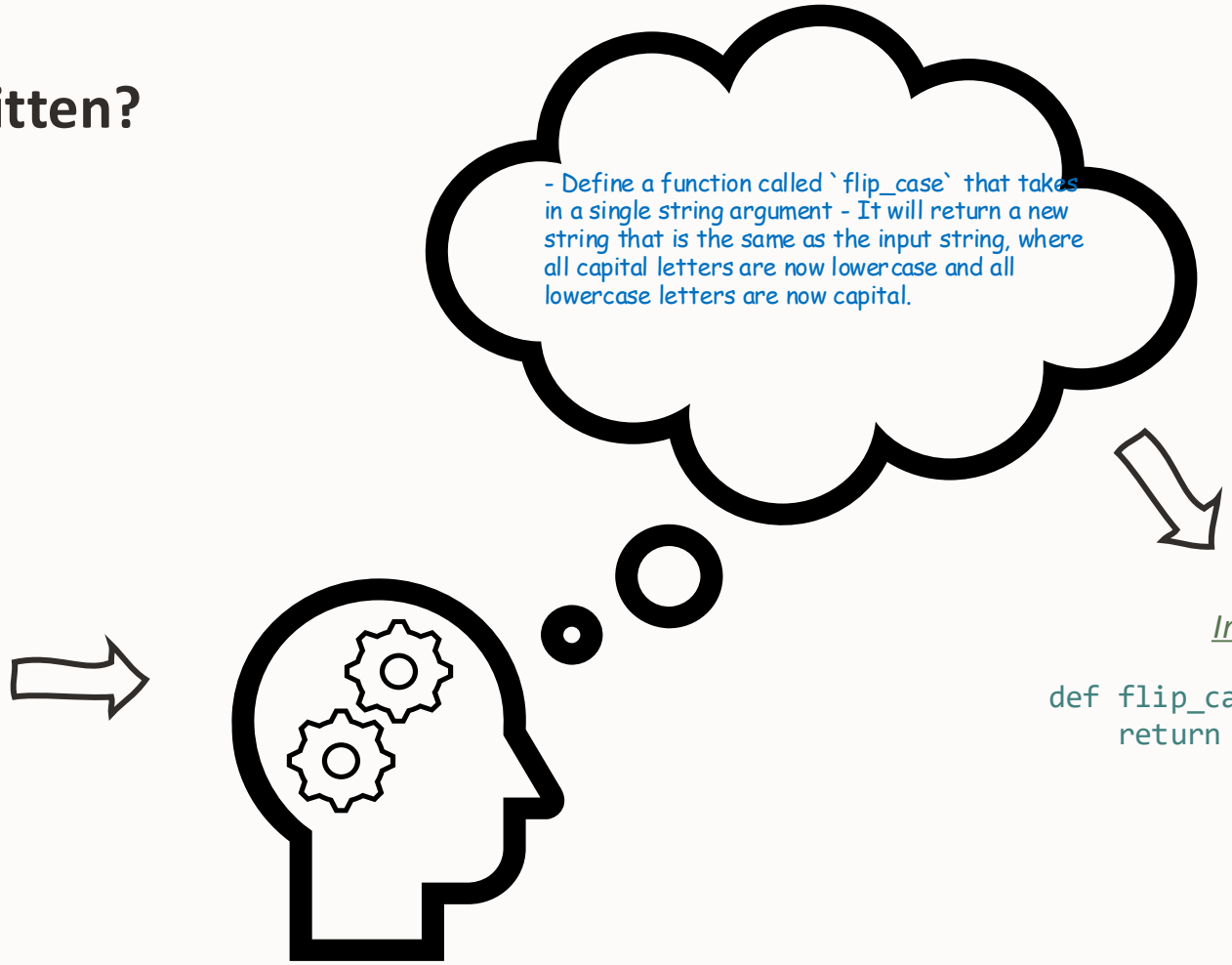


How Are Programs Written?

Specification

Write a Python function
`flip_case(string: str) -> str`
to solve the following problem: For a given string, flip lowercase characters to uppercase and uppercase to lowercase.

```
>>> flip_case('Hello')
'hELLO'
```



Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

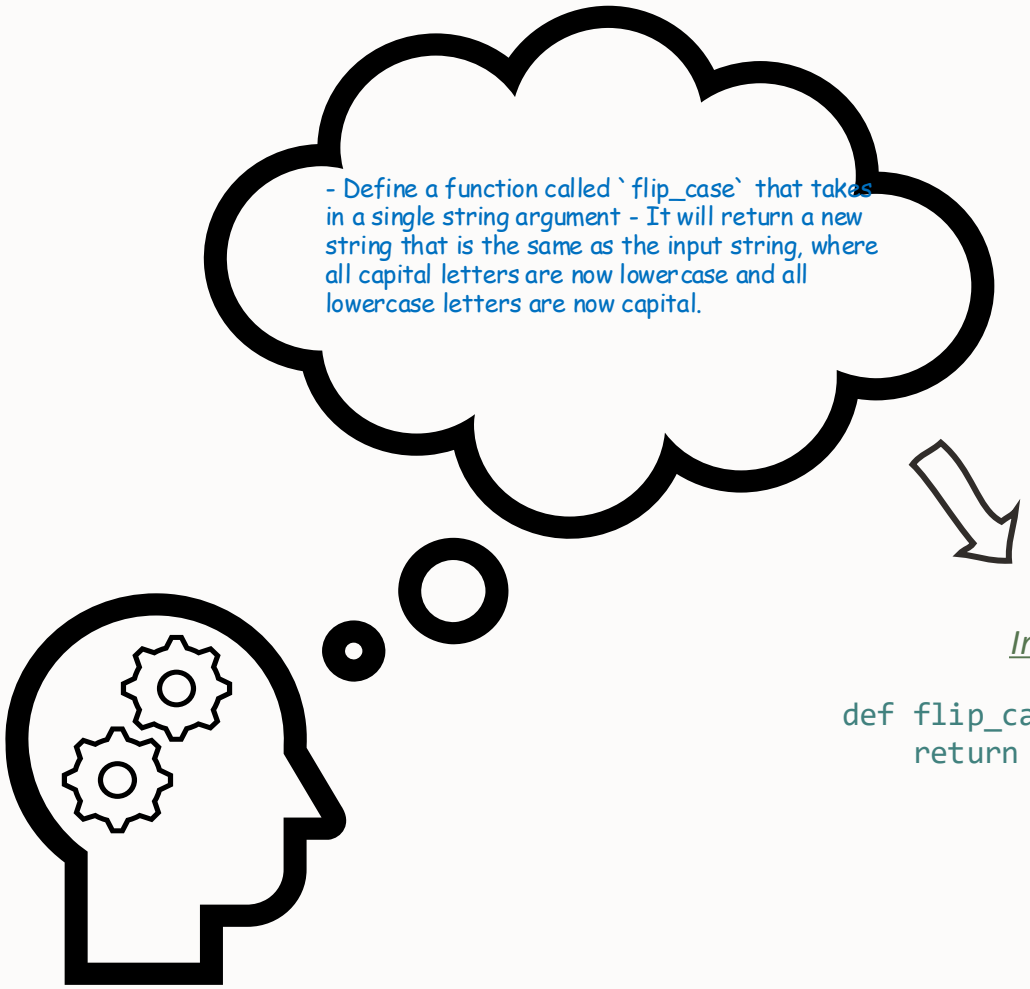
How Are Programs Written?

Specification

Write a Python function
`flip_case(string: str) -> str`
to solve the following problem: For a given string, flip lowercase characters to uppercase and uppercase to lowercase.

```
>>> flip_case('Hello’)
'hELLO’
```

Sketch



Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

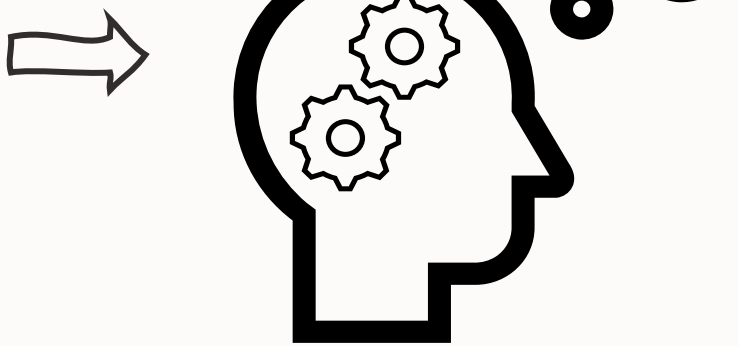
How Are Programs Written?

Specification

Write a Python function
`flip_case(string: str) -> str`
to solve the following problem: For a given string, flip lowercase characters to uppercase and uppercase to lowercase.

```
>>> flip_case('Hello')
'hELLO'
```

I



Sketch

Z

- Define a function called `flip_case` that takes in a single string argument - It will return a new string that is the same as the input string, where all capital letters are now lowercase and all lowercase letters are now capital.

Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

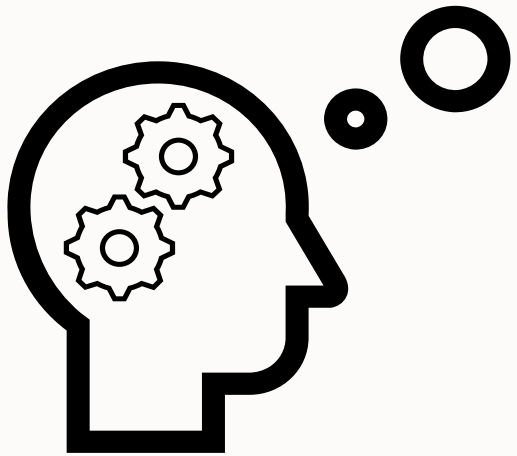
How Are Programs Written?

Specification

Write a Python function
`flip_case(string: str) -> str`
to solve the following problem: For a given string, flip lowercase characters to uppercase and uppercase to lowercase.

```
>>> flip_case('Hello')
'hELLO'
```

I



Sketch

Z

- Define a function called `flip_case` that takes in a single string argument - It will return a new string that is the same as the input string, where all capital letters are now lowercase and all lowercase letters are now capital.

Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

P

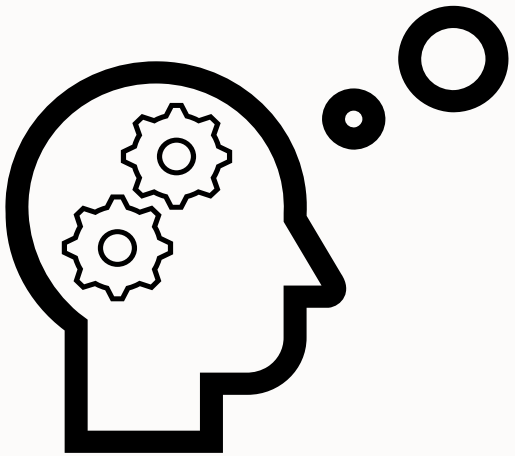
How Are Programs Written?

Specification

Write a Python function
`flip_case(string: str) -> str`
to solve the following problem: For a given string, flip lowercase characters to uppercase and uppercase to lowercase.

```
>>> flip_case('Hello')
'hELLO'
```

I



Sketch

Z

- Define a function called `flip_case` that takes in a single string argument - It will return a new string that is the same as the input string, where all capital letters are now lowercase and all lowercase letters are now capital.

Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

$\overline{P}$

How Are Programs Written?

Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

How Are Programs Written?

Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

How Are Programs Written?

Sketch

- Define a function called `flip_case` that takes in a single string argument
- It will return a new string that is the same as the input string, where all capital letters are now lowercase and all lowercase letters are now capital.

Z

Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

$\overline{P}$

How Are Programs Written?

Sketch

- Define a function called `flip_case` that takes in a single string argument
- It will return a new string that is the same as the input string, where all capital letters are now lowercase and all lowercase letters are now capital.

Z

Implementation

```
def flip_case(string: str) -> str:
    return string.lower()
```

$\overline{P}$

Bugfix

```
def flip_case(string: str) -> str:
    res = ""
    for c in string:
        if c.islower():
            res += c.upper()
        else:
            res += c.lower()
    return res
```

P

What Are the Advantages of Working with Sketches?

What Are the Advantages of Working with Sketches?

- Sketches as byproduct of debugging — independently, this is useful to developer
  - **Easier to eyeball and filter** for correctness/complexity, use as human-readable “certificate”

What Are the Advantages of Working with Sketches?

- Sketches as byproduct of debugging — independently, this is useful to developer
  - **Easier to eyeball and filter** for correctness/complexity, use as human-readable “certificate”

- If this machinery works, will yield **solutions closer to original** buggy code than if we just try to generate solution

What Are the Advantages of Working with Sketches?

- Sketches as byproduct of debugging — independently, this is useful to developer
  - **Easier to eyeball and filter** for correctness/complexity, use as human-readable “certificate”

- If this machinery works, will yield **solutions closer to original** buggy code than if we just try to generate solution

- ≅ **Chain-of-Thought Prompting Elicits Reasoning in LLMs** (Wei et al., 2022)
  - There: taking word problem and working through how it should be solved in English
  - Here: learning to reason about code in latent sketch space

What Are the Advantages of Working with Sketches?

- Sketches as byproduct of debugging — independently, this is useful to developer
  - **Easier to eyeball and filter** for correctness/complexity, use as human-readable “certificate”

- If this machinery works, will yield **solutions closer to original** buggy code than if we just try to generate solution

- ≅ **Chain-of-Thought Prompting Elicits Reasoning in LLMs** (Wei et al., 2022)
  - There: taking word problem and working through how it should be solved in English
  - Here: learning to reason about code in latent sketch space

- **: : Classical program sketching**
  - Cf. *Neural Sketch Learning for Conditional Program Generation* (Murali et al., 2018)

Outline

- Debugging motivation
- Latent Program Sketches
- Method

Method

- **Recall:** given instructions I and buggy program $\bar{P}$
  - want fix P
  - want P to be close to $\bar{P}$

Method

- **Recall:** given instructions I and buggy program $\bar{P}$
  - want fix P
  - want P to be close to $\bar{P}$

```
Algorithm BootstrapSketches (dev, test):
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```

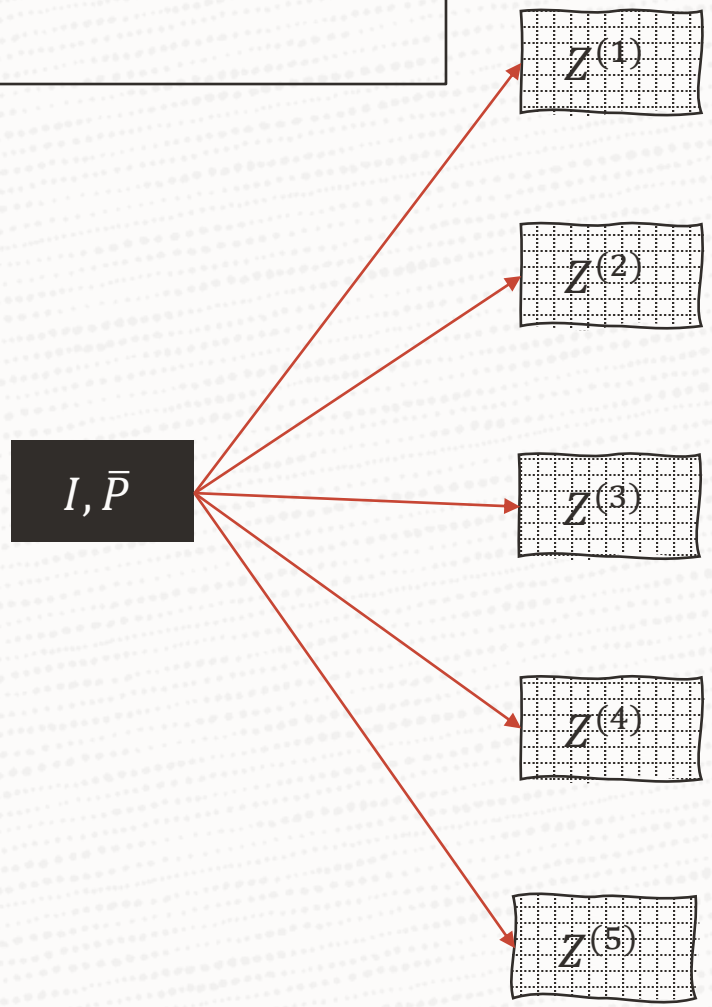
Algorithm BootstrapSketches (dev, test):

```
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```

$$I, \bar{P}$$

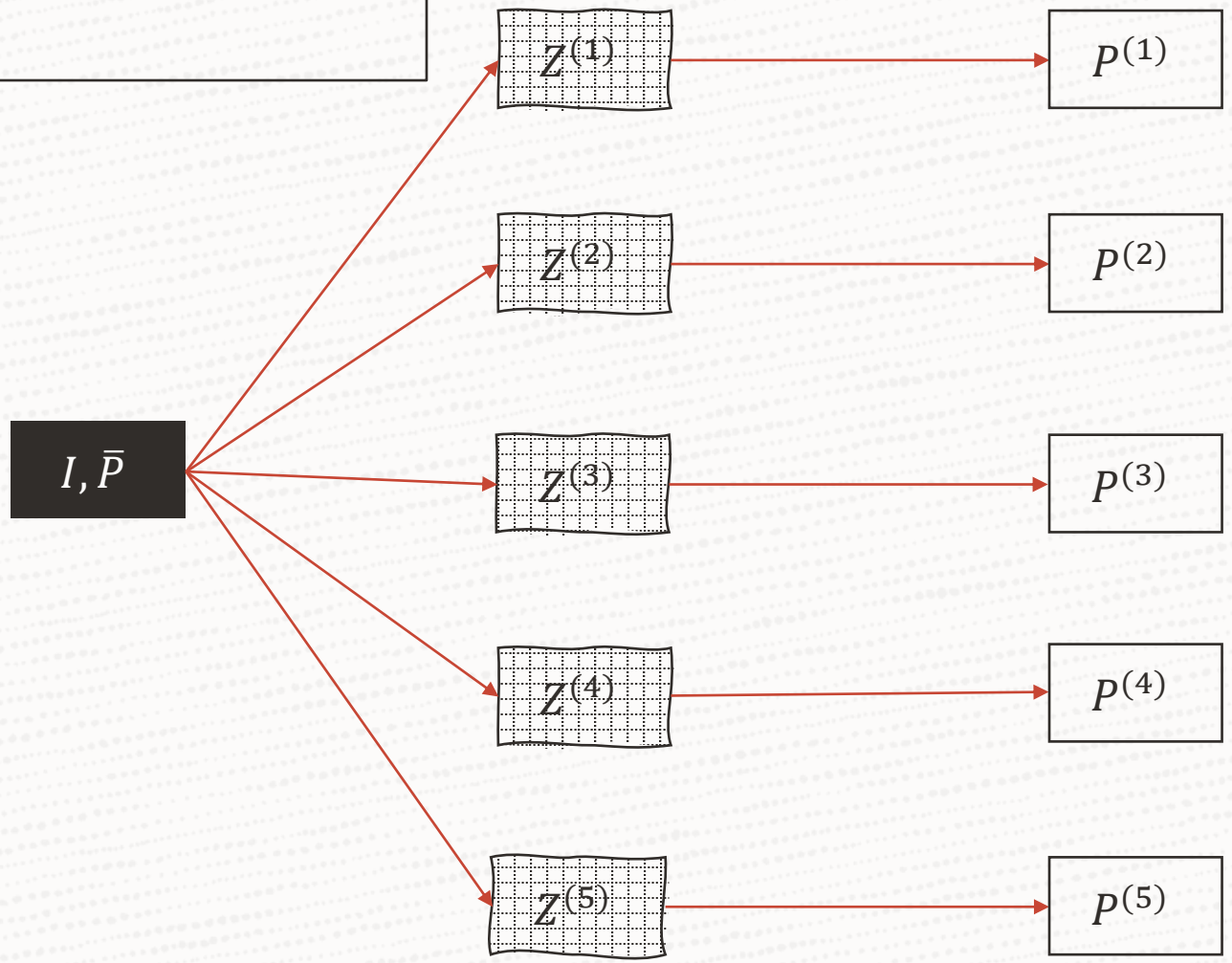
Algorithm BootstrapSketches (dev, test):

```
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```



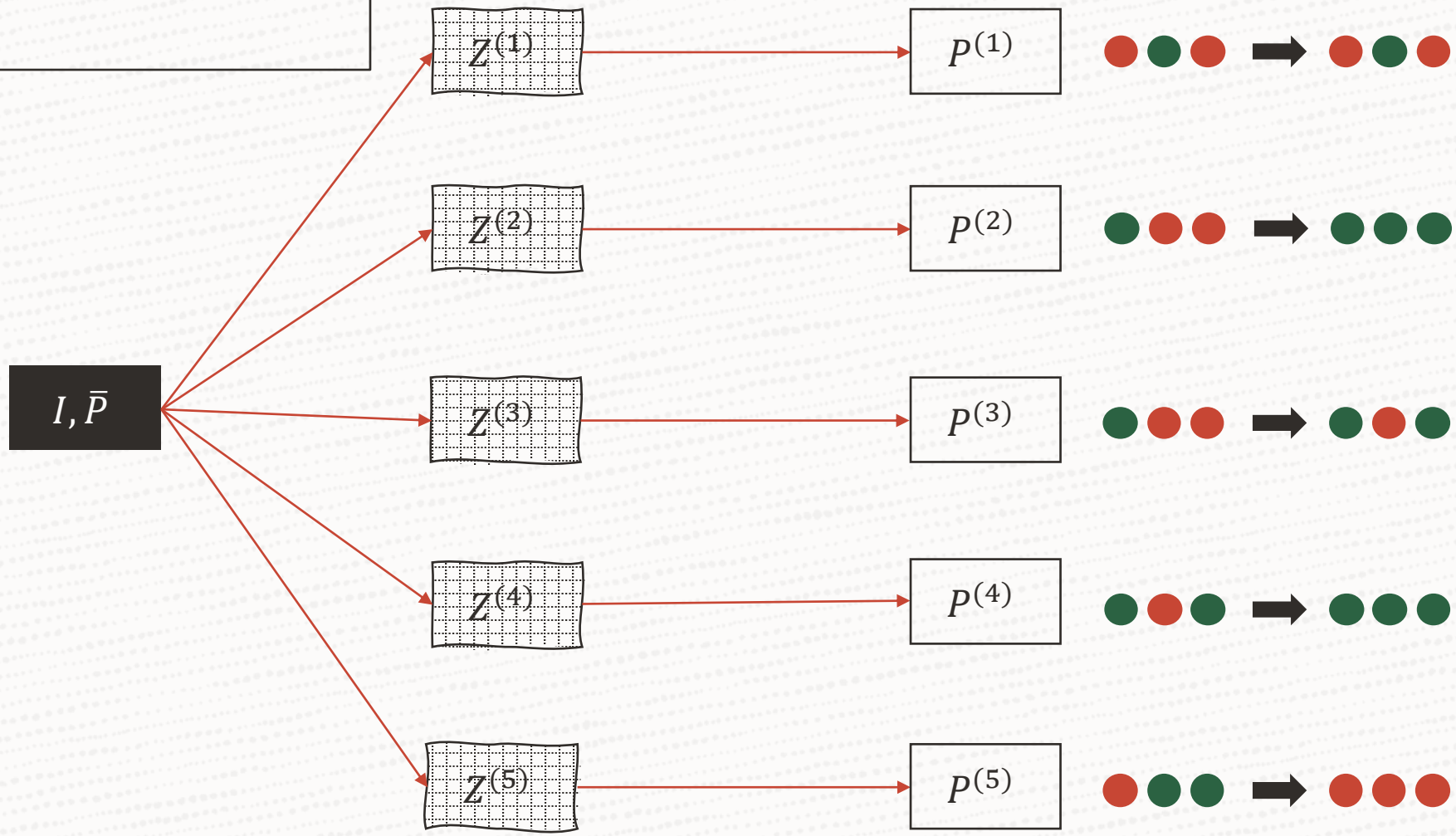
Algorithm BootstrapSketches (dev, test):

```
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```



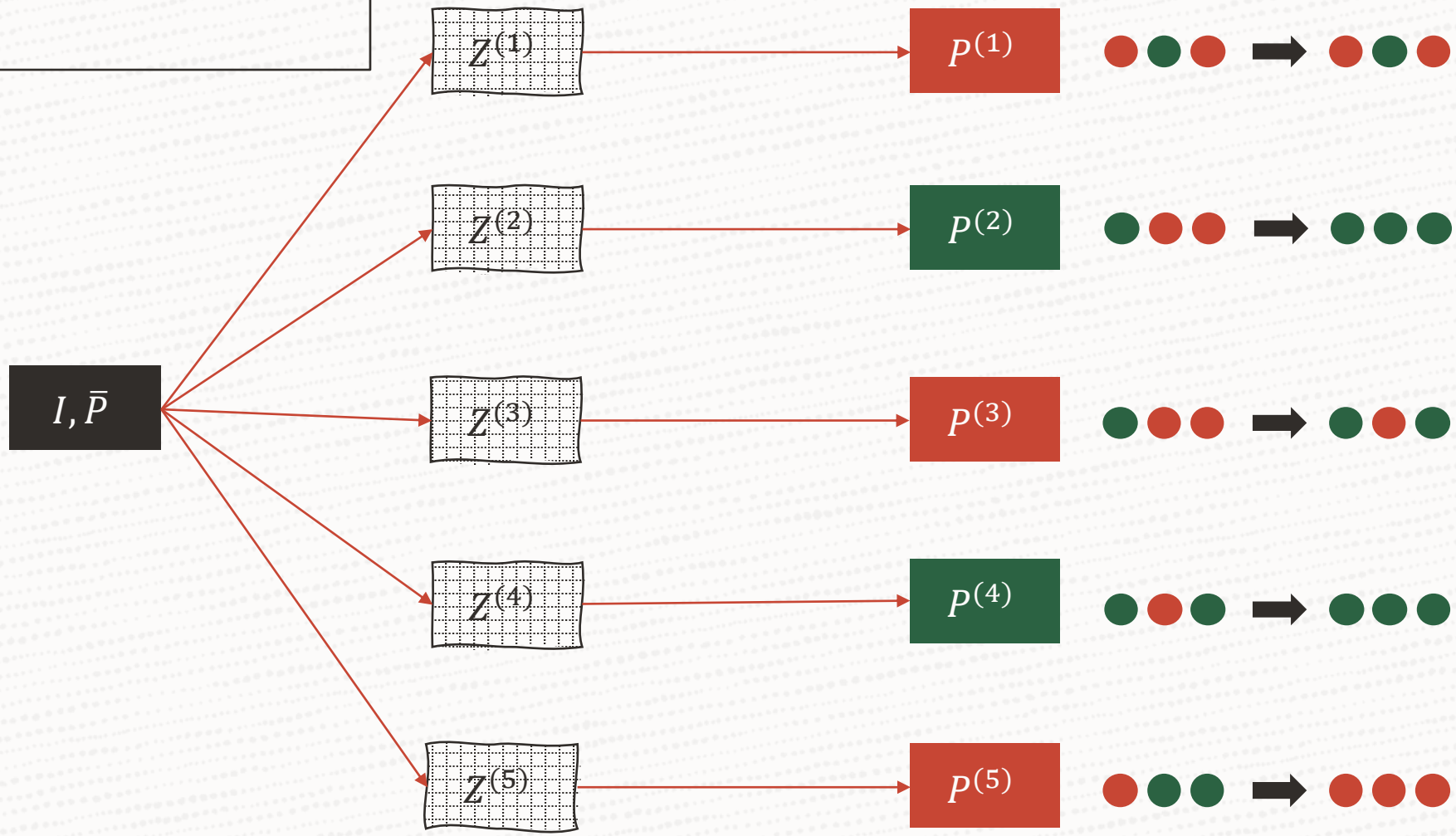
Algorithm BootstrapSketches (dev, test):

```
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```



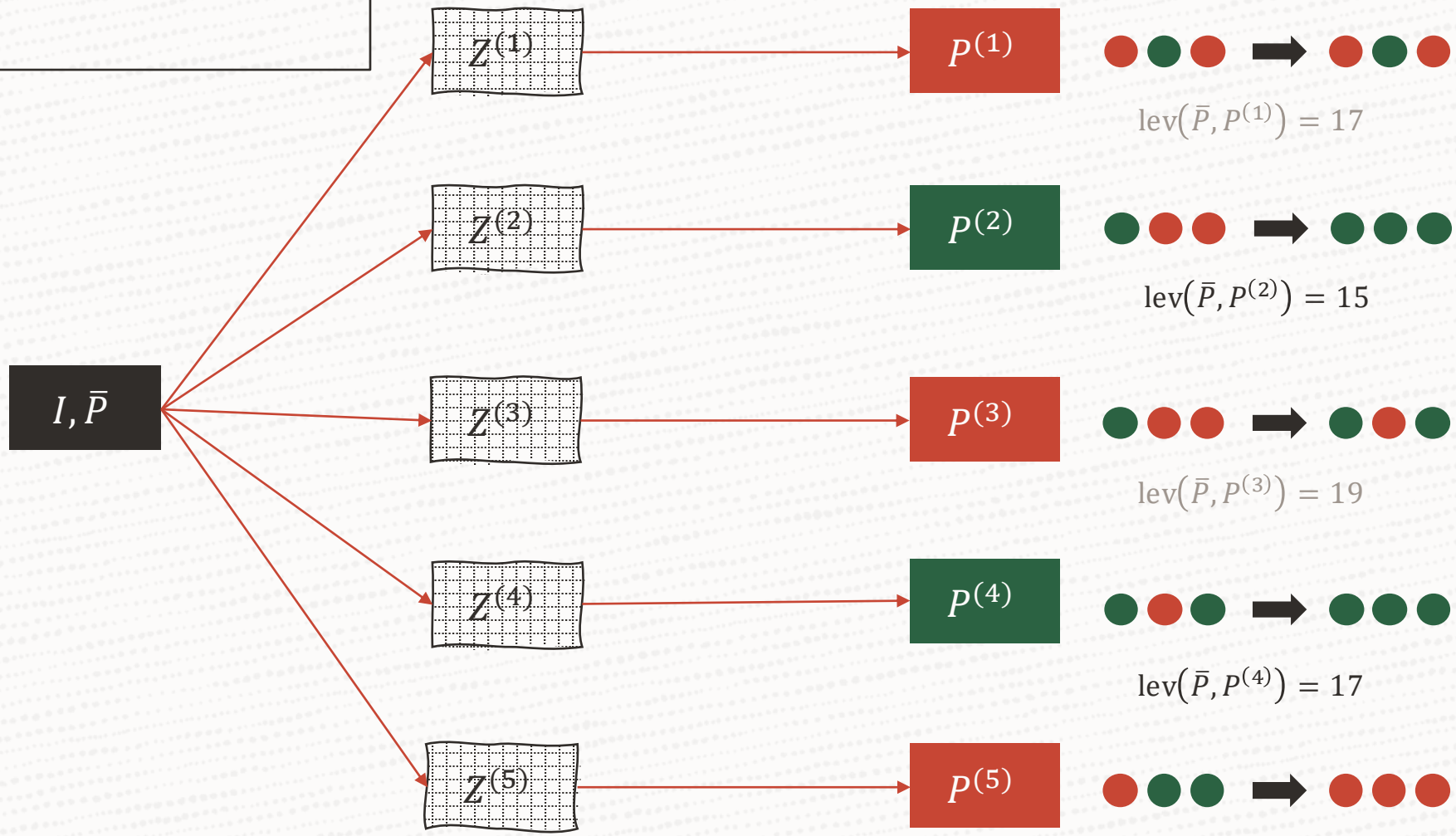
Algorithm BootstrapSketches (dev, test):

```
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```



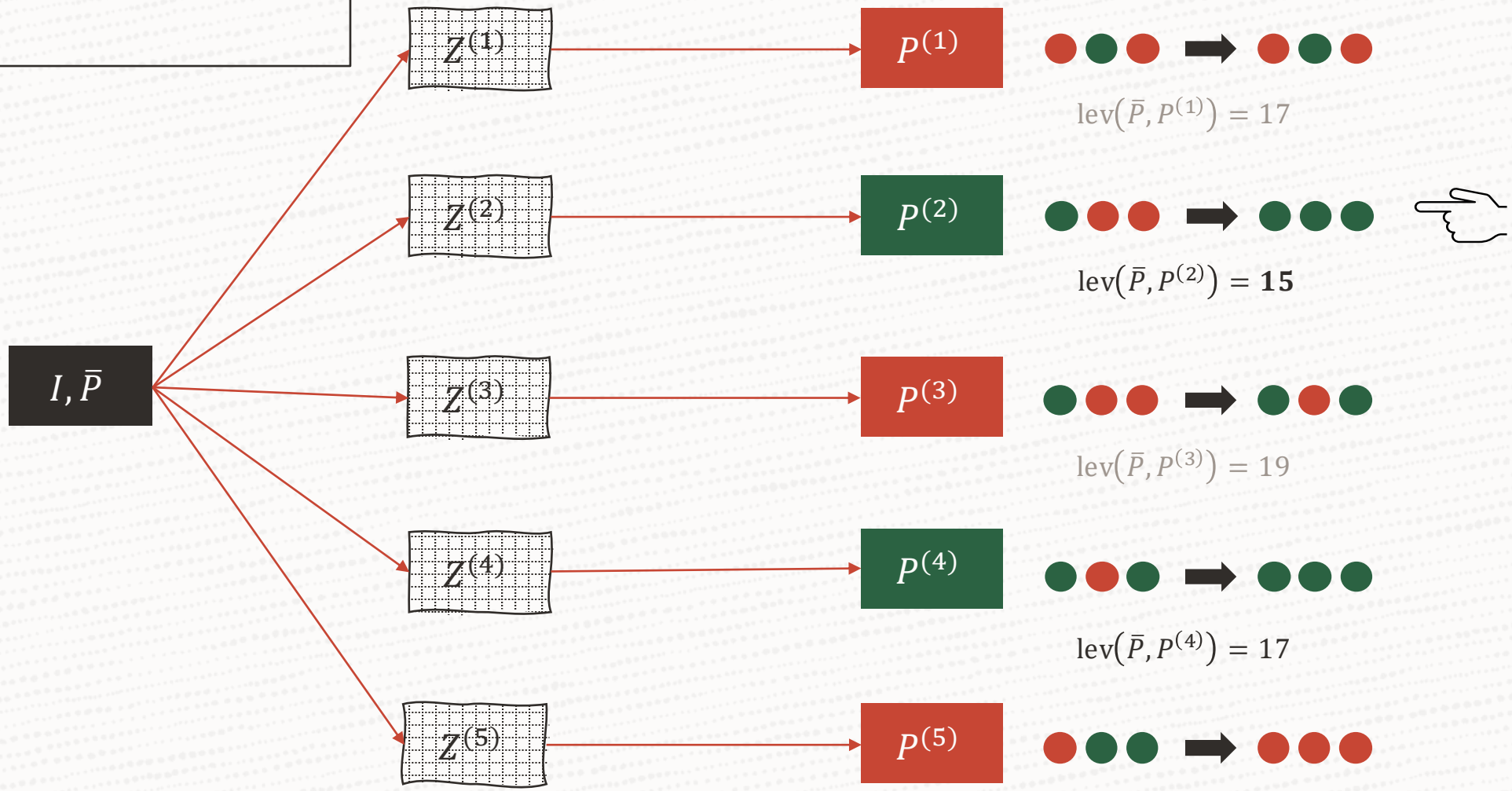
Algorithm BootstrapSketches (dev, test):

```
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```



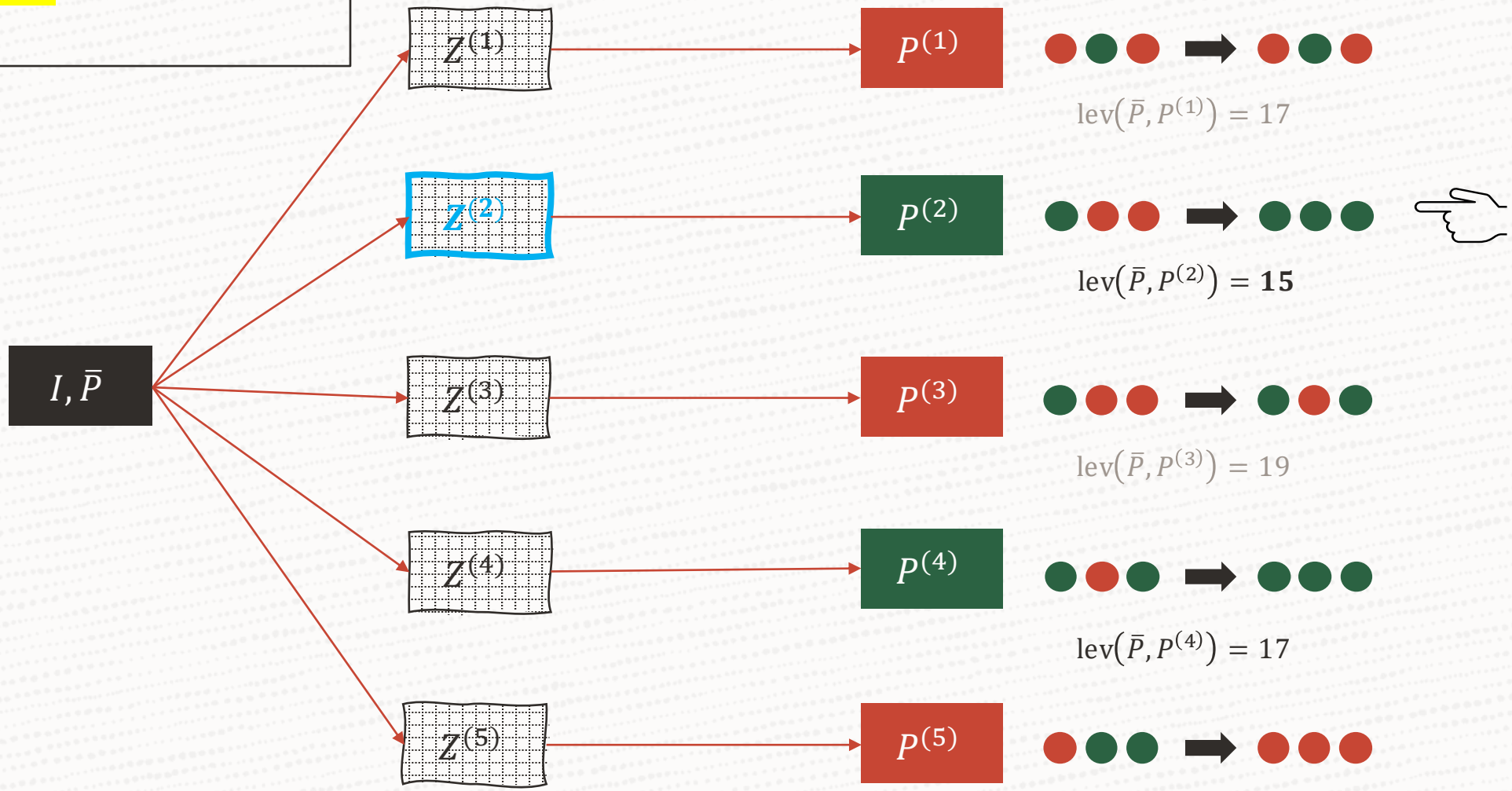
Algorithm BootstrapSketches (dev, test):

```
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```



Algorithm BootstrapSketches (dev, test):

```
// at dev time
1. for each dev example:
2.     GENERATE collection of sketches
3.     for each sketch:
4.         GENERATE a bug fix
5.     FILTER those sketches for the ones that lead to fixes
6.     SELECT the closest fix to the original
7.     SAVE corresponding successful sketch
// at test time
8. use successful sketches for ICL
```



Outline

- Debugging motivation
- Latent Program Sketches
- Method
- Experiments

Experimental Setup

HumanEvalFix

Experimental Setup

HumanEvalFix

- *OctoPack: Instruction Tuning Code Large Language Models* (Muenninghoff et al. 2024)

Experimental Setup

HumanEvalFix

- *OctoPack: Instruction Tuning Code Large Language Models* (Muenninghoff et al. 2024)

- 164 problems, created random split
  - 60 dev
  - 100 test *(omitted 4 problems that posed evaluation difficulties using Python exec)*


```
{
  "task_id": "Python/0",
  "prompt": "from typing import List\n\n\ndef has_close_elements(numbers: List[float], thresh
  "declaration": "from typing import List\n\n\ndef has_close_elements(numbers: List[float], t
  "canonical_solution": "    for idx, elem in enumerate(numbers):\n        for idx2, elem2 in
  "buggy_solution": "    for idx, elem in enumerate(numbers):\n        for idx2, elem2 in enu
  "bug_type": "missing logic",
  "failure_symptoms": "incorrect output",
  "entry_point": "has_close_elements",
  "import": ""
  "test_setup": ""
  "test": "\n\n\n\n\ndef check(has_close_elements):\n    assert has_close_elements([1.0, 2.0,
  "example_test": "def check(has_close_elements):\n    assert has_close_elements([1.0, 2.0, 3
  "signature": "has_close_elements(numbers: List[float], threshold: float) -> bool",
  "docstring": "Check if in given list of numbers, are any two numbers closer to each other t
  "instruction": "Write a Python function `has_close_elements(numbers: List[float], threshold
}
```

Experimental Setup
Evaluation

Experimental Setup

Evaluation

- Generated code gets run through unit tests

Experimental Setup

Evaluation

- Generated code gets run through unit tests

- If bug fix passes *all* unit tests, it is counted as correct 
  - (We do not consider bug fixes that improve unit tests but do not pass all of them) 

Experimental Setup

Evaluation

- Generated code gets run through unit tests
- If bug fix passes *all* unit tests, it is counted as correct

  - (We do not consider bug fixes that improve unit tests but do not pass all of them)

- Use **Pass @ k** from Codex paper

$$\text{pass@}k := \mathop{\mathbb{E}}_{\text{Problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

```
def pass_at_k(n, c, k):
    """
    :param n: total number of samples
    :param c: number of correct samples
    :param k: k in pass@$k$
    """
    if n - c < k: return 1.0
    return 1.0 - np.prod(1.0 - k /
        np.arange(n - c + 1, n + 1))
```

Figure 3. A numerically stable script for calculating an unbiased estimate of pass@k.

* *Evaluating Large Language Models Trained on Code (Chen et al. 2021)*

Experiments

Experiments

- **Baselines**
  - $I \rightarrow P$ simplest thing is to generate bug fix from scratch (throw out buggy code)
  - $(I, \bar{P}) \rightarrow P$ don't go into latent space at all

Experiments

- **Baselines**
  - $I \rightarrow P$ simplest thing is to generate bug fix from scratch (throw out buggy code)
  - $(I, \bar{P}) \rightarrow P$ don't go into latent space at all

- **Design choices involving sketches**
  - How to generate sketches Z^* using successful sketches in ICL
  - How to generate bug fixes P

Experiments

- **Baselines**
  - $I \rightarrow P$ simplest thing is to generate bug fix from scratch (throw out buggy code)
  - $(I, \bar{P}) \rightarrow P$ don't go into latent space at all

- **Design choices involving sketches**
  - How to generate sketches Z^* using successful sketches in ICL
  - How to generate bug fixes P

- **Approaches**
  - $(I, \bar{P}) \rightarrow Z^*$ instructions + buggy code to sketches
  - $Z^* \rightarrow P$ sketch to bug fix
  - $(I, \bar{P}) \rightarrow (Z^*, P)$ instructions + buggy code to bug fix + sketch
  - $(I, \bar{P}, (I, \bar{P}) \rightarrow Z^*) \rightarrow P$ instruction + buggy code + sketch to bug fix

- beam search
- top-k sampling
- greedy decoding

CodeLlama-34b-Instruct

#	decoding	data flow	Pass@1	Pass@2	Pass@5	Pass@10
1		$I \rightarrow P$	56.09	58.37	60.30	61
2		$(I, \bar{P}) \rightarrow P$	**66.79**	**69.68**	**72.94**	**75**
3	,	$(I, \bar{P}) \rightarrow \tilde{Z} \rightarrow P$	5.81	9.59	17.20	22
4	,	$(I, \bar{P}) \rightarrow Z^* \rightarrow P$	9.44	15.64	26.85	34
5		$(I, \bar{P}) \rightarrow (Z^*, P)$	55.39	58.97	63.56	66
6	,	$(I, \bar{P}, (I, \bar{P}) \rightarrow Z^*) \rightarrow P$	34.44	40.96	50.08	56.56
7	,	$(I, \bar{P}, (I, \bar{P}) \rightarrow Z^*) \rightarrow P$	**51.15**	**62.55**	**72.94**	**78**

† 6 out of 60 successful “sketches” obtained from dev set

- beam search
- top-k sampling
- greedy decoding

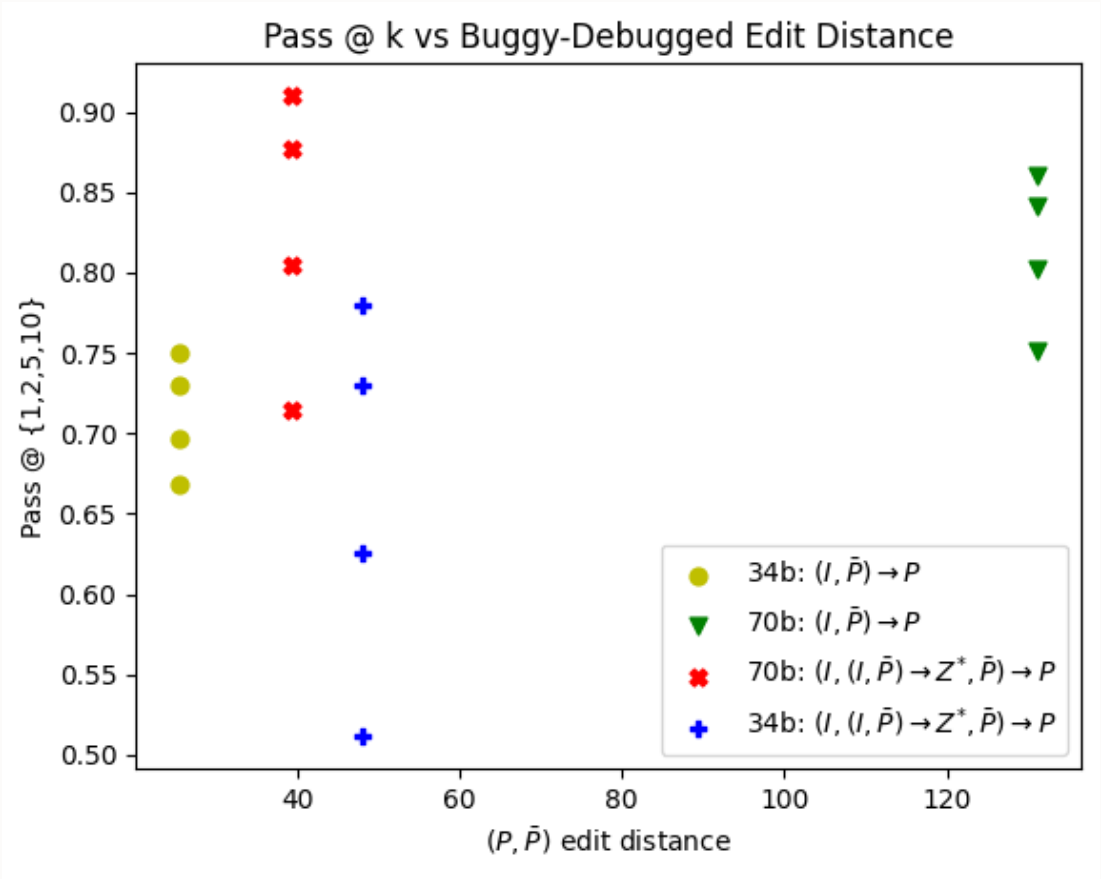
CodeLlama-70b-Instruct

#	decoding	data flow	Pass@1	Pass@2	Pass@5	Pass@10
1		$I \rightarrow P$	20.99	27.71	36.06	41
2		$(I, \bar{P}) \rightarrow P$	**75.09**	**80.19**	**84.17**	**86**
3	,	$(I, \bar{P}) \rightarrow \tilde{Z} \rightarrow P$	7.17	12.80	25.29	39
4	,	$(I, \bar{P}) \rightarrow Z^* \rightarrow P$	23.44	37.09	56.10	66
5		$(I, \bar{P}) \rightarrow (Z^*, P)$	50.2	52.97	55.19	57
6	,	$(I, \bar{P}, (I, \bar{P}) \rightarrow Z^*) \rightarrow P$	30.58	43.25	56.59	65.11
7	,	$(I, \bar{P}, (I, \bar{P}) \rightarrow Z^*) \rightarrow P$	**71.40**	**80.50**	**87.71**	**91**

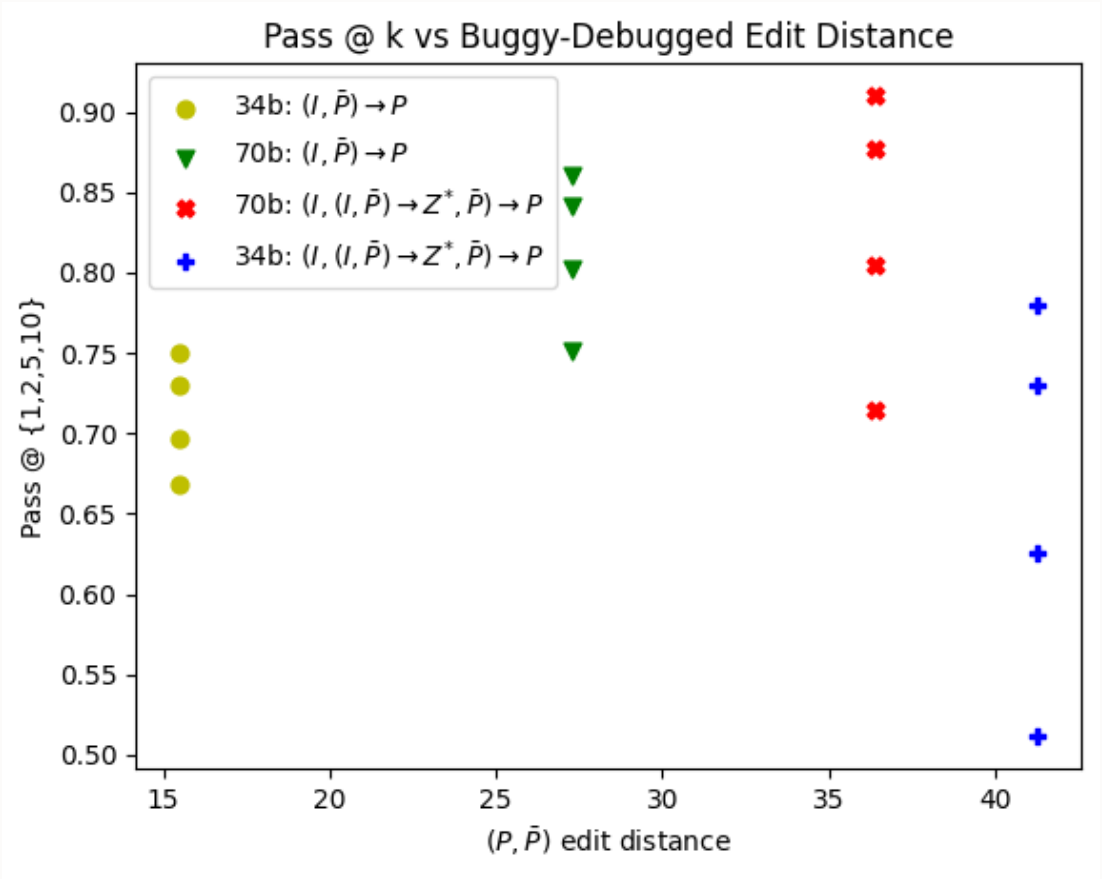
† 14 out of 60 successful “sketches” obtained from dev set

* At least 8 examples from baseline overgenerated and weren’t caught by regex extractor — baseline results skewed downwards

Are Fixes from BuFaLoS  Close to the Original Code?

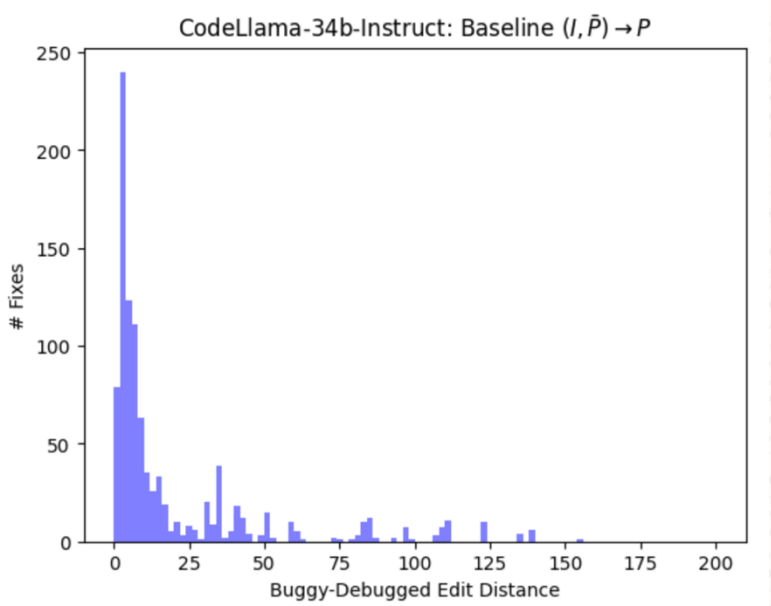


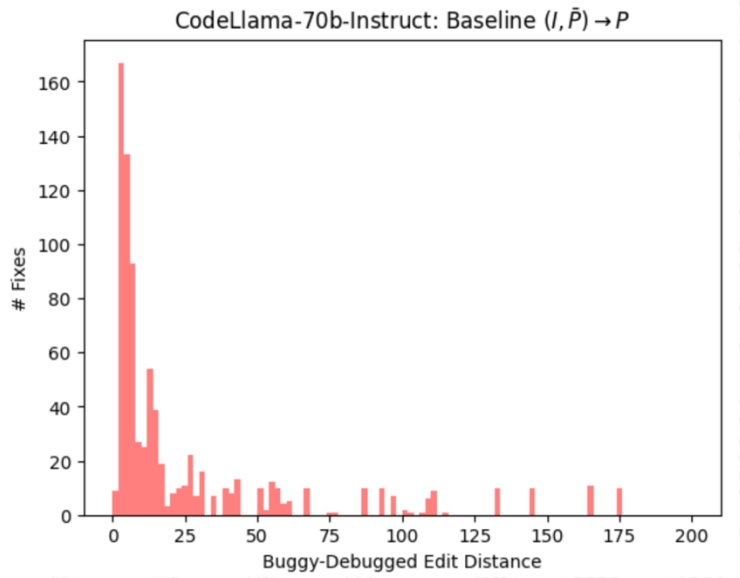
All bugfixes

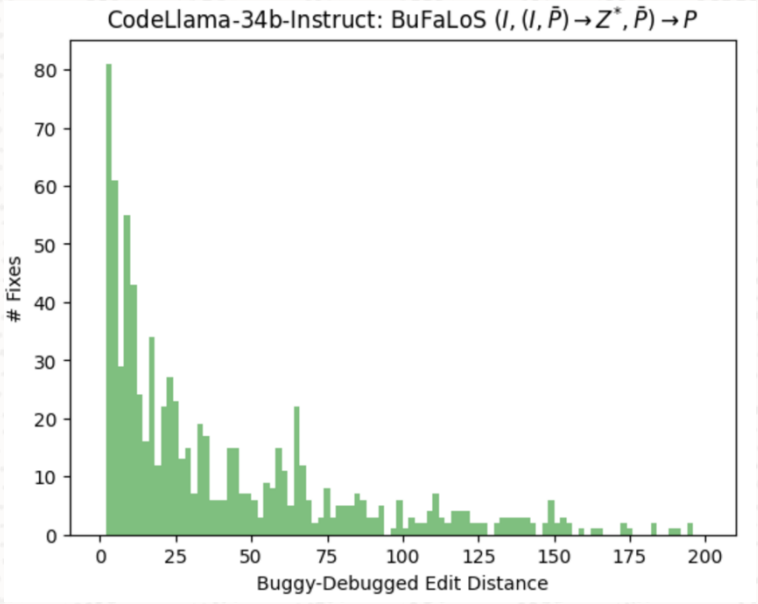


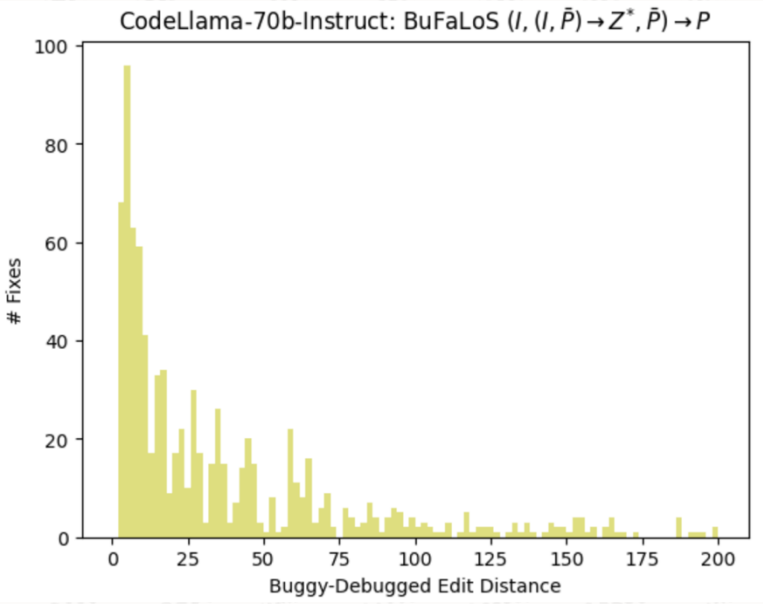
Only bugfixes that fixed all unit tests

Buggy-debugged edit distance distributions: Baselines vs BuFaLoS









Successful sketch used in BuFaLoS 70b bootstrapping

Python/71

Write a Python function `triangle_area(a, b, c)` to solve the following problem:
Given the lengths of the three sides of a triangle. Return the area of the triangle rounded to 2 decimal points if the three sides form a valid triangle.
Otherwise return -1
Three sides make a valid triangle when the sum of any two sides is greater than the third side.
Example:
triangle_area(3, 4, 5) == 6.00
triangle_area(1, 2, 10) == -1

```
def triangle_area(a, b, c):
    if a + b <= c or a + c <= b or b + c <= a:
        return -1
    s = (a + b + c)
    area = (s * (s - a) * (s - b) * (s - c)) ** 0.5
    area = round(area, 2)
    return
```

- Check if the three given sides satisfy the **triangle inequality** theorem.
- If they do, compute the area of the triangle using **Heron's formula**.
- If there is an edge case (for example, if b + c is less than or equal to a), return -1.
- Rounding errors in Python occur because float(3) is represented as 2.9999999999999995, not 3.
- However, using the round() function corrects for this.
- To round to 2 decimal places, use the round() function, e.g. round(6.3333333333333, 2) -> 6.33 .

```
def triangle_area(a, b, c):
    if a + b <= c or a + c <= b or b + c <= a:
        return -1
    s = (a + b + c) / 2
    area = (s * (s - a) * (s - b) * (s - c)) ** 0.5
    return round(area, 2)
```

Example extracted by BuFaLoS 70b but not by Baseline 70b

Python/125

Write a Python function `split_words(txt)` to solve the following problem:
Given a string of words, return a list of words split on whitespace, if no whitespaces exists in the text you should split on commas ',' if no commas exists you should return the number of lower-case letters with odd order in the alphabet, ord('a') = 0, ord('b') = 1, ... ord('z') = 25
Examples
split_words("Hello world!") ➞ ["Hello", "world!"]
split_words("Hello,world!") ➞ ["Hello", "world!"]
split_words("abcdef") == 3

```
def split_words(txt):
    if " " in txt:
        return txt.split()
    elif "," in txt:
        return txt.replace(' ',',').split()
    else:
        return len([i for i in txt if i.islower() and ord(i)%2 == 0])
```

- Check if the input string txt contains any white space characters.
  - If so, return a list of words split on each occurrence of a white space character.
- Otherwise, check if the input string txt contains any commas.
  - If so, replace all commas with white space characters, and then return a list of words split on each occurrence of a white space character.
- Otherwise, count the number of lower case letters in the input string txt whose ordinal ASCII values are even.
  - Return this count.

```
def split_words(txt):
    if " " in txt:
        return txt.split()
    elif "," in txt:
        return txt.replace(' ',',').split()
    else:
        return len([i for i in txt if i.islower() and ord(i)%2 != 0])
```

```
def split_words(txt):
    if " " in txt:
        return txt.split()
    elif "," in txt:
        return txt.replace(',', ' ').split()
    else:
        return len([i for i in txt if i.islower() and ord(i) % 2 == 0])
```

Outline

- Debugging motivation
- Latent Program Sketches
- Method
- Experiments
- Limitations and Future Work

Limitations

Limitations

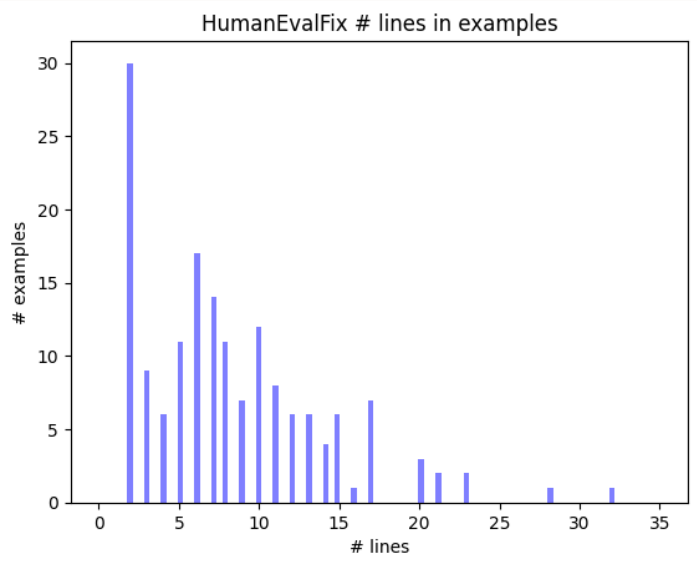
- HumanEvalFix is easy, baselines are too good

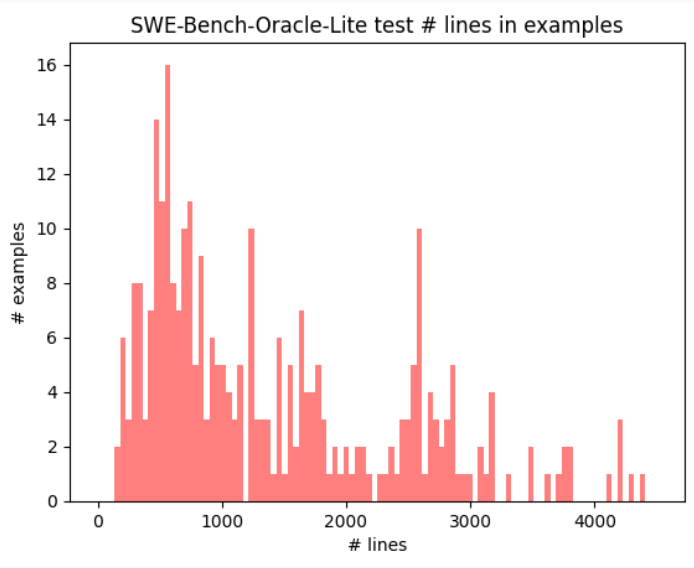
Limitations

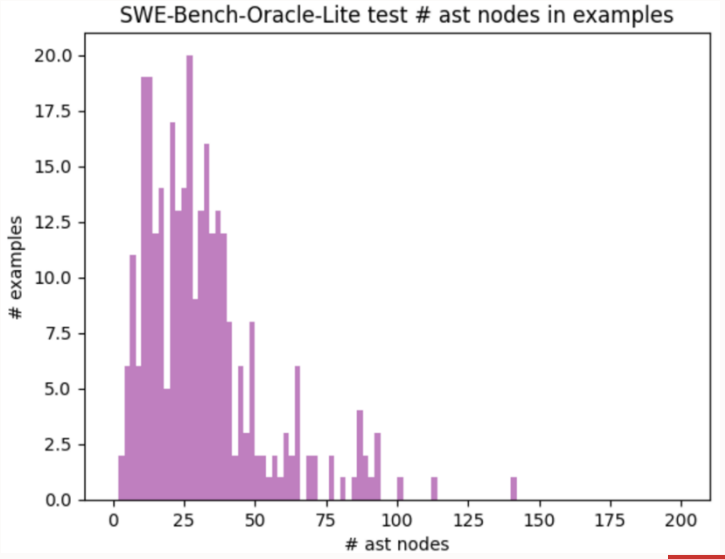
- HumanEvalFix is easy, baselines are too good
- HumanEvalFix is short — no need to localize bug

Limitations

- HumanEvalFix is easy, baselines are too good
- HumanEvalFix is short — no need to localize bug







Approach: Localization

Approach: Localization

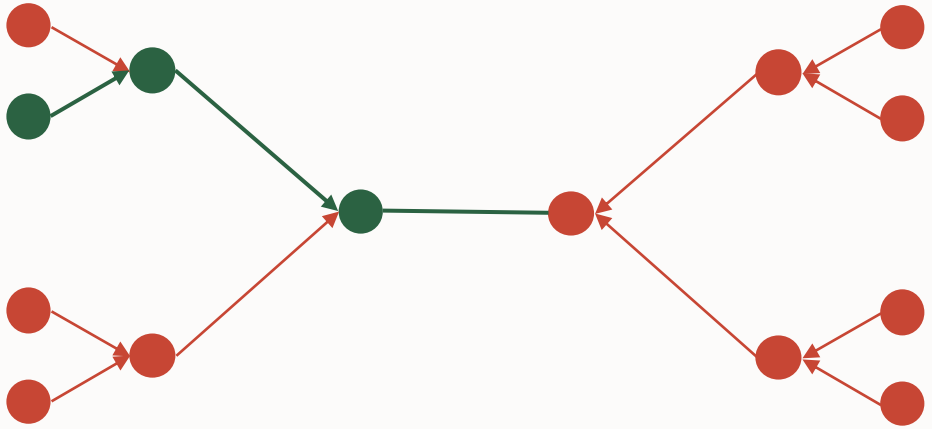
- Instead of directly debugging a long file, **pinpoint location of bug** and then intervene on it locally

Approach: Localization

- Instead of directly debugging a long file, pinpoint location of bug and then intervene on it locally
- **Decompose** a buggy piece of code into smaller components

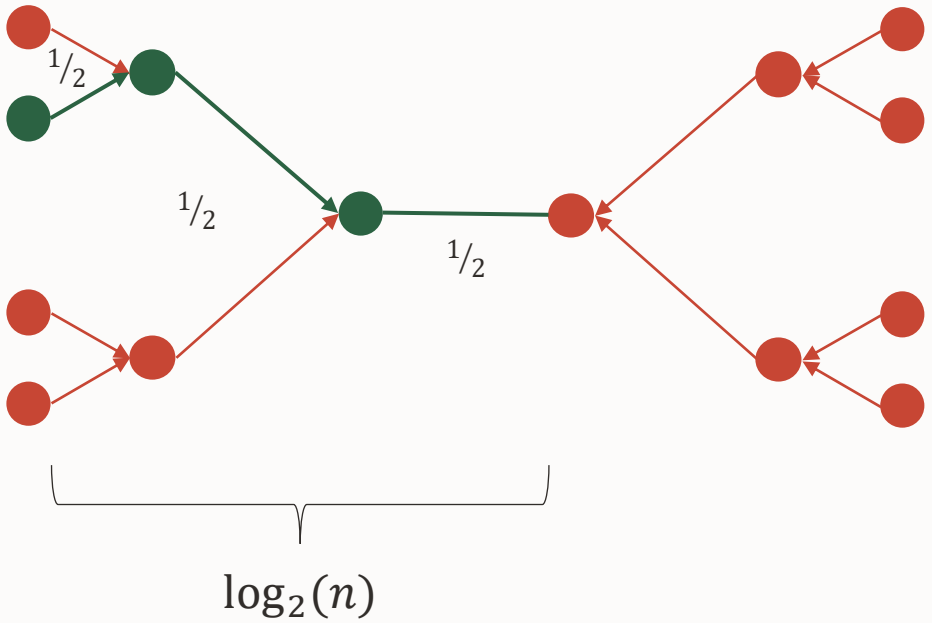
Approach: Localization

- Instead of directly debugging a long file, pinpoint location of bug and then intervene on it locally
- Decompose a buggy piece of code into smaller components
- Do **tournament-style (ReSET) competition** between components using LLM to find buggy one



Approach: Localization

- Instead of directly debugging a long file, pinpoint location of bug and then intervene on it locally
- Decompose a buggy piece of code into smaller components
- Do tournament-style (ReSET) competition between components using LLM to find buggy one



$$\text{random baseline win probability} = \left(\frac{1}{2}\right)^{\log_2(n)} = \frac{1}{n}$$

$$\mathbb{E}[\text{random baseline accuracy}] = \frac{1}{|\mathcal{D}|}\sum_{x \in \mathcal{D}} \frac{1}{|x|}$$

HEFix Line-Level Localization

	1-shot	3-shot
Coinflip	0.2326	
CodeLlama 34b Instruct	39 / 86 = 0.4534	**43** / 86 = **0.5**
CodeLlama 70b Instruct	**61** / 86 = **0.7093**	58 / 86 = 0.6744
CodeLlama 70b Python	**55** / 86 = **0.6395**	52 / 86 = 0.6046
CodeLlama 70b	**65** / 86 = **0.7558**	56 / 86 = 0.6511

Line-level localization accuracies on HumanEvalFix test split

Future Work

Future Work

- Want to know how this works on **real codebase**
  - apply to **Oracle VOS**
  - are sketches helpful to humans? do study with **human-in-the-loop**

Future Work

- Want to know how this works on **real codebase**
  - apply to **Oracle VOS**
  - are sketches helpful to humans? do study with **human-in-the-loop**

- Very **long context**: sketch it for compression, then do debugging

Future Work

- Want to know how this works on **real codebase**
  - apply to **Oracle VOS**
  - are sketches helpful to humans? do study with **human-in-the-loop**

- Very **long context**: sketch it for compression, then do debugging

- Right now all our sketches look like algorithms, but we can **add more stuff**
  - data structures
  - corner cases
  - failing test case info
  - Etc.

Future Work

- Want to know how this works on **real codebase**
  - apply to **Oracle VOS**
  - are sketches helpful to humans? do study with **human-in-the-loop**

- Very **long context**: sketch it for compression, then do debugging

- Right now all our sketches look like algorithms, but we can **add more stuff**
  - data structures
  - corner cases
  - failing test case info
  - Etc.

- Do actual bootstrapping — **training**!

Conclusion

Conclusion

- We came up with idea of **sketching** for LLM-based debugging

Conclusion

- We came up with idea of **sketching** for LLM-based debugging
- Think it has benefits of interpretability

Conclusion

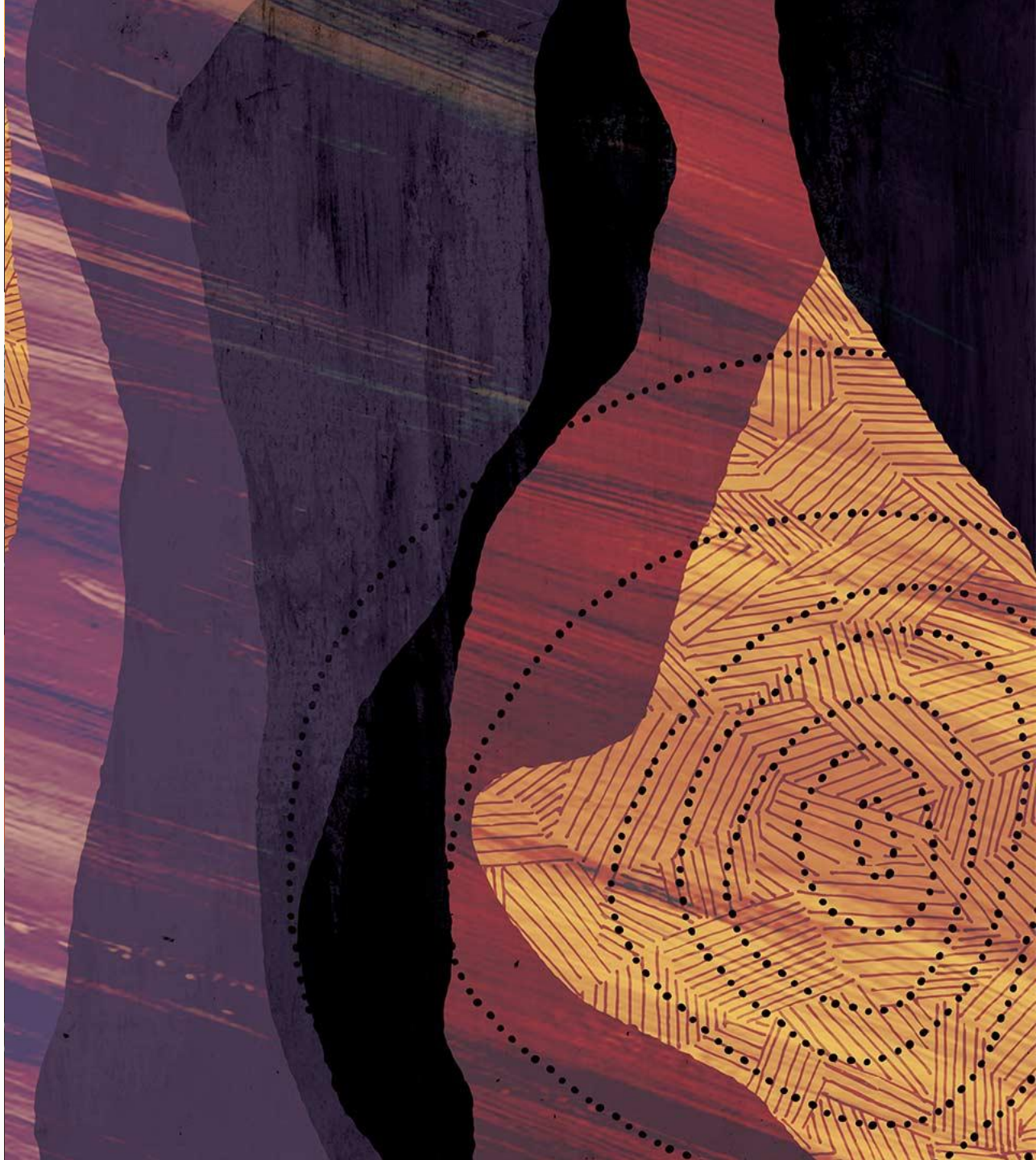
- We came up with idea of **sketching** for LLM-based debugging
- Think it has benefits of interpretability
- But doesn’t seem to help too much on HumanEvalFix

Conclusion

- We came up with idea of **sketching** for LLM-based debugging
- Think it has benefits of interpretability
- But doesn’t seem to help too much on HumanEvalFix
- We think this is fruitful way of thinking about problem, and look forward to future work

Questions?

Thank you!



Appendix

Decoding hyperparameters

- *sampling*
  - {"do_sample": **True**, "max_new_tokens": 250, "num_beams": **1**, "num_return_sequences": **10**, "temperature": 1.0}
    - keep defaults of top_k=50 and top_p=0 to achieve top-k filtering
- *beam*
  - {"do_sample": **False**, "max_new_tokens": 250, "num_beams": **10**, "num_return_sequences": **10**, "temperature": 1.0}
- *greedy*
  - {"do_sample": **False**, "max_new_tokens": 250, "num_beams": **1**, "num_return_sequences": **1**, "temperature": 1.0}

$I \rightarrow P$ “instruction to bugfix”

You are a helpful debugging assistant. Given task instructions in natural language, your task is to generate the correct implementation in Python.

```
------ Example 1 ------

You are given the task instruction:

<instruction>
Write a function to find the similar elements from the given two tuple lists.
</instruction>

Given the information above, generate the correct Python implementation for the task:

<debugged>
def similar_elements(t1, t2):
    s1 = set(t1)
    s2 = set(t2)
    return tuple(s1.intersection(s2))
</debugged>
```

```
------ Real Example ------

You are given the task instruction:

<instruction>
{instruction}
</instruction>

Given the information above, please generate the correct Python implementation for this task:
```

$(I, \bar{P}) \rightarrow P$

“instruction+buggy to bugfix”

You are a helpful debugging assistant. Given task instructions in natural language and a buggy Python implementation, your task is to generate the correct Python implementation, while adhering as closely as possible to the conventions of the original implementation.

```
------ Example 1 ------

You are given the task instruction:

<instruction>
Write a function to find the similar elements from the given two tuple lists.
</instruction>

You are also given the following buggy Python implementation:

<buggy_implementation>
def similar_elements(t1, t2):
    s1 = set(t1)
    s2 = set(t2)
    return s1.intersection(s2)
</buggy_implementation>

Please generate the correct Python implementation, while adhering as closely as possible to the conventions of the original implementation:

<debugged>
def similar_elements(t1, t2):
    s1 = set(t1)
    s2 = set(t2)
    return tuple(s1.intersection(s2))
</debugged>
```

```
------ Real Example ------

You are given the task instruction:

<instruction>
{instruction}
</instruction>

You are also given the following buggy Python implementation:

<buggy_implementation>
{buggy_code}
</buggy_implementation>

Please generate the correct Python implementation, while adhering as closely as possible to the conventions of the original implementation:
```

$(I, \bar{P}) \rightarrow Z^*$ “instruction + buggy to sketch”

You are a helpful debugging assistant. Given task instructions and a buggy Python implementation, your task is to generate a logically correct sketch, in English, for the algorithm which the buggy code is trying to implement.

```
------ Example {example_number} ------

You are given the task instruction:

<instruction>
{instruction}
</instruction>

You are also given the following buggy Python implementation:

<buggy_implementation>
{buggy_code}
</buggy_implementation>

Please generate a sketch, in English, for the algorithm which the buggy code is trying to implement:

<algorithm_sketch>
{sketch}
</algorithm_sketch>
```

$Z^* \rightarrow P$

“sketch to bugfix”

You are a helpful debugging assistant. Given an algorithm sketch in English, your task is to generate the correct Python implementation for the algorithm described in the sketch.

```
------ Example {example_number} ------

You are given the following algorithm sketch:

<algorithm_sketch>
{sketch}
</algorithm_sketch>

Please generate the correct Python implementation for the algorithm described in the above sketch:

<debugged>
{implementation}
</debugged>
```

$(I, \bar{P}) \rightarrow (Z^*, P)$ “instruction + buggy to sketch + bugfix”

You are a helpful debugging assistant. Given task instructions and a buggy Python implementation, you should generate a logically correct sketch, in English, for the algorithm which the buggy code is trying to implement. You should also generate the correct Python implementation for the algorithm described in the sketch you generated.

```
------ Example {example_number} ------

You are given the task instruction:

<instruction>
{instruction}
</instruction>

You are also given the following buggy Python implementation:

<buggy_implementation>
{buggy_code}
</buggy_implementation>

Please generate a sketch, in English, for the algorithm which the buggy code is trying to implement:

<algorithm_sketch>
{sketch}
</algorithm_sketch>

Also please generate the correct Python implementation for the algorithm described in the above sketch you generated:

<debugged>
{bugfix}
</debugged>
```

$(I, \bar{P}, (I, \bar{P}) \rightarrow Z^*) \rightarrow P$ “instruction + buggy + sketch to bugfix”

You are a helpful debugging assistant. You are given task instructions, a buggy Python implementation, and a logically correct sketch for the algorithm which the buggy code is trying to implement. Your task is to generate the correct Python implementation for the algorithm described in the sketch, while adhering as closely as possible to the conventions of in the original implementation.

```
------ Example {example_number} ------

You are given the task instruction:

<instruction>
{instruction}
</instruction>

You are also given the following buggy Python implementation:

<buggy_implementation>
{buggy_code}
</buggy_implementation>

You are also given a sketch, in English, for the algorithm which the buggy code is trying to implement:

<algorithm_sketch>
{sketch}
</algorithm_sketch>

Please generate the correct Python implementation for the algorithm described in the above sketch, while adhering as closely as possible to the conventions of in the original implementation:

<debugged>
{debugged}
</debugged>
```

HumanEvalFix line-level bug localization

You will be given an instruction that describes a function, followed by an implementation of that function that contains a bug. Then, you will be provided with a few candidate code chunks copied from the buggy implementation with corresponding identifiers. Please read the instruction and buggy implementation carefully, and return the identifier that corresponds to the copied code chunk most likely to contain the bug.

```
------------ Example {example_idx} ------------
<instruction>
{instruction}
</instruction>
<implementation>
{buggy_code}
</implementation>
<candidates>
(A) {label_a}     {choice_a}
(B) {label_b}     {choice_b}
</candidates>
<answer>
{answer}
</answer>
```

```
------------ Real Example ------------
<instruction>
{instruction}
</instruction>
<implementation>
{buggy_code}
</implementation>
<candidates>
(A) {label_a}     {choice_a}
(B) {label_b}     {choice_b}
</candidates>"""
```

ORACLE