Contextual Communication in Programming

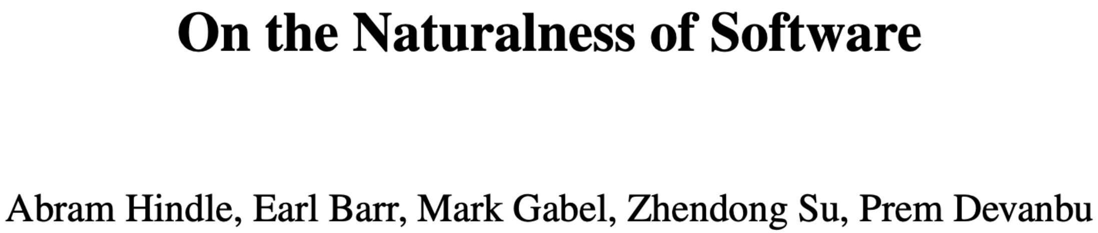
Daniel Fried

Meta → Language Technologies Institute

Today’s Question

Are bigger models the solution for AI-assisted programming?

Posing This Question in 2012…



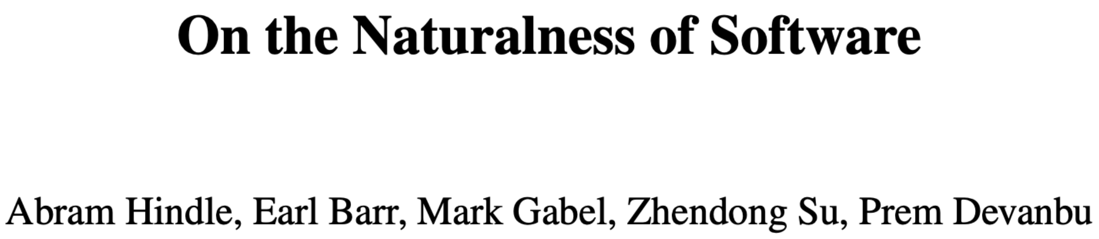
On the Naturalness of Software

Abram Hindle, Earl Barr, Mark Gabel, Zhendong Su, Prem Devanbu

[ICSE 2012; Most Influential Paper 2022]

Natural languages like English are rich, complex, and powerful. We begin with the conjecture that most software is also natural, in the sense that it is created by humans at work, with all the attendant constraints and limitations—and thus, like natural language, it is also likely to be repetitive and predictable. We then proceed to ask whether a) code can be usefully modeled by statistical language models and b) such models can be leveraged to support software engineers.

Posing This Question in 2012…

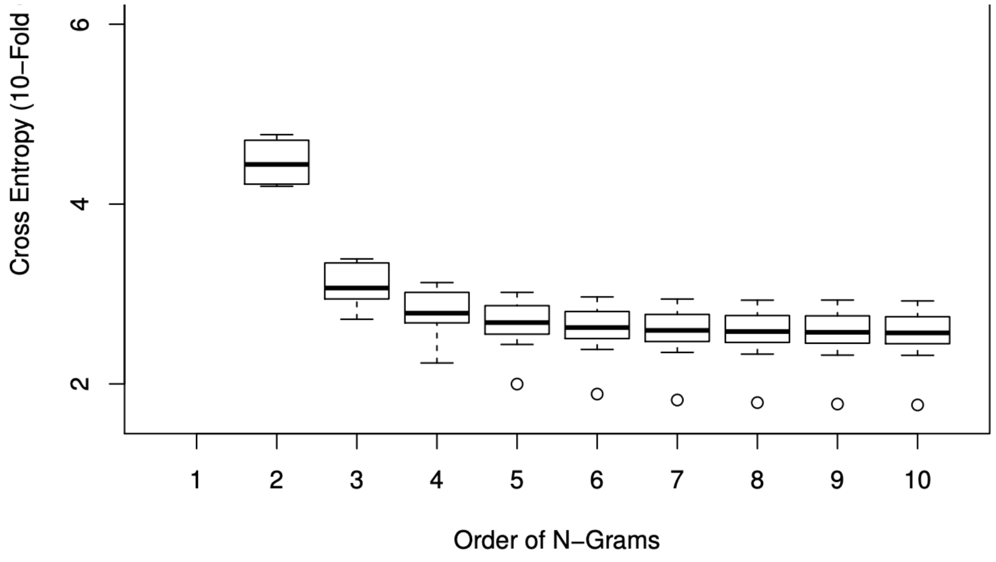


On the Naturalness of Software

Abram Hindle, Earl Barr, Mark Gabel, Zhendong Su, Prem Devanbu

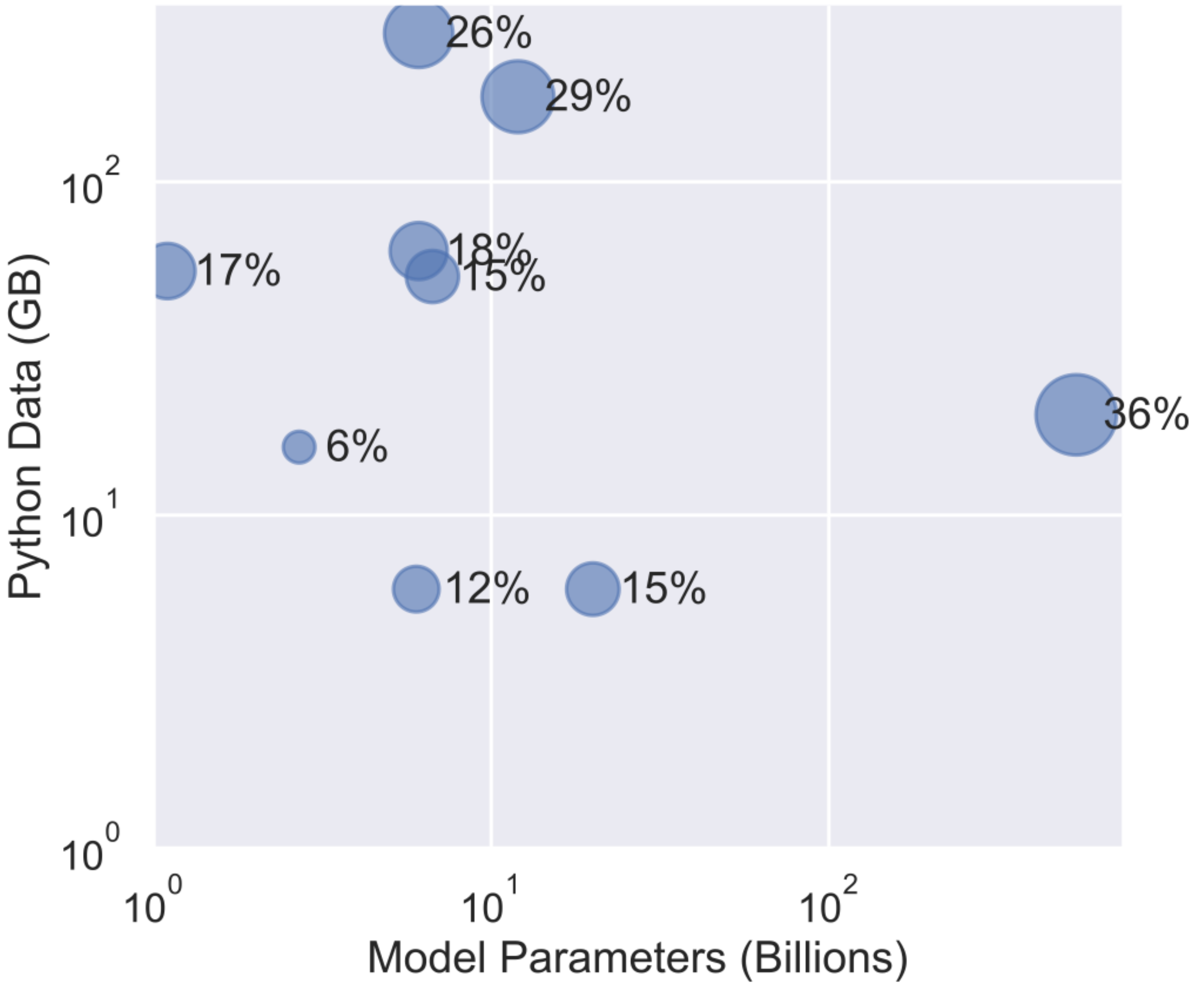
[ICSE 2012; Most Influential Paper 2022]

- n-gram models trained on ~25 million lines of code
- Substantial improvements to Eclipse’s auto-complete
- But, 3-4 orders of magnitude less data than modern neural models



… and now

Function pass rate on a Python synthesis dataset [Chen et al. 2021] by data & model scale:



[Compiled from Chen et al. 2021, Xu et al. 2021, Li et al. 2021, Fried et al. 2022, Nijkamp et al. 2022, Chowdhery et al. 2022]

Today’s Question

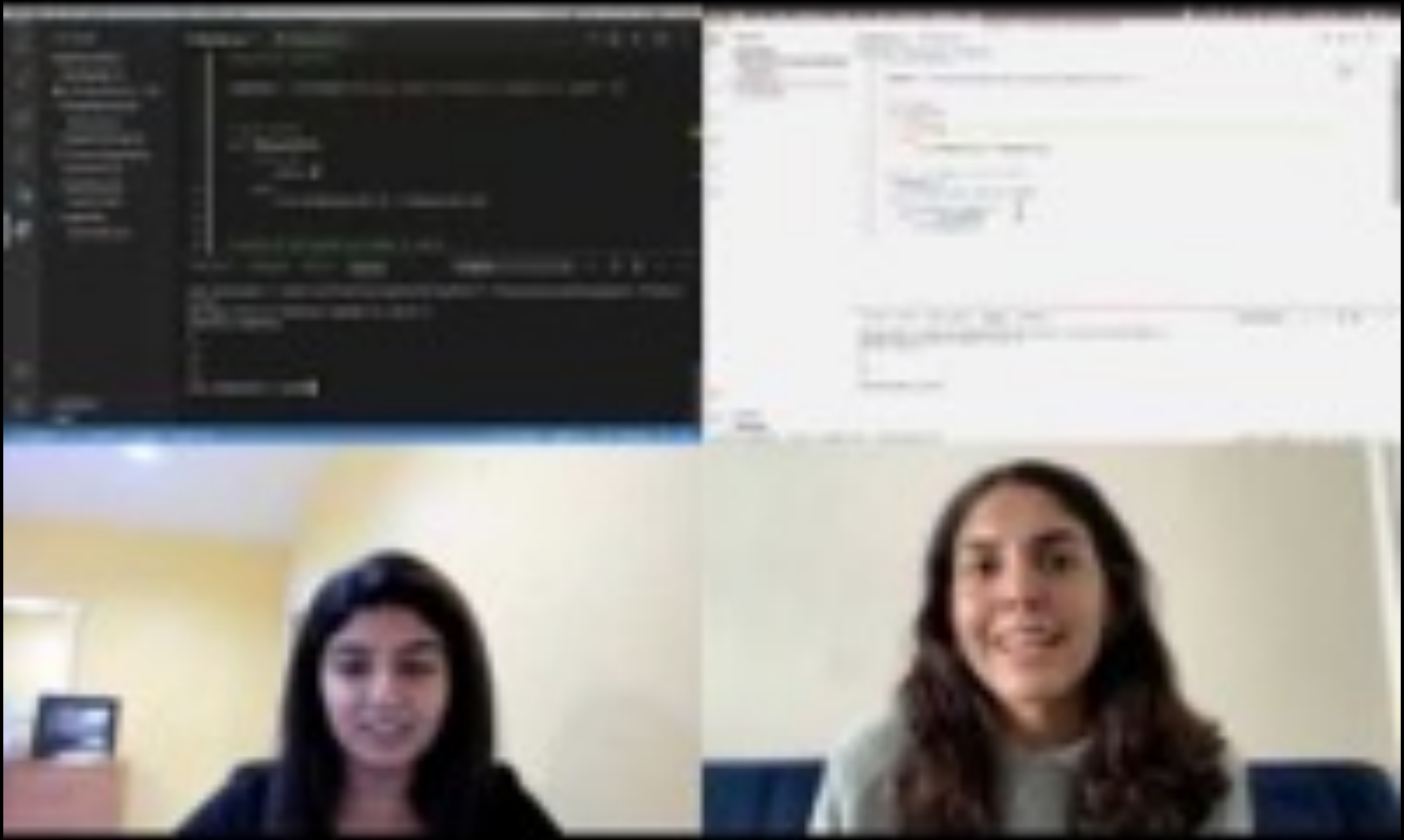
part of

Are bigger models the solution
^
for AI-assisted programming?

YES

Programming as Communication

We begin with the conjecture that most software… is created by humans at work,
~~with all the attendant constraints and limitations~~
communicating with the compiler, other developers, and themselves,
and thus, like natural language,
~~it is also likely to be repetitive and predictable.~~
writing software is a form of contextual and interactive communication.
We then proceed to ask whether a) code can be usefully modeled by statistical language models and b) such models can be leveraged to support software engineers.



[VSCode Live Share Demo, 2020]

Communicating with Multiple Modalities

	Natural Language	Partial Code	Tests & Execution	Edits to Code	Deictic (Pointing / Highlighting)	…
As Inputs	[Fried & Aghajanyan et al., 2022]	[Fried & Aghajanyan et al., 2022]	[Shi et al. 2022]			
As Outputs	[Fried & Aghajanyan et al., 2022]			[Wallace et al., in progress]		

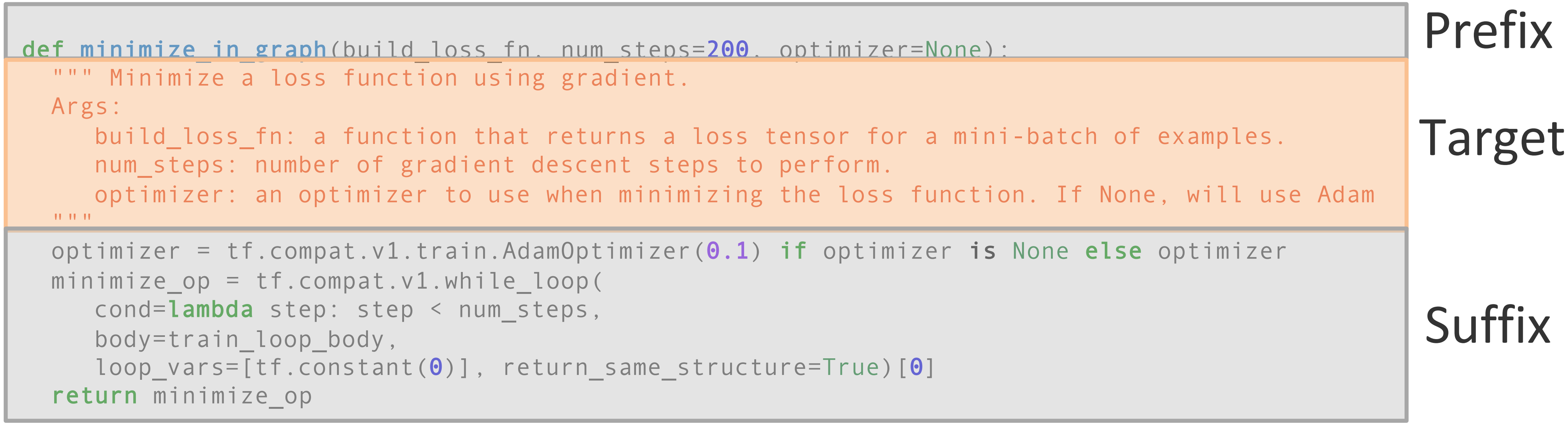
Modality Choice
[Lin et al., 2022]

Communicating with Multiple Modalities

	Natural Language	Partial Code	Tests & Execution	Edits to Code	Deictic (Pointing / Highlighting)	…
As Inputs	[Fried & Aghajanyan et al., 2022]	[Fried & Aghajanyan et al., 2022]	[Shi et al. 2022]			
As Outputs	[Fried & Aghajanyan et al., 2022]			[Wallace et al., in progress]		

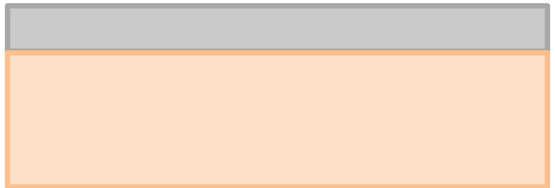
Modality Choice
[Lin et al., 2022]

Neural Code Model Objectives



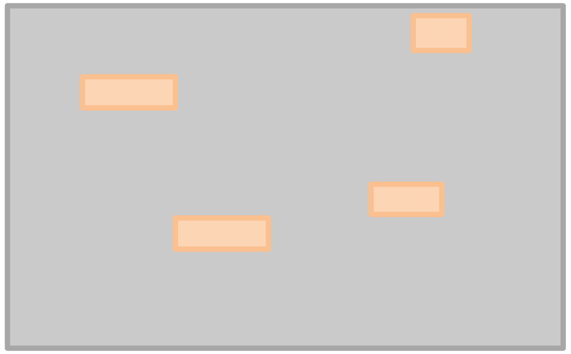
```
def minimize_in_graph(build_loss_fn, num_steps=200, optimizer=None):
  """ Minimize a loss function using gradient.
  Args:
     build_loss_fn: a function that returns a loss tensor for a mini-batch of examples.
     num_steps: number of gradient descent steps to perform.
     optimizer: an optimizer to use when minimizing the loss function. If None, will use Adam
  """
  optimizer = tf.compat.v1.train.AdamOptimizer(0.1) if optimizer is None else optimizer
  minimize_op = tf.compat.v1.while_loop(
     cond=lambda step: step < num_steps,
     body=train_loop_body,
     loop_vars=[tf.constant(0)], return_same_structure=True)[0]
  return minimize_op
```

“Causal” (L-to-R)



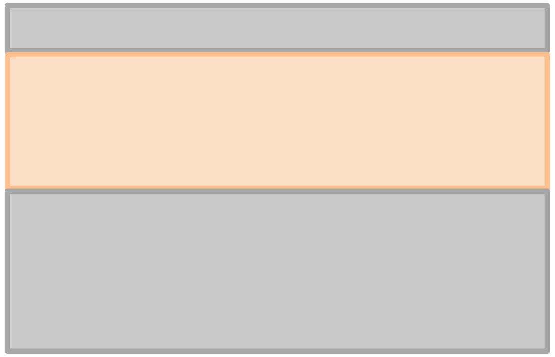
[e.g. GPT-*, Codex]

Masked Infilling



[e.g. BERT, CodeBERT]

Causal Masking

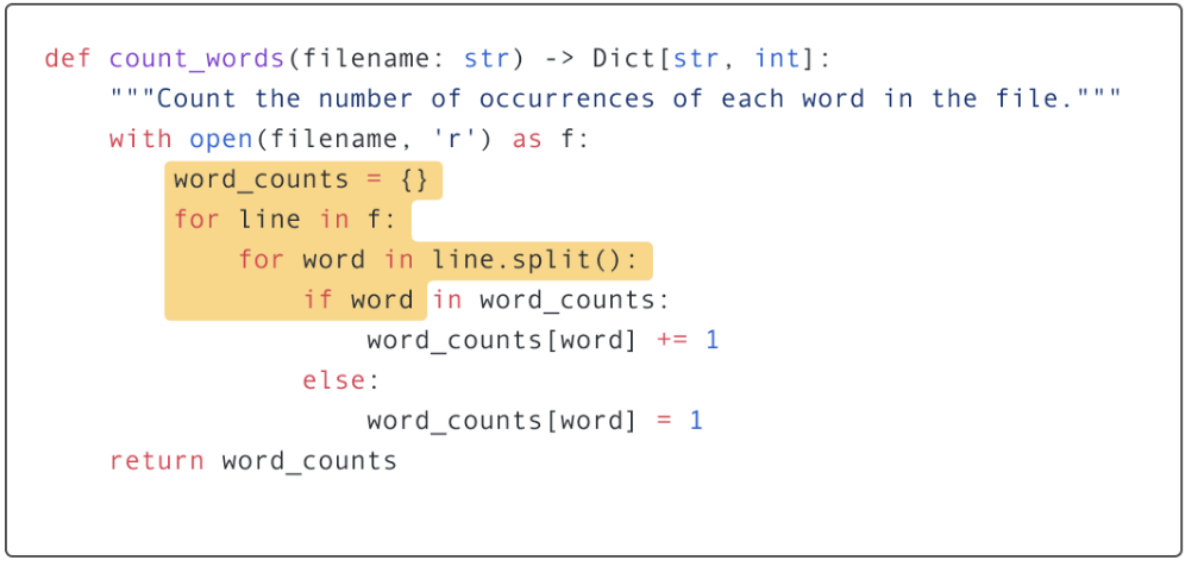


[Donahue+ 2020, Aghajanyan+ 2022, ours, Bavarian+ 2022]

InCoder: Code Generation and Infilling

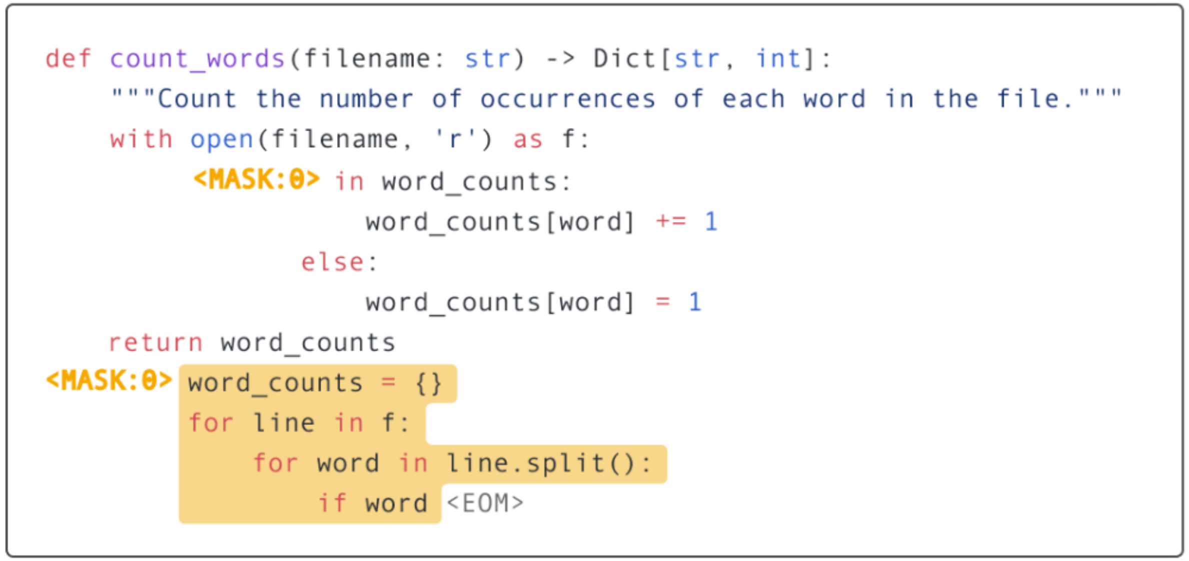
Training

Original Document



```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

Masked Document

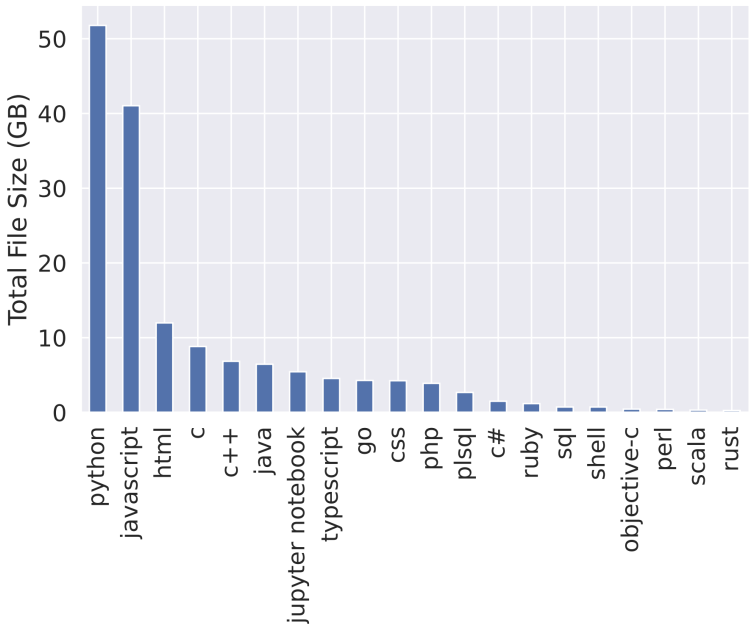


```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
         <MASK:0> in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
<MASK:0> word_counts = {}
         for line in f:
             for word in line.split():
                 if word <EOM>
```

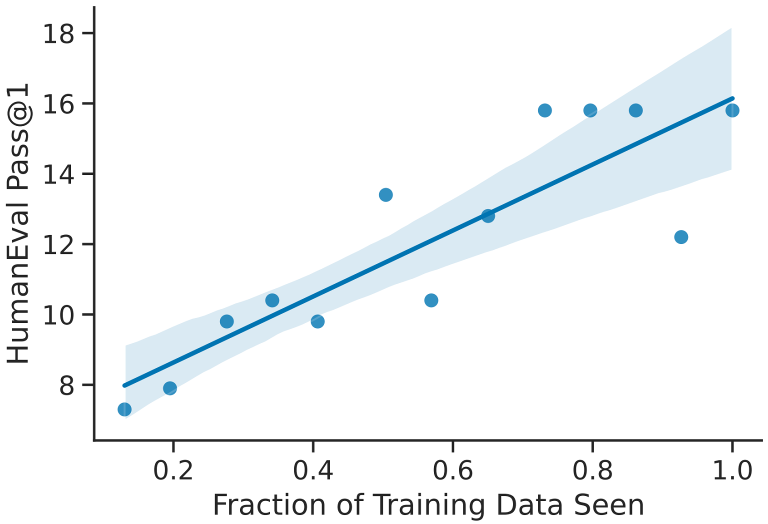
[Fried*, Aghajanyan* et al., 2022]

InCoder: Code Generation and Infilling

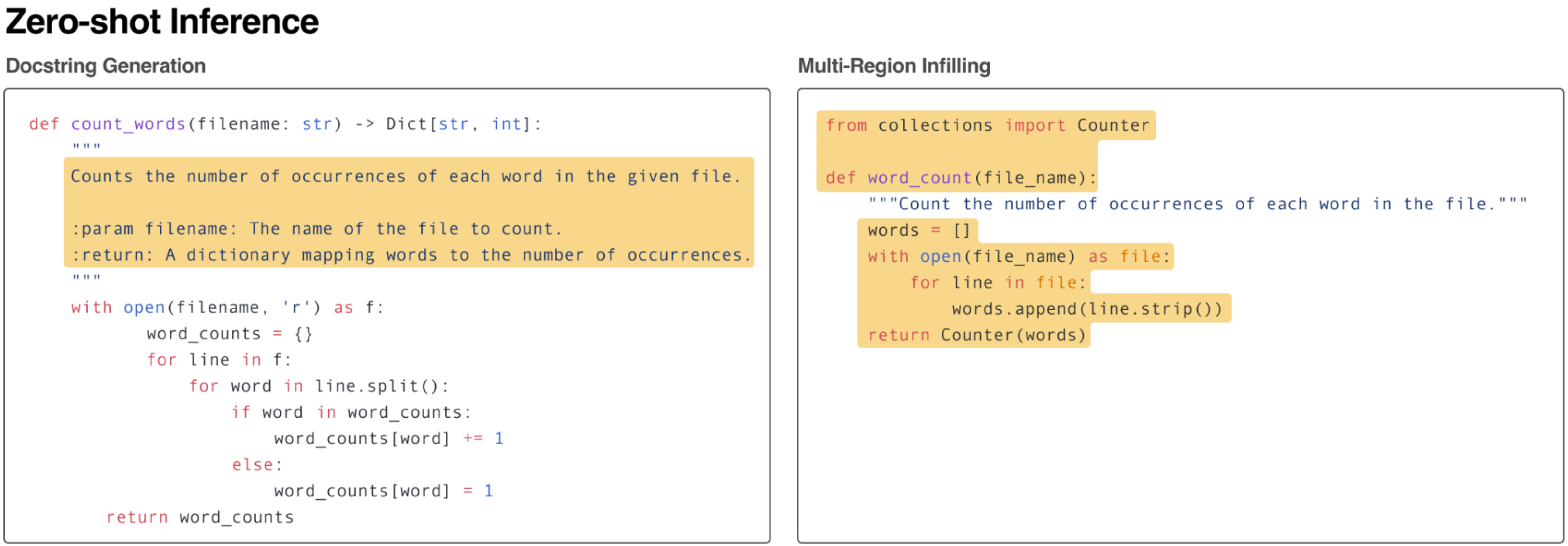
- Data
  - 600K permissively-licensed repositories from GitHub & GitLab
  - StackOverflow: questions, answers, comments



- Models
  - Standard transformer LM
  - 1B model: ~1 week on 128 GPUs
  - 6B model: ~3 weeks on 240 GPUs



InCoder: Code Generation and Infilling



Zero-shot Inference

Docstring Generation

```
def count_words(filename: str) -> Dict[str, int]:
    """
    Counts the number of occurrences of each word in the given file.

    :param filename: The name of the file to count.
    :return: A dictionary mapping words to the number of occurrences.
    """
    with open(filename, 'r') as f:
           word_counts = {}
           for line in f:
               for word in line.split():
                   if word in word_counts:
                       word_counts[word] += 1
                   else:
                       word_counts[word] = 1
       return word_counts
```

Multi-Region Infilling

```
from collections import Counter

def word_count(file_name):
    """Count the number of occurrences of each word in the file."""
    words = []
    with open(file_name) as file:
        for line in file:
            words.append(line.strip())
    return Counter(words)
```

Also usable as a left-to-right generation model with no apparent loss in performance [see also Bavarian et al. 2022]

[Fried*, Aghajanyan* et al., 2022]

Evaluation

- Zero-shot evaluation on realistic code infilling tasks
- Compare the model in three different modes to evaluate benefits of suffix context

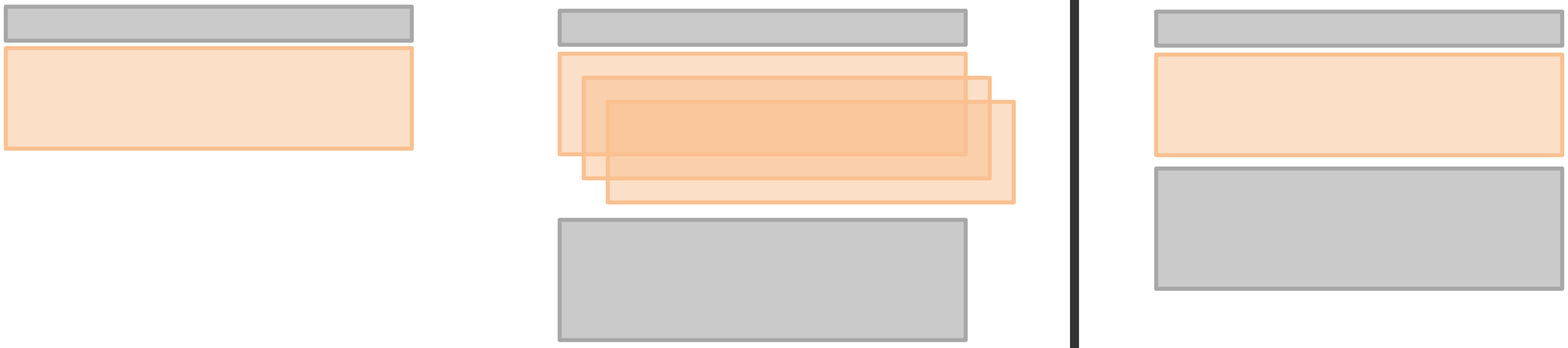
Baselines

Ours

Left-to-Right Single

Left-to-Right Rerank

Causal Masking



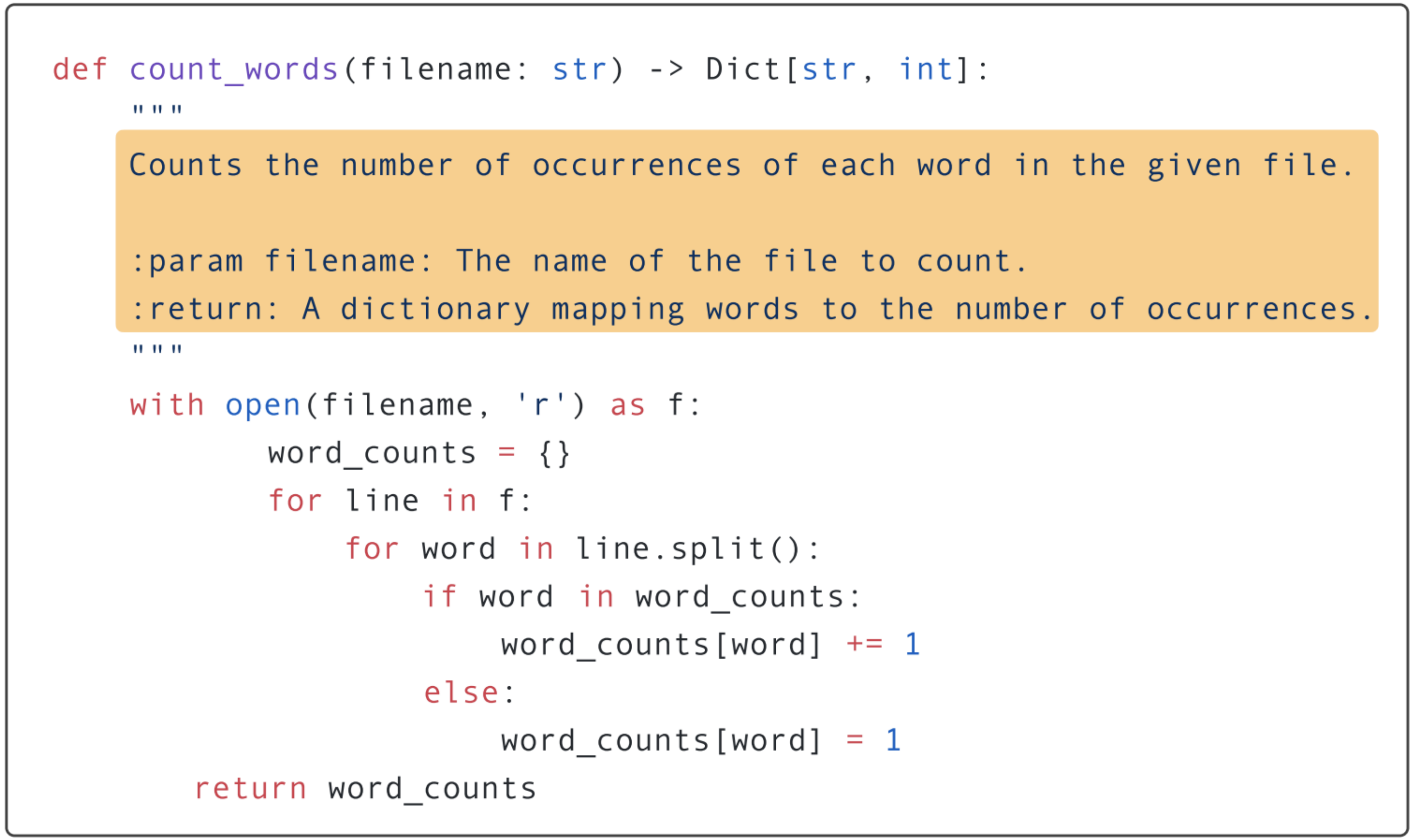
Doesn’t use suffix

Only uses suffix when reranking

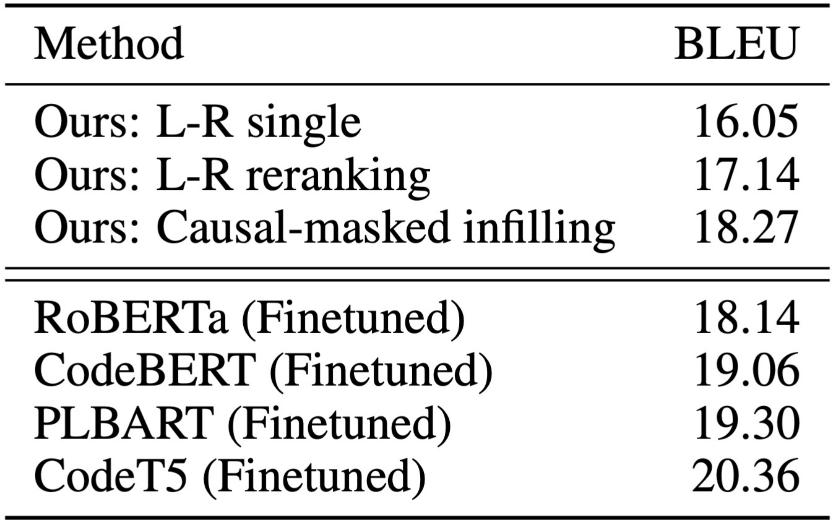
Uses suffix when generating

Evaluation

Docstring Generation



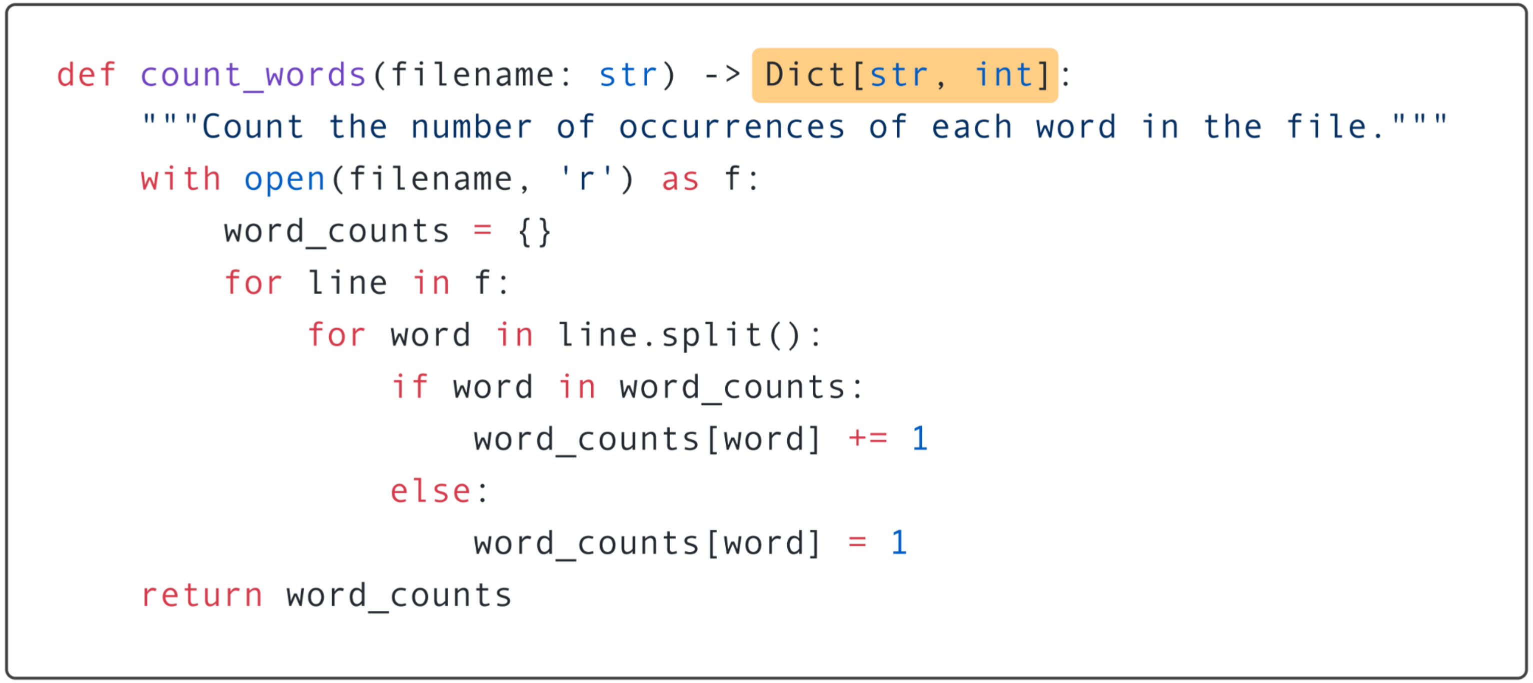
```
def count_words(filename: str) -> Dict[str, int]:
    """
    Counts the number of occurrences of each word in the given file.

    :param filename: The name of the file to count.
    :return: A dictionary mapping words to the number of occurrences.
    """
    with open(filename, 'r') as f:
           word_counts = {}
           for line in f:
               for word in line.split():
                   if word in word_counts:
                       word_counts[word] += 1
                   else:
                       word_counts[word] = 1
       return word_counts
```



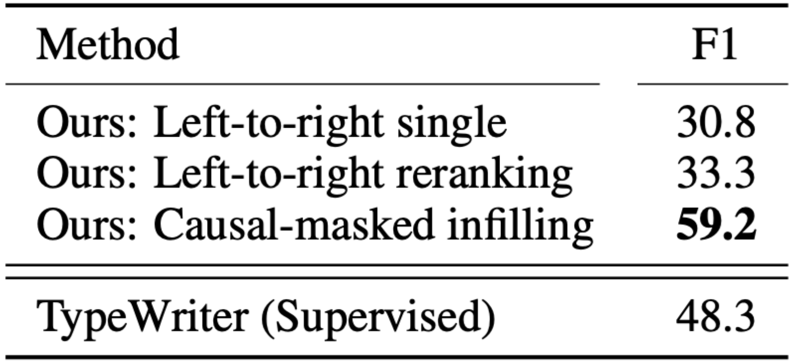
Method	BLEU
Ours: L-R single	16.05
Ours: L-R reranking	17.14
Ours: Causal-masked infilling	18.27
RoBERTa (Finetuned)	18.14
CodeBERT (Finetuned)	19.06
PLBART (Finetuned)	19.30
CodeT5 (Finetuned)	20.36

Evaluation

Type Inference

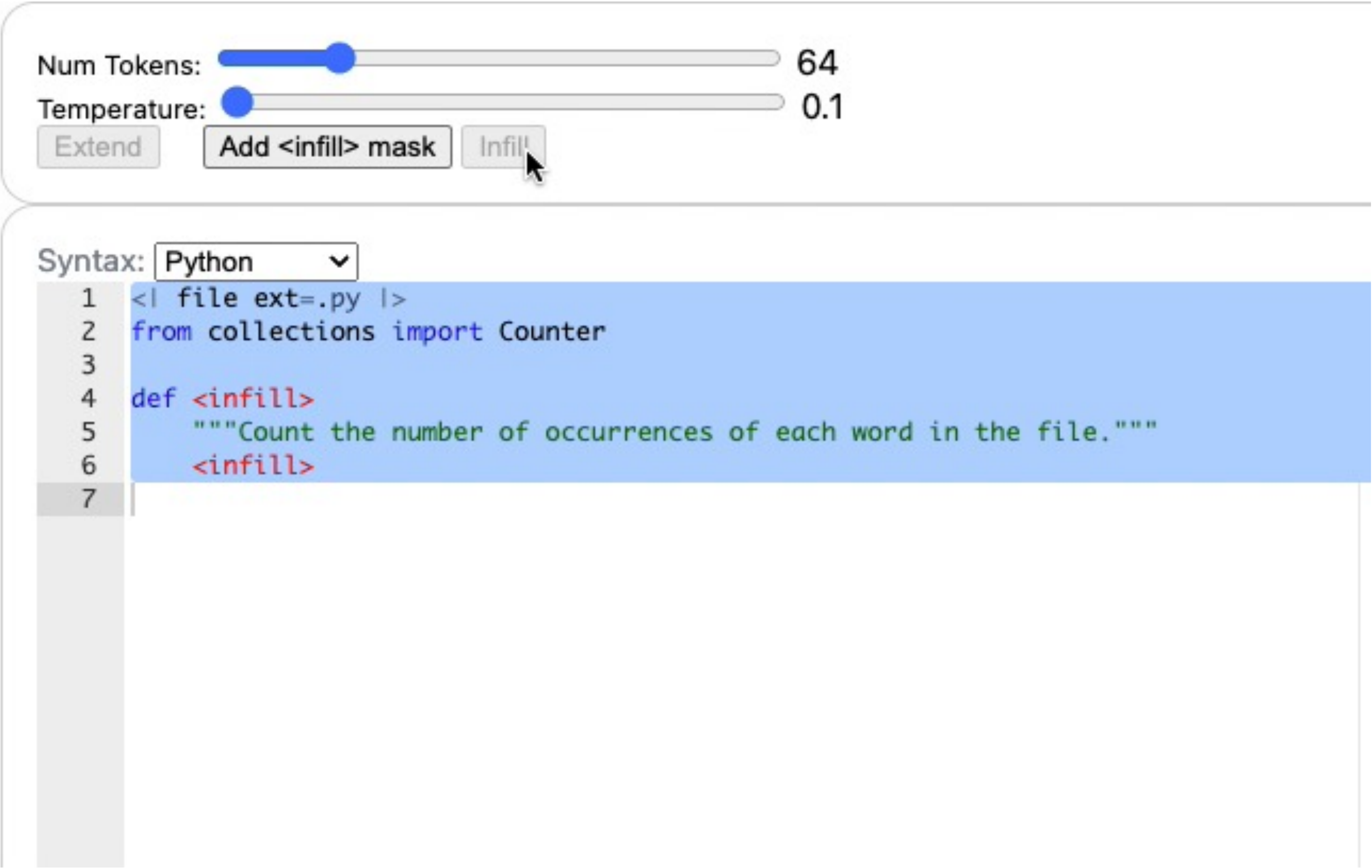


```python
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```



Method	F1
Ours: Left-to-right single	30.8
Ours: Left-to-right reranking	33.3
Ours: Causal-masked infilling	**59.2**
TypeWriter (Supervised)	48.3

Demo



Num Tokens: 64

Temperature: 0.1

```python
<| file ext=.py |>
from collections import Counter

def <infill>
    """Count the number of occurrences of each word in the file."""
    <infill>

```

Model Weights & Interactive Demo: huggingface.co/facebook/incoder-1B

Communicating with Multiple Modalities

	NL Instructions	Partial Code	Tests & Execution	Edits to Code	Deictic (Pointing / Highlighting)	…
As Inputs	[Fried & Aghajanyan et al., 2022]	[Fried & Aghajanyan et al., 2022]	[Shi et al. 2022]			
As Outputs	[Fried & Aghajanyan et al., 2022]			[Wallace et al., in progress]		

Modality Choice
[Lin et al., 2022]

Using Test Inputs

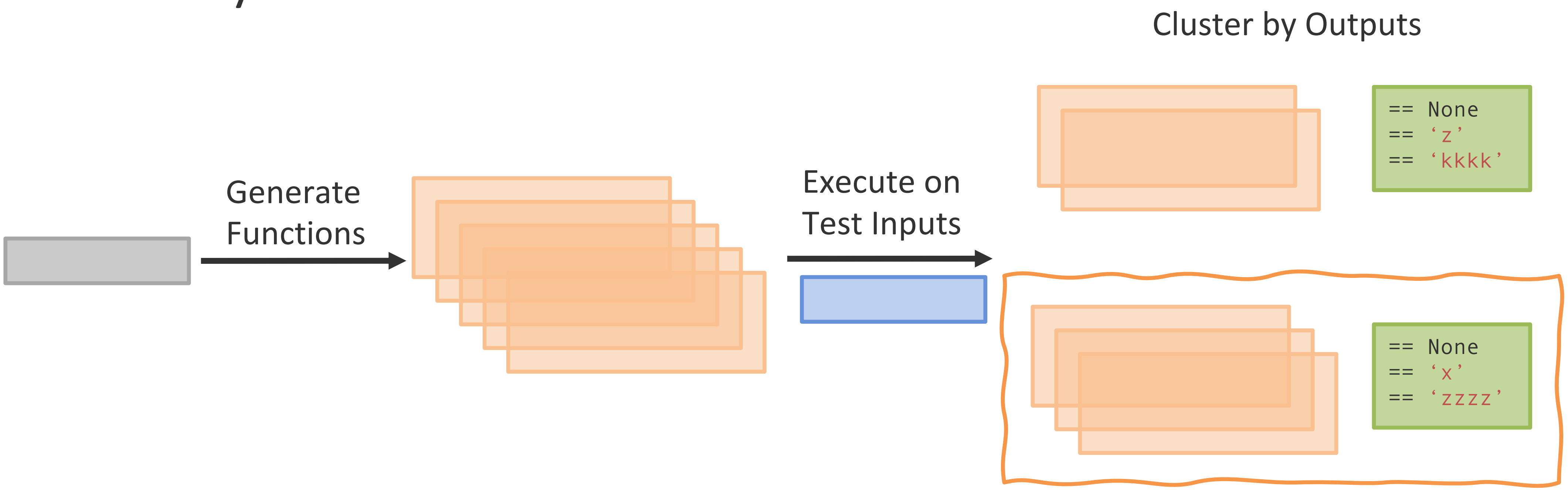
Description:

```
def longest(strings: List[str]) -> Optional[str]:
  """ Out of list of strings, return the longest one.
  Return the first one in case of multiple strings of
  the same length. Return None if the list is empty."""
```

Test Inputs:

```
longest([]) == ___
longest(['x', 'y', 'z']) == ___
longest(['x', 'yyy', 'zzzz', 'www', 'kkkk', 'abc']) == ___
```

Minimum Bayes Risk with Execution:

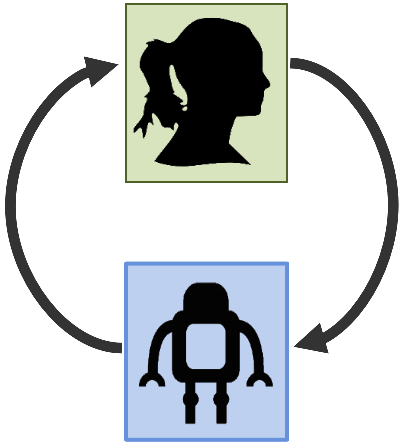


```
== None
== ‘z’
== ‘kkkk’
```

```
== None
== ‘x’
== ‘zzzz’
```

[Shi et al. 2022. See also *AlphaCode*, Li et al. 2022]

Other Features of Communication



- Communicative cost
  - Copilot outputs can be hard to understand [Vaithilingam et al. 2022]
  - Would a user rather type a comment or edit code?
- Resolving uncertainty
  - Disambiguate by prompting with test inputs [Zhong et al. 2022]
  - How to convey uncertainty to the user & build trust?
- Adaptation
  - Acceleration vs exploration modes for using Copilot [Barke et al. 2022]
  - API preferences, functional vs imperative, design patterns, documentation style …

Collaborators

Armen
Aghajanyan

Anca
Dragan

Marjan
Ghazvininejad

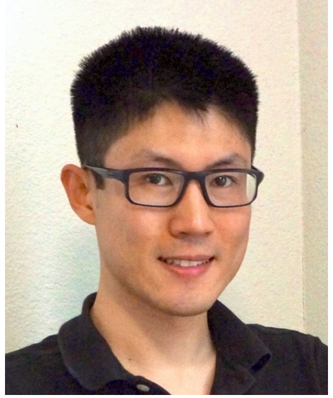
Dan
Klein

Mike
Lewis

Jessy
Lin

Freda
Shi

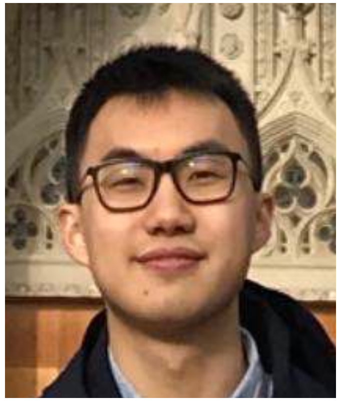
Eric
Wallace



Sida
Wang

Scott
Yih

Luke
Zettlemoyer



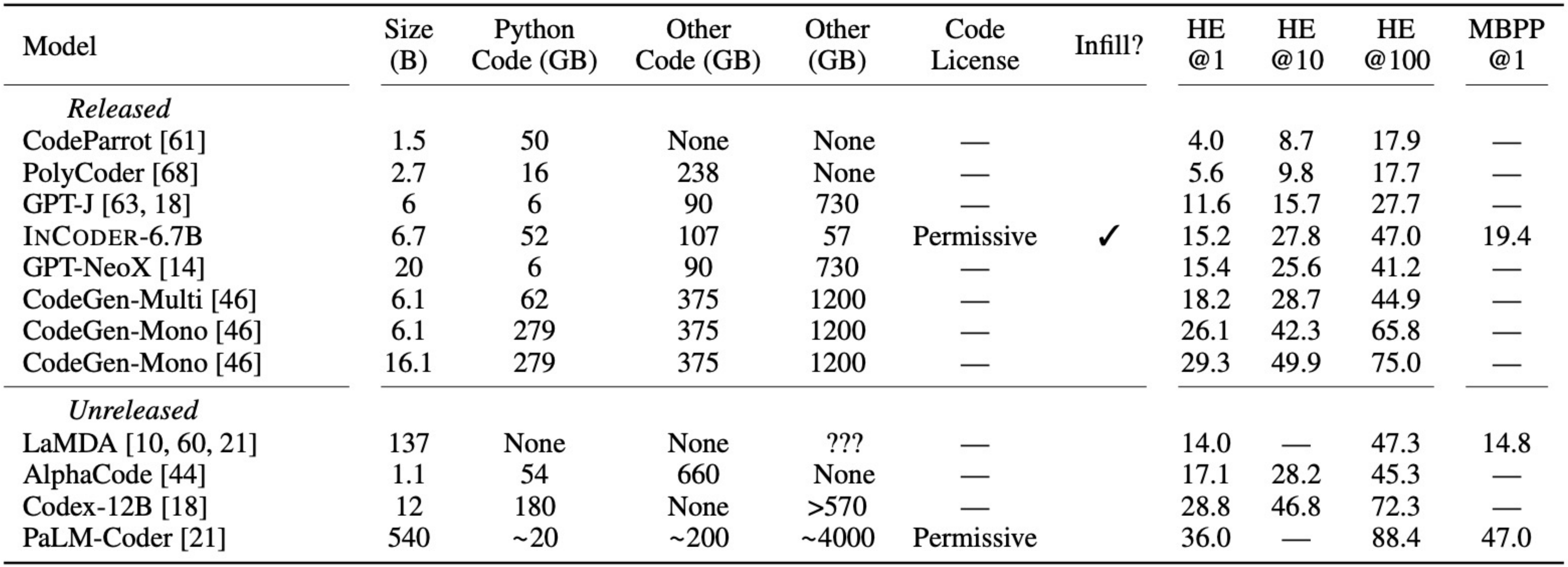
Ruiqi
Zhong

Thanks!

dfried@cs.cmu.edu
dpfried.github.io

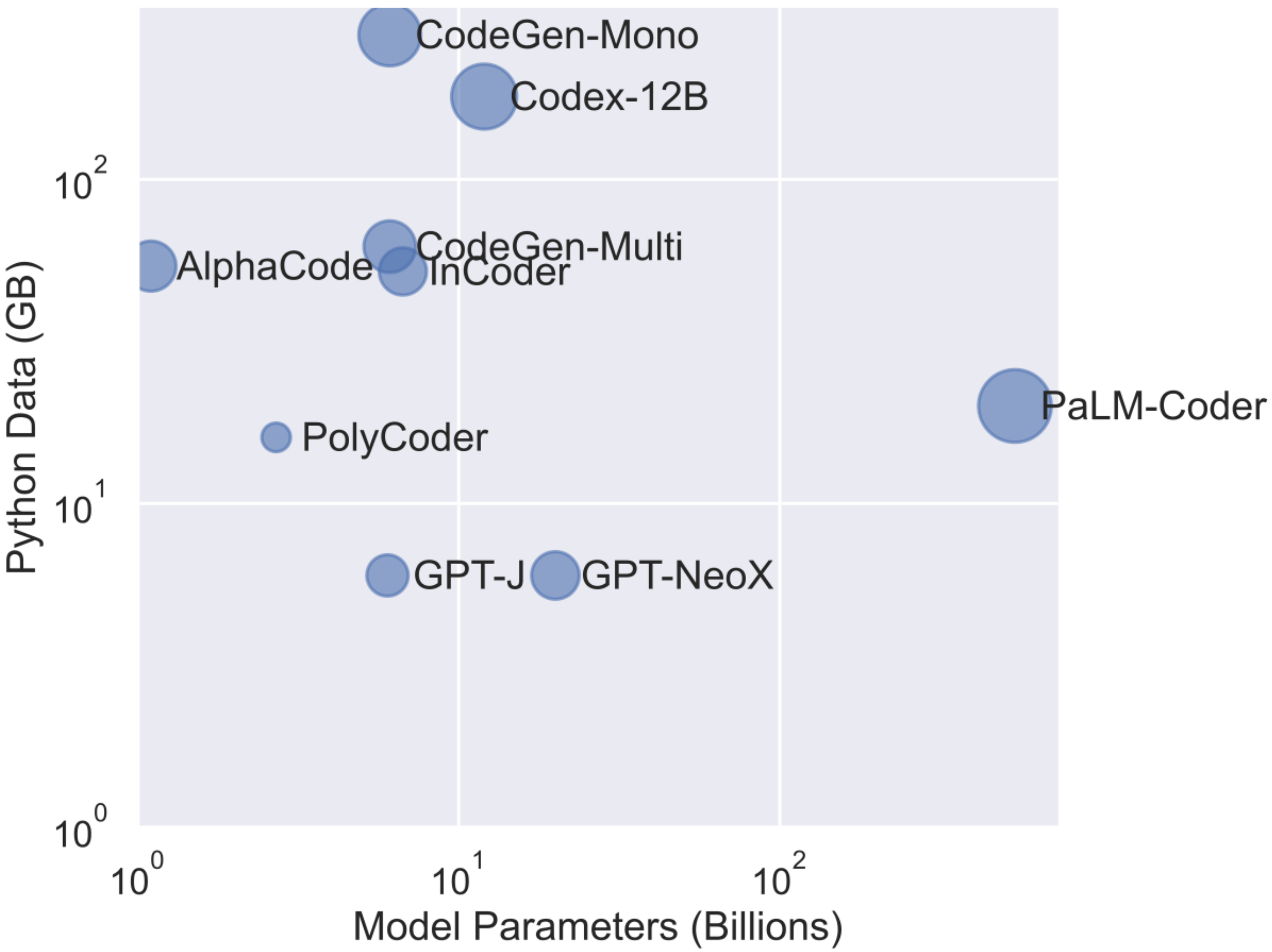
Backup Slides

Scale and Performance



Model	Size (B)	Python Code (GB)	Other Code (GB)	Other (GB)	Code License	Infill?	HE @1	HE @10	HE @100	MBPP @1
Released										
CodeParrot [61]	1.5	50	None	None	—		4.0	8.7	17.9	—
PolyCoder [68]	2.7	16	238	None	—		5.6	9.8	17.7	—
GPT-J [63, 18]	6	6	90	730	—		11.6	15.7	27.7	—
InCoder-6.7B	6.7	52	107	57	Permissive	✓	15.2	27.8	47.0	19.4
GPT-NeoX [14]	20	6	90	730	—		15.4	25.6	41.2	—
CodeGen-Multi [46]	6.1	62	375	1200	—		18.2	28.7	44.9	—
CodeGen-Mono [46]	6.1	279	375	1200	—		26.1	42.3	65.8	—
CodeGen-Mono [46]	16.1	279	375	1200	—		29.3	49.9	75.0	—
Unreleased										
LaMDA [10, 60, 21]	137	None	None	???	—		14.0	—	47.3	14.8
AlphaCode [44]	1.1	54	660	None	—		17.1	28.2	45.3	—
Codex-12B [18]	12	180	None	>570	—		28.8	46.8	72.3	—
PaLM-Coder [21]	540	~20	~200	~4000	Permissive		36.0	—	88.4	47.0

… and now

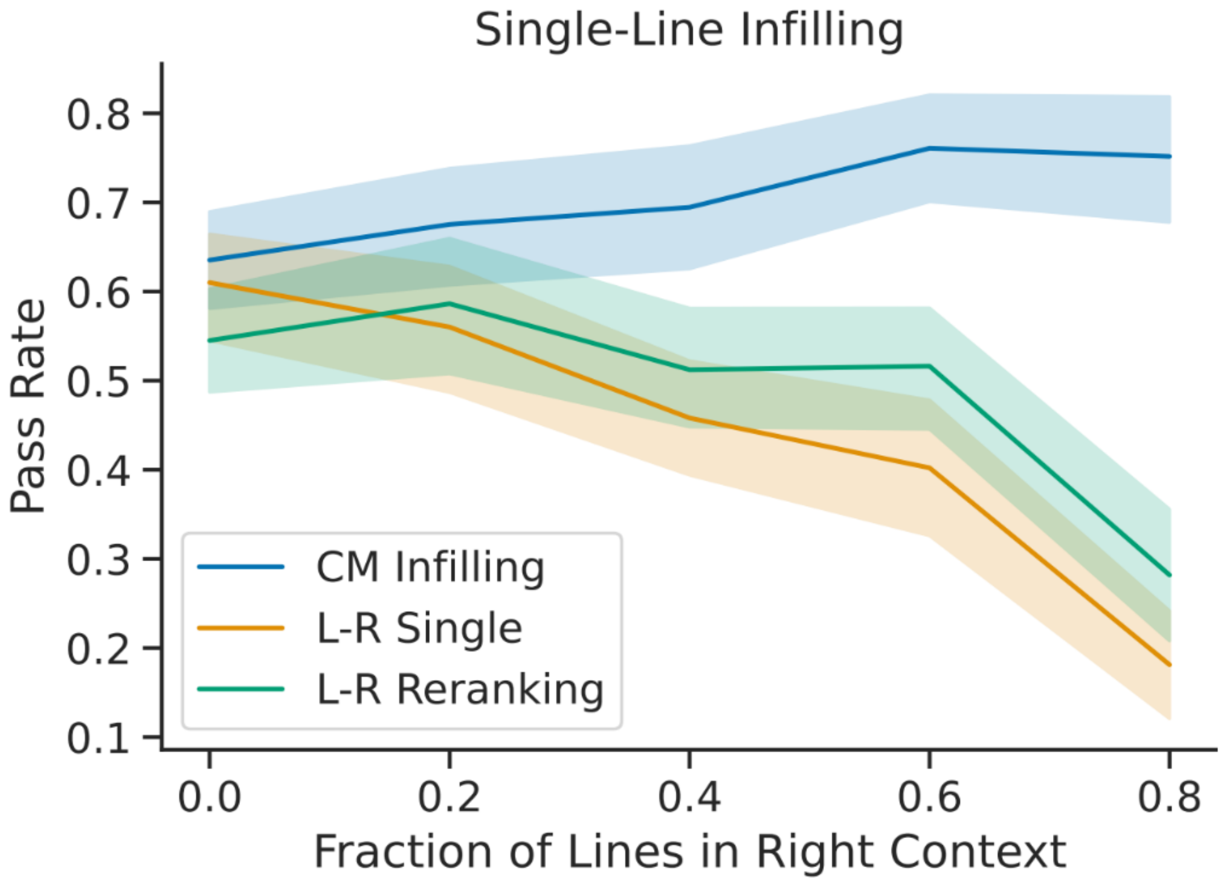


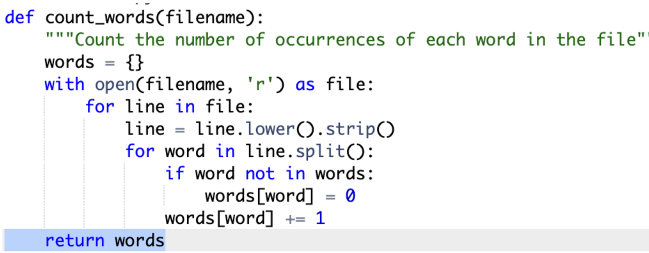
CodeGen-Mono
Codex-12B
CodeGen-Multi
AlphaCode
InCoder
PaLM-Coder
PolyCoder
GPT-J
GPT-NeoX
Python Data (GB)
10^2
10^1
10^0
10^0
10^1
10^2
Model Parameters (Billions)

Details

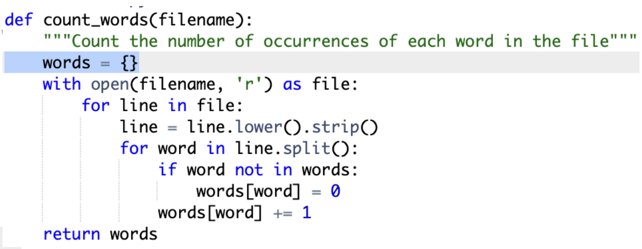
- Remove duplicate files using exact match on alphanumeric token sequences
- Tokenization: retrain byte-level BPE, modified to allow merging across spaces
- e.g. import numpy as np is a single token
  - 40% reduction in token count compared to GPT-2's tokenizer
    - Longer effective contexts & more efficient training
    - But, less compute spent in training may affect performance
- Meta-data conditioning and prediction
- <| file source=github stars=high filename=setup.py ext=.py |>
- Attributes can appear at beginning of the file (conditioning) or end (prediction)

Function completion



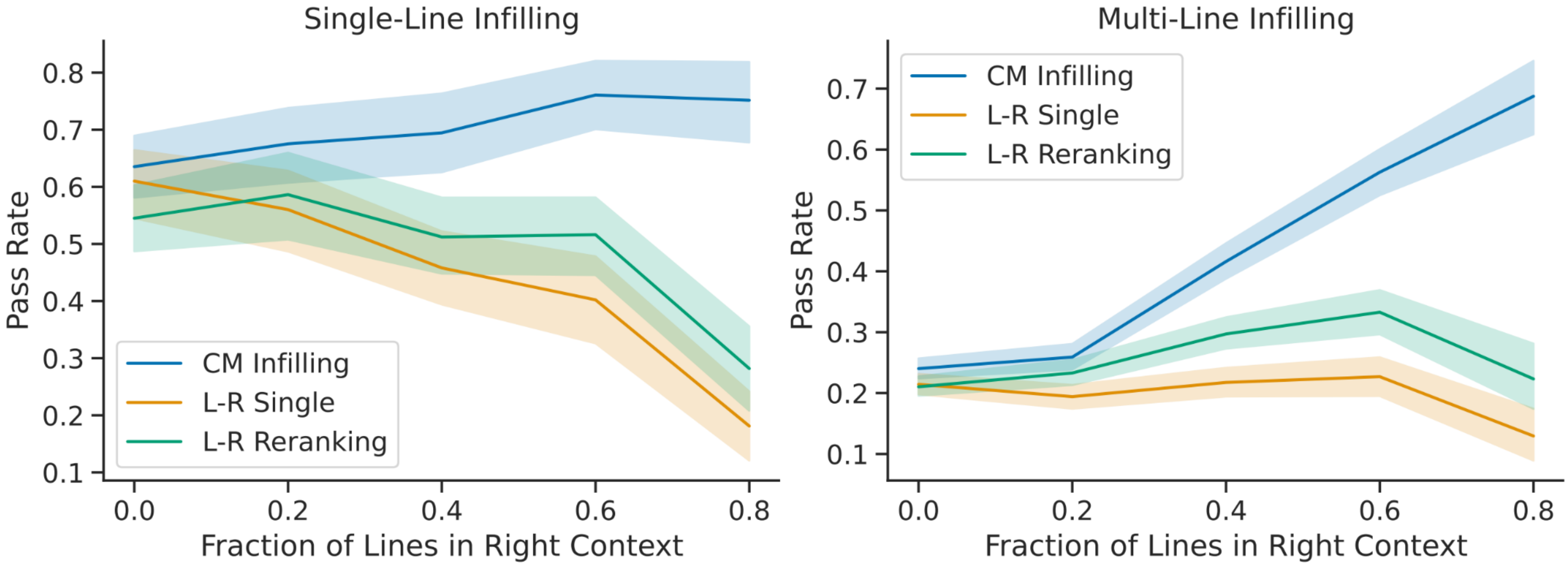


```python
def count_words(filename):
    """Count the number of occurrences of each word in the file"
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```



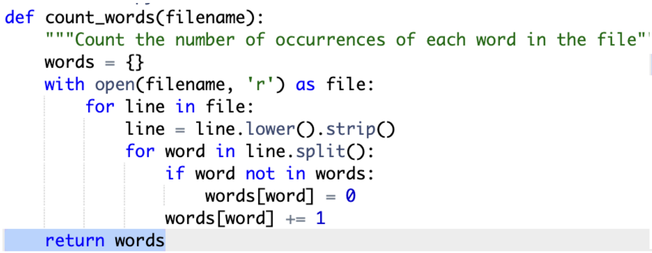
```python
def count_words(filename):
    """Count the number of occurrences of each word in the file"""
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```

Function completion

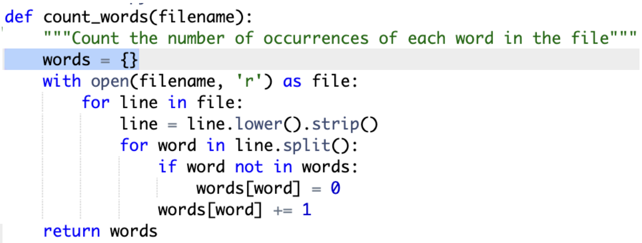


Single-Line Infilling

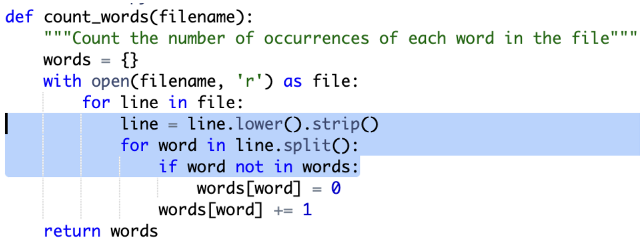
Multi-Line Infilling



```
def count_words(filename):
    """Count the number of occurrences of each word in the file"
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```



```
def count_words(filename):
    """Count the number of occurrences of each word in the file"""
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```



```
def count_words(filename):
    """Count the number of occurrences of each word in the file"""
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```