

Talk @ UVA  
Sep 30, 2024

# Building Code Intelligence with Language Models

 Yuxiang Wei, 3rd year PhD student at UIUC

 @YuxiangWei9

 <https://yuxiang.cs.illinois.edu>

# The ubiquitous code intelligence

How many of you use Copilot?

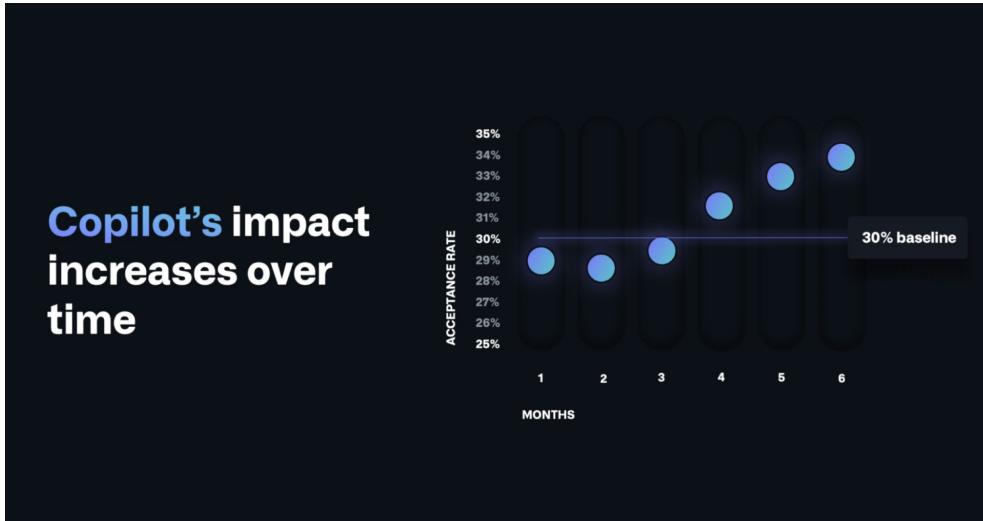
```
"""An efficient quick sort program"""

def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1

    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

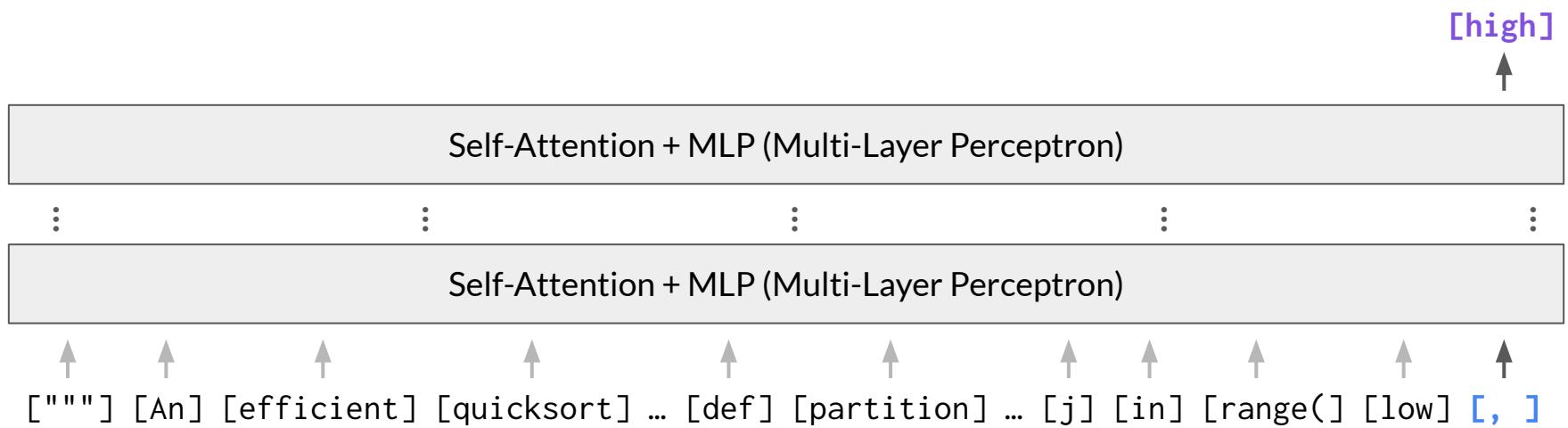
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1
```

# Language models (LMs) aids code intelligence

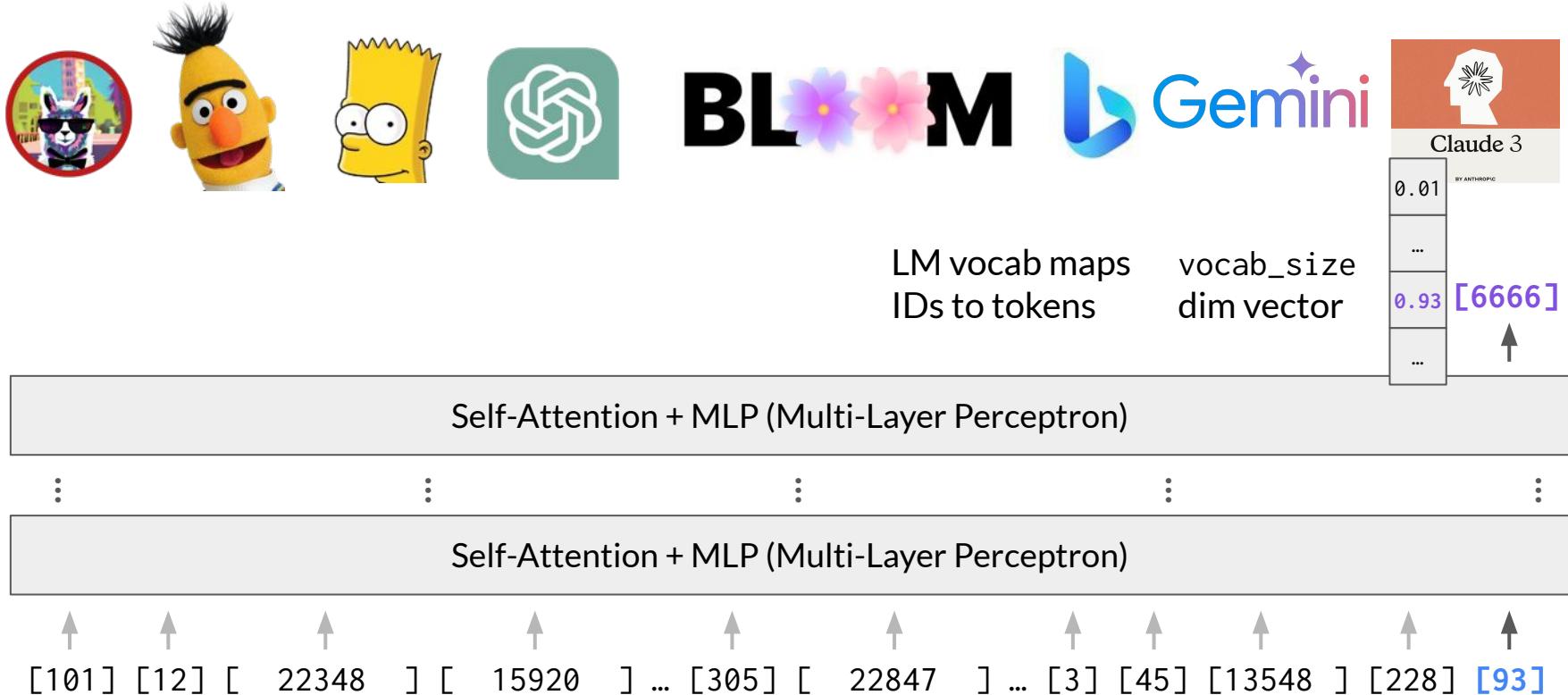


"GitHub Copilot has been activated by more than **one million developers** and adopted by over **20,000 organizations**. It has generated over **three billion accepted lines of code**, and is the world's most widely adopted AI developer tool."

# LM 101

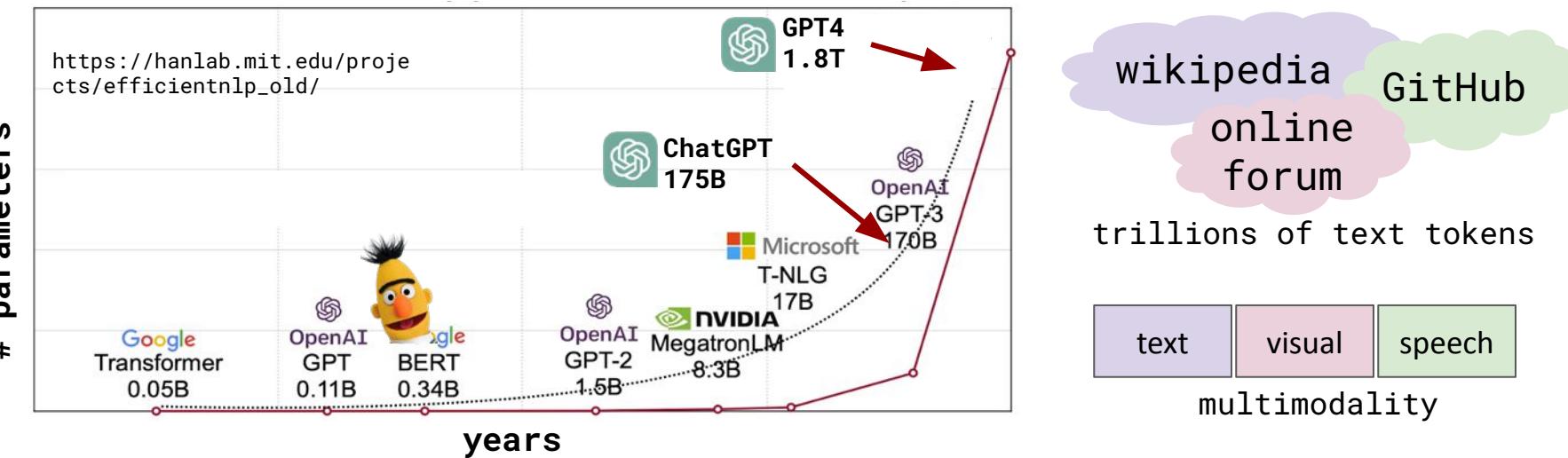


# LM 101



# What makes LMs powerful and scale?

- Huge number of model parameters
- Large amounts of unsupervised data for pre-training



# Building Code Intelligence with Language Models

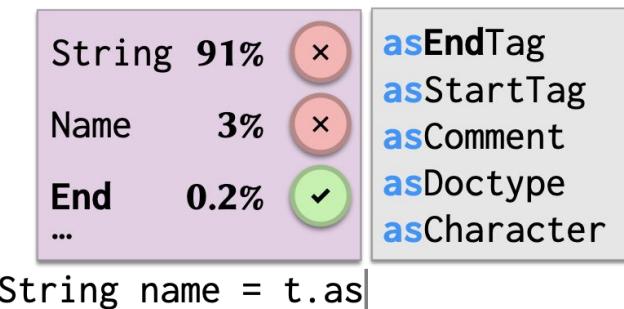
- How to *use* code LMs?
  - Repilot [FSE'23]
  - APR-LLM [ICSE'23]
- How to *build* code LMs through *posttraining*?
  - Magicoder [ICML'24]
  - SelfCodeAlign [NeurIPS'24]
- How to *build* code LMs through *pretraining*?
  - SnowCoder [arXiv]

# How to use code LMs – Repilot

- In the old days, we only rely on semantics-based, IDE autocompletions.
- Can autocomplete <-> LM synergistically improve each other?

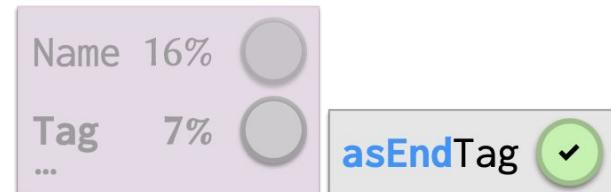
Language model

Predictions      Completions



String name = t.as|

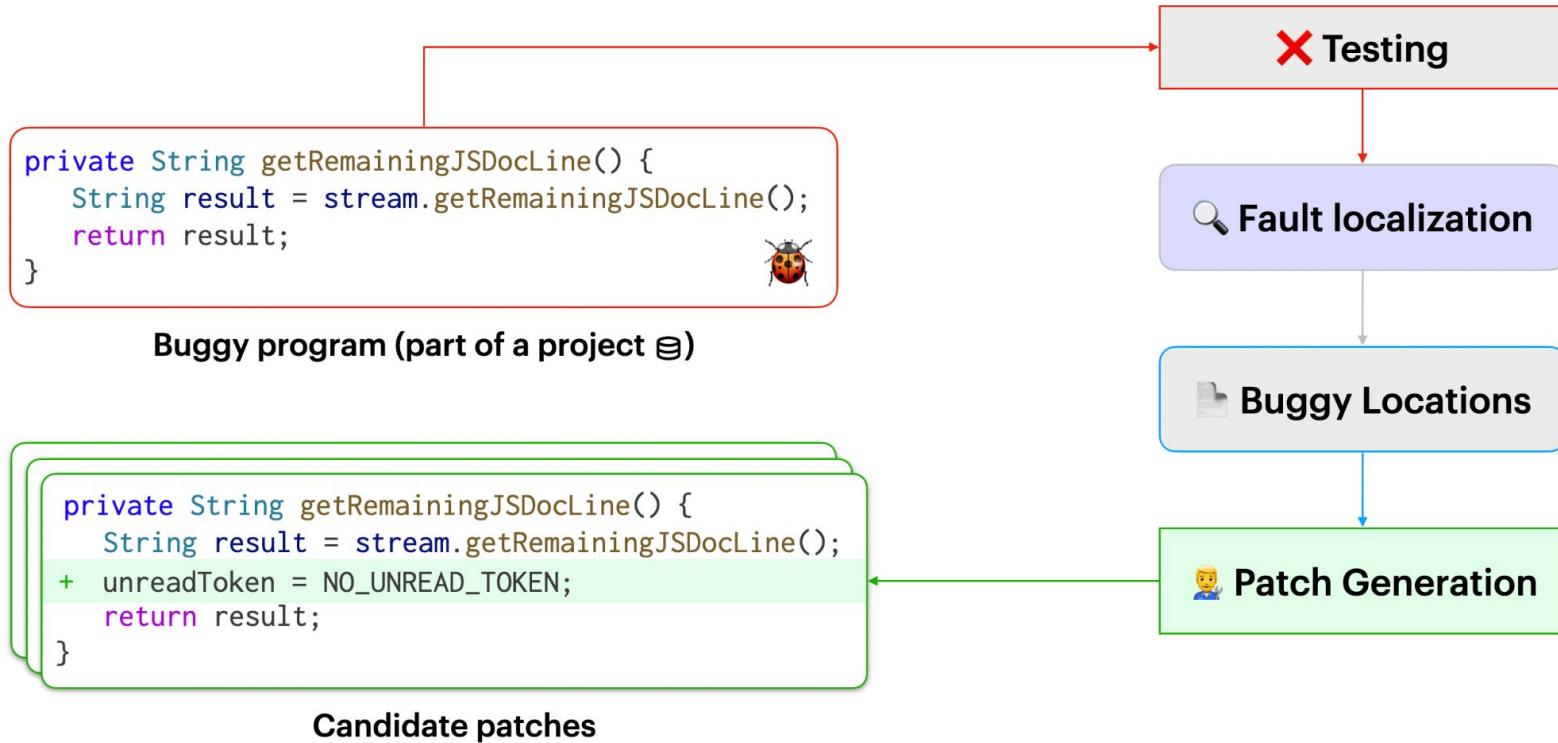
❶ Generating infeasible tokens



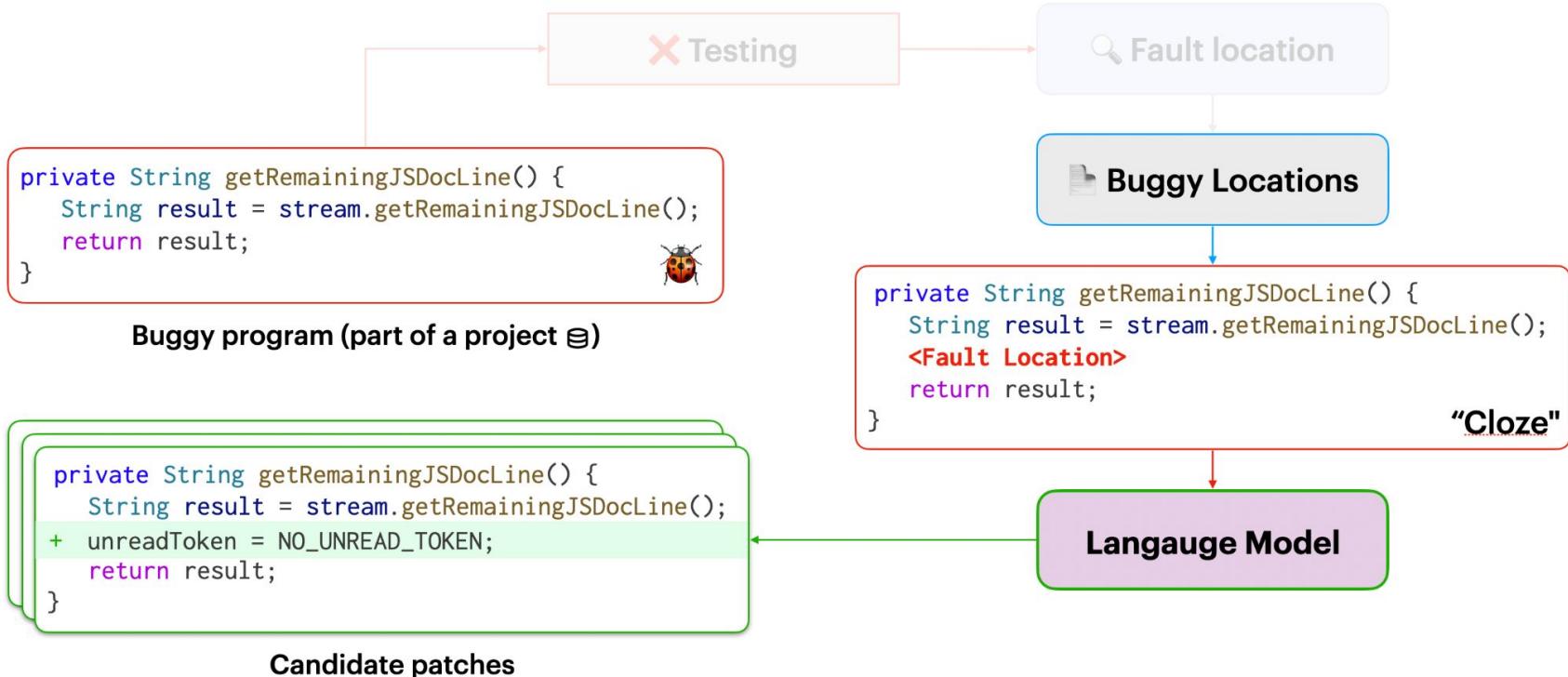
String name = t.asEndTag|

❷ Hard to generate rare tokens

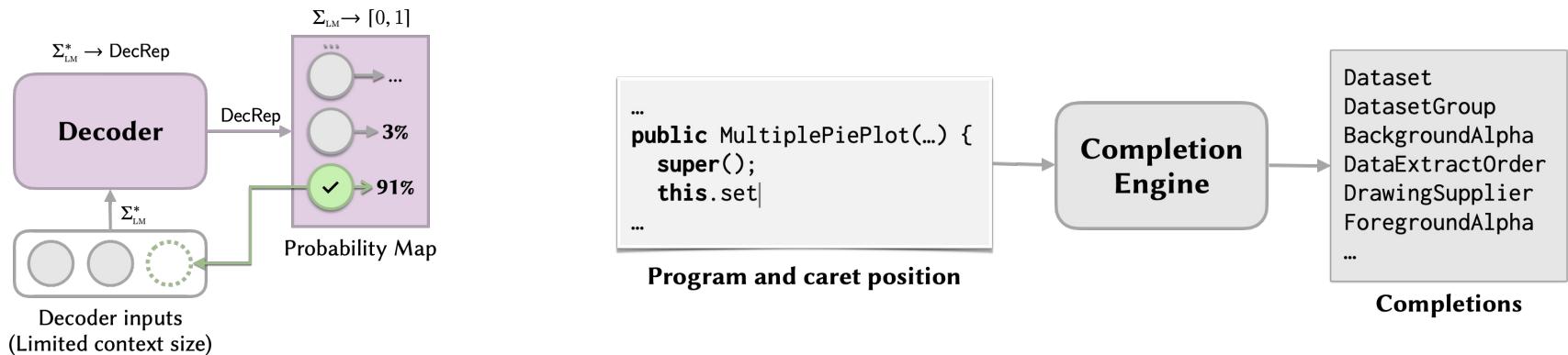
# Automated program repair (APR)



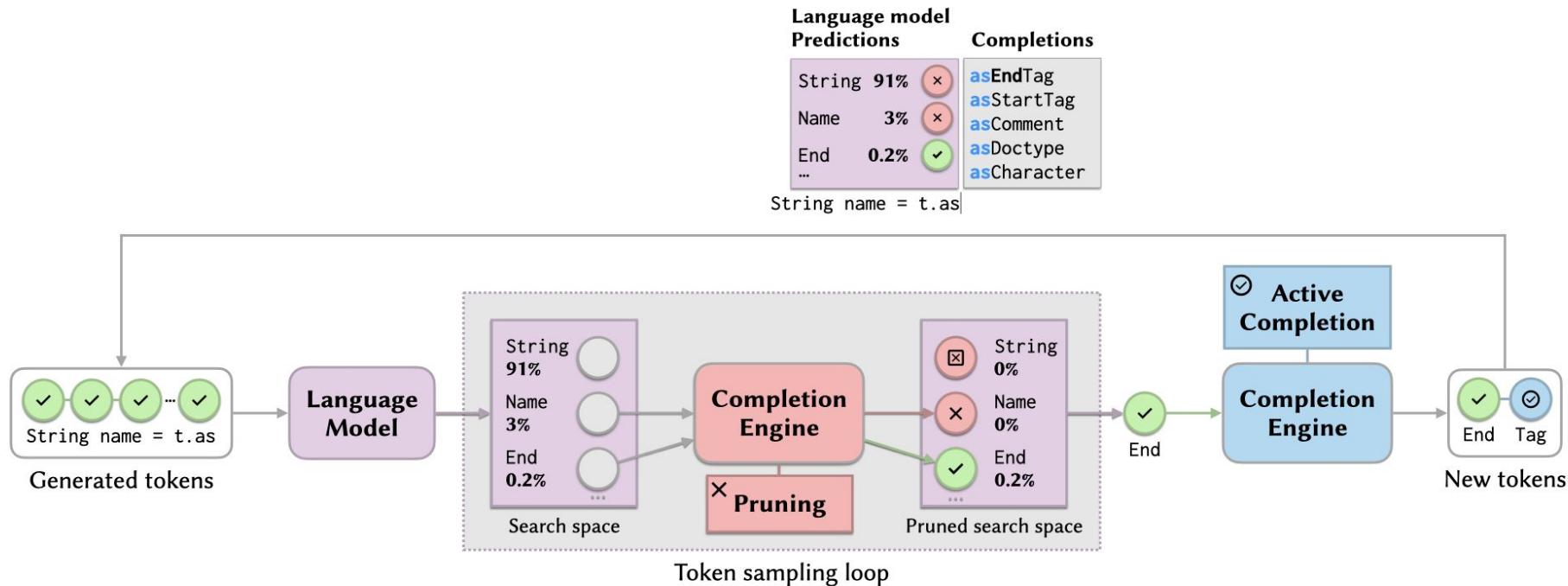
# Patch generation with LMs



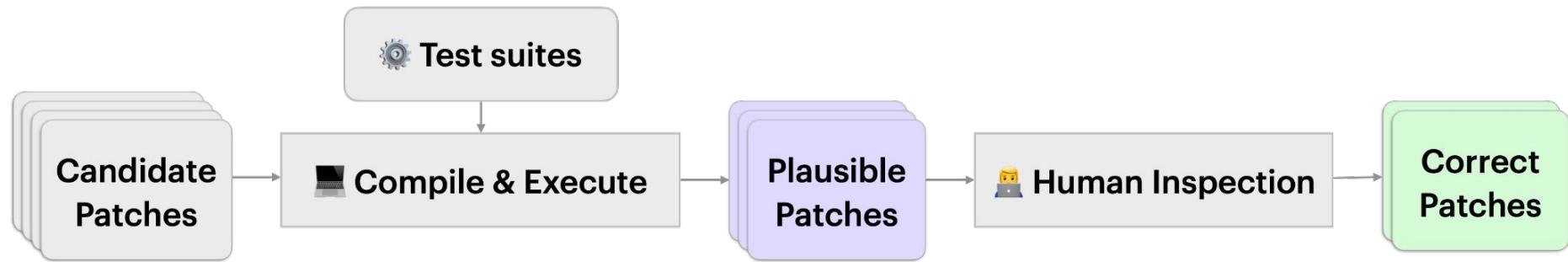
# LM completion vs. autocompletion



# How Repilot works

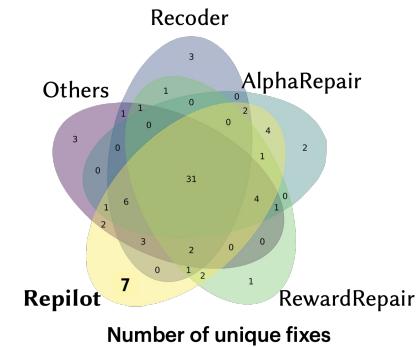


# Evaluation



# Evaluation: comparison with existing tools

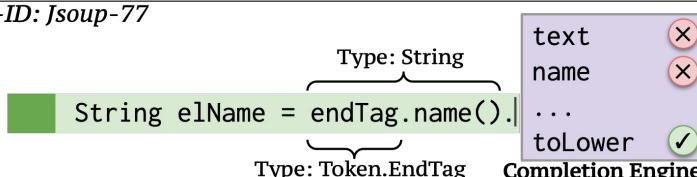
Tool	Methodology	#Correct Fixes		
		Defects4J 1.2	Defects4J 2.0	Total
CoCoNuT	NMT	30	-	-
DLFix	NMT	32	-	-
PraPR	Template	35	-	-
TBar	Template	41	7	48
CURE	NMT	43	-	-
RewardRepair	NMT	45	24	69
Recoder	NMT	51	10	61
AlphaRepair	LLM	52	34	86
Repilot	LLM	<b>66</b>	<b>50</b>	<b>116</b>



# Unique fixes generated by Repilot

```
- private void popStackToClose(Token.EndTag endTag) {  
+   String elName = endTag.name();  
+   String elName = endTag.name().toLowerCase();  
     Element firstFound = null;
```

Bug-ID: Jsoup-77



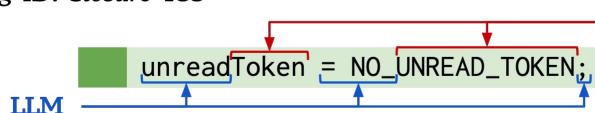
Patch Generation Process

Completion engine filters out invalid tokens

```
+ private String getRemainingJSLine() {  
  String result = stream.getRemainingJSLine();  
+   unreadToken = NO_UNREAD_TOKEN;  
  return result;  
}
```

Bug-ID: Closure-133

Completion Engine



Patch Generation Process

Interaction between LLM and completion engine

# Ablation study

Each component contributes positively to Repilot

Repilot Variants	Generation Time	%Compilable Patches	%Plausible Patches	#Plausible Fixes	#Correct Fixes
Base LLM (CodeT5)	0.232s	43.2%	3.95%	56	37
+ Pruning	0.294s	60.7%	5.02%	62	41
+ Memorization	0.255s	58.7%	4.82%	60	40
+ Active completion	0.248s	63.4%	5.21%	63	42

# Generalizability

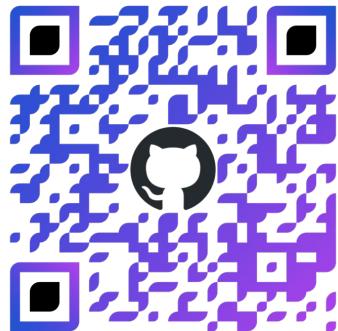
Repilot is generalizable across bug subjects and models

Variant	Model	Subject of Bugs	Generation Time	%Compilable Patches	#Correct Fixes
Base LLM	CodeT5-large	Defects4J 1.2	0.232s	43.2%	37
<b>Repilot</b>	<b>CodeT5-large</b>	<b>Defects4J 1.2</b>	<b>0.248s</b>	<b>63.4%</b>	<b>42</b>
Base LLM	CodeT5-large	Defects4J 2.0	0.230s	46.7%	41
<b>Repilot</b>	<b>CodeT5-large</b>	<b>Defects4J 2.0</b>	<b>0.247s</b>	<b>64.8%</b>	<b>45</b>
Base LLM	INCODER-6.7B	Defects4J 1.2	1.70s	32.4%	48
<b>Repilot</b>	<b>INCODER-6.7B</b>	<b>Defects4J 1.2</b>	<b>1.70s</b>	<b>47.2%</b>	<b>54</b>
Base LLM	INCODER-6.7B	Defects4J 2.0	1.67s	34.6%	45
<b>Repilot</b>	<b>INCODER-6.7B</b>	<b>Defects4J 2.0</b>	<b>1.69s</b>	<b>48.0%</b>	<b>46</b>

# Repilot

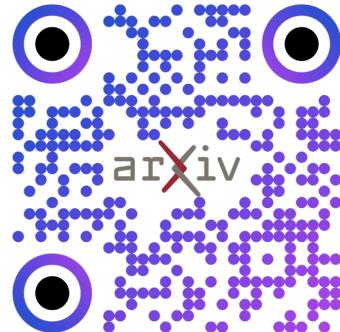
- Copiloting the Copilots
  - Fuses Large Language Models with Completion Engines for more effective Patch Generation in Automated Program Repair
  - Can be applied to general program synthesis

(★127) [ise-uiuc/Repilot](#)



Code & Docker & Artifact

[arXiv:2309.00608](#)



Paper

# First comprehensive study on LMs for APR

- Extensive evaluations using **9 different LMs**
- Use **5 different repair datasets** across **3 different programming languages** (Java, Python, and C).
- Compare LMs against each other using the **3 repair settings**.
- More in the paper!

# How to *build* code LMs through *posttraining*

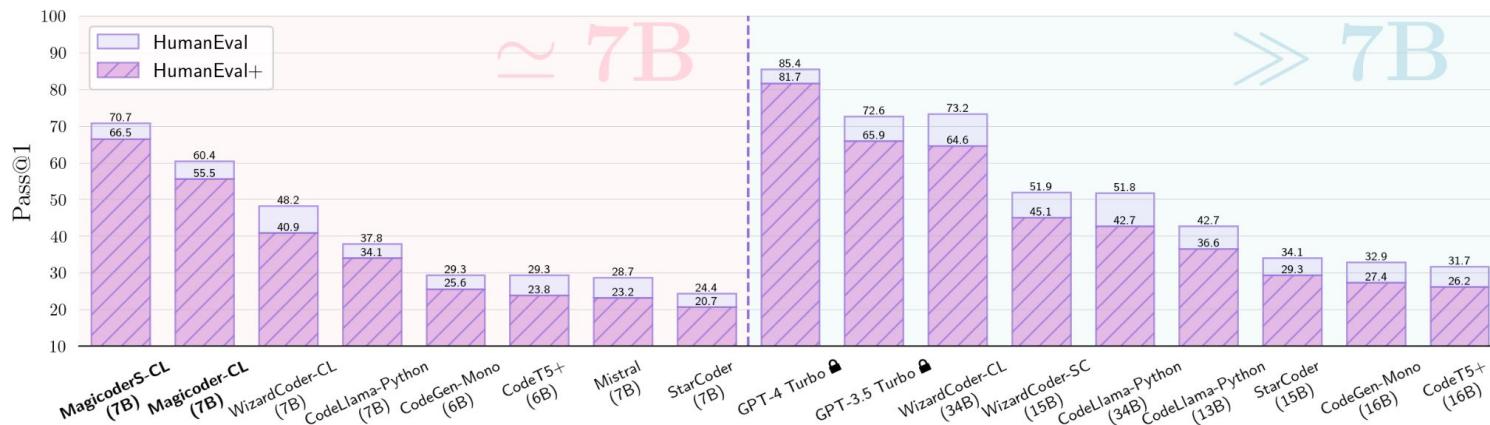
- **Magicoder [ICML'24]**
  - OSS-Instruct: instruction tuning from open source
- **SelfCodeAlign [NeurIPS'24]**
  - Self-improvement for code generation

**Yuxiang Wei**, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. *Magicoder: Empowering Code Generation with OSS-Instruct.* [ICML'24]

**Yuxiang Wei**, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro Von Werra, Arjun Guha, and Lingming Zhang, *Fully Transparent Self-Alignment for Code Generation,* [NeurIPS'24]

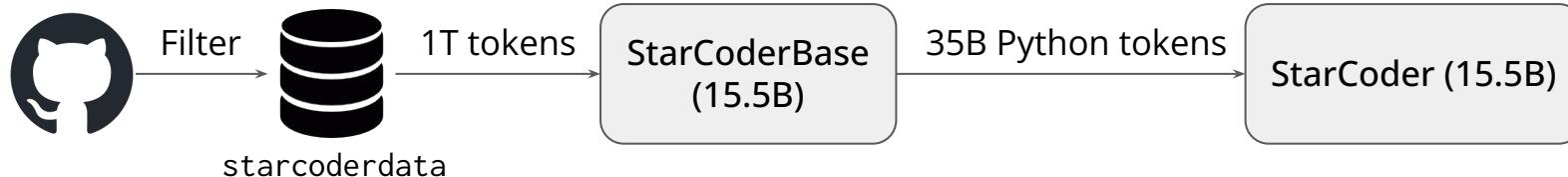
# Large language models for code

- Open source code LLMs are catching up with the best closed source ones (e.g., ChatGPT and GPT-4)!!
- **Instruction tuning (alignment)** plays a crucial role



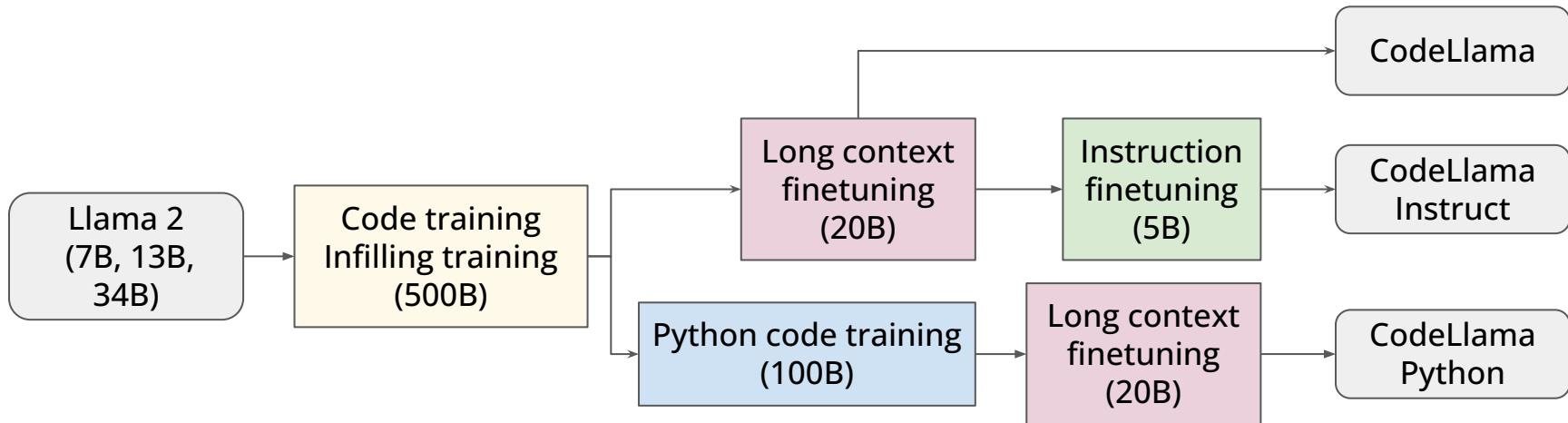
# Foundation code models

- StarCoder (15.5B)
  - Collect massive code from GitHub: The Stack
    - 6TB permissive code data
  - Filtering and cleaning: starcoderdata
    - 783GB of code in 86 programming languages
  - StarCoderBase (1T training tokens); StarCoder (+35B Python tokens)



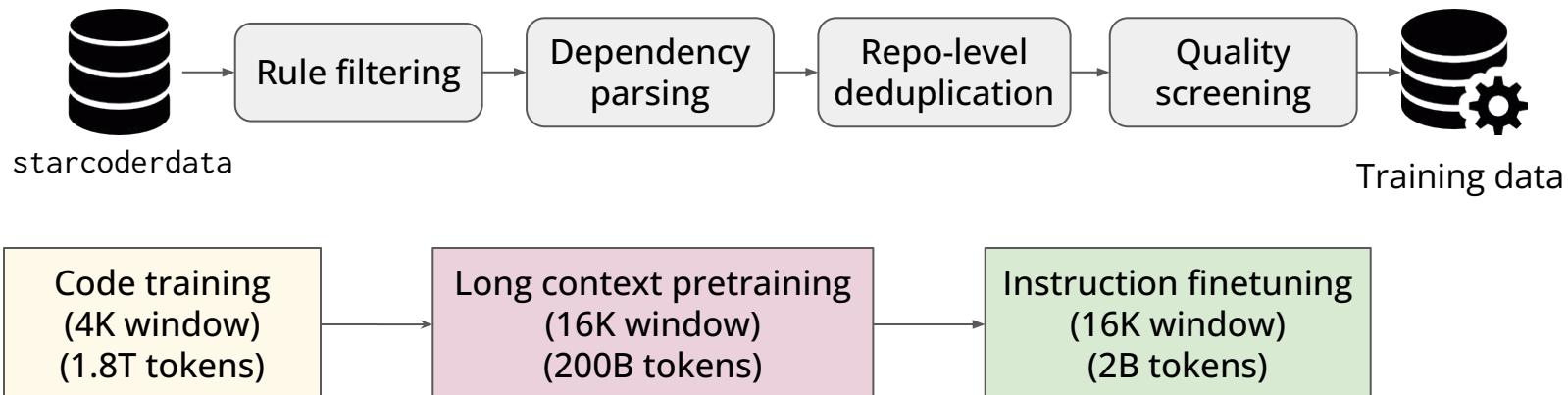
# Foundation code models

- CodeLlama (7B, 15B, 34B)
  - Continue pretraining Llama 2 on code and natural language + code data
  - CodeLlama (500B tokens); CodeLlama-Python (+100B Python tokens)



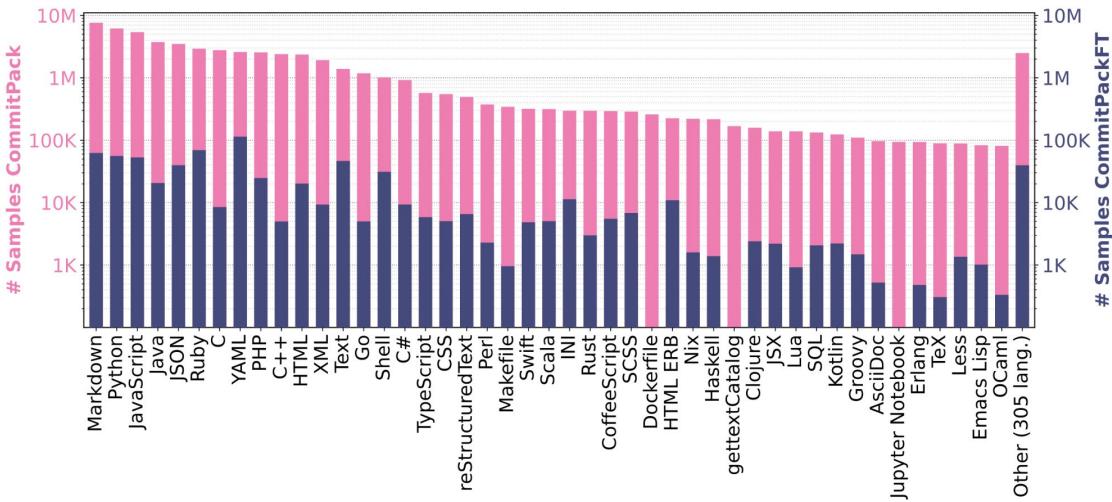
# Foundation code models

- Deepseek-Coder (1.3B, 6.7B, 33B)
  - Same training data as StarCoder, but **concatenates dependent files** to form a single training example
  - Trained on 2T tokens



# Instruction tuning for code

- Collect real-world instructions
  - OctoCoder uses GitHub commits to instruction-tune StarCoder and achieves **46.2 HumanEval pass@1** (13pp higher than the base StarCoder)



Source: OctoPack: Instruction Tuning Code Large Language Models

# Instruction tuning for code

- Self-generated instructions
  - CodeLlama-Instruct applies Llama-2 to generate coding problems and CodeLlama to generate solutions.
  - Modest Impact: 4pp improvements over 7B and 2pp over 13B base models (results for the 34B model not reported). **42.7 HumanEval pass@1 (13B)**

**Prompt:** [INST] Write 50 programming interview questions of easy and medium complexity. Provide questions on a diverse range of subjects, and make sure no two questions are alike. Make sure the problems can be solved with a single standalone Python function using standard libraries. [/INST]

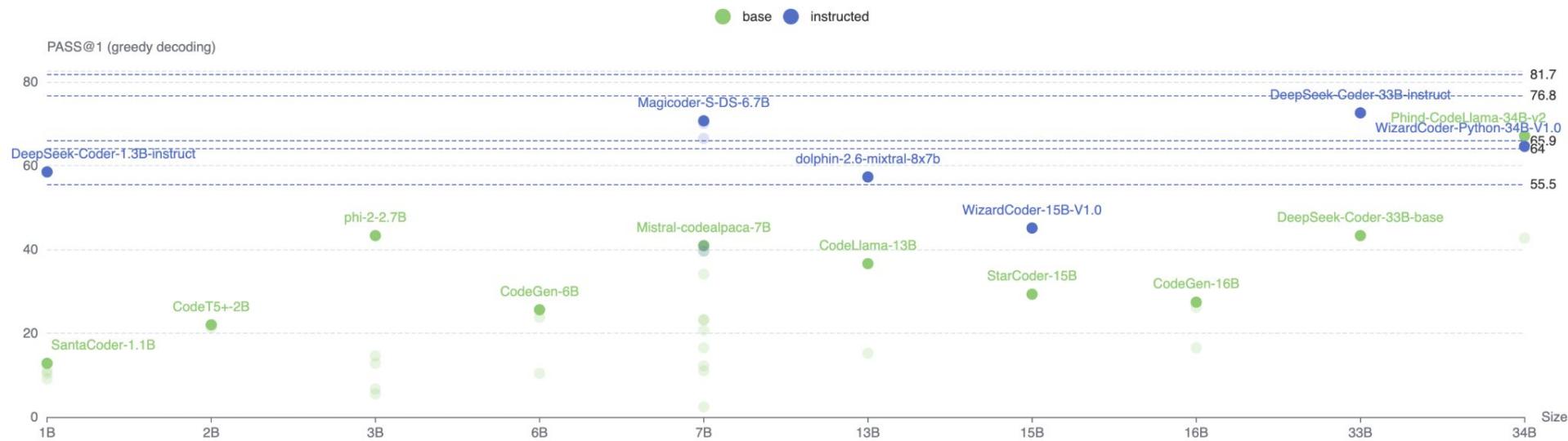
1. Write a function that finds the maximum depth of list nesting in a given list.
2. Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.
3. Write a function that gets the musical pitch that is closest to a given frequency in Hz. A pitch should consist of a letter, possibly a # sign, and an octave number.
4. Write a function that removes any sequences of whitespace that are between numbers in an input string.
5. Write a function that counts the number of words in a string that have length n, where n is an input. The function should ignore characters that aren't letters, numbers, or spaces.
6. Write a function that returns the longest palindromic substring in a given string.
7. Create a function that will find the length of the longest substring without repeating characters in a given string.
8. Write a function that reverses the digits in a number.
9. Write a function that finds the valleys in a list of integers.
10. Write a python function that reverses every group of k words in a sentence.

# Instruction tuning for code

- Distill knowledge from teacher models (e.g., ChatGPT)
  - Self-Instruct: use ChatGPT to generate synthetic code instructions
    - Representative dataset: Code Alpaca
  - WizardCoder (Evol-Instruct): **73.2 HumanEval pass@1 (34B)**
    - 20pp improvement over base model
  - Magicoder (OSS-Instruct + Evol-Instruct): **76.7 HumanEval pass@1 (6.7B)**
    - 30pp improvement over base model
- More discussions in the next few slides

# Summary of EvalPlus Leaderboard

HumanEval+ performance of different LLMs for code



# Distill synthetic code instructions

- **Self-Instruct (Code Alpaca)**
  - Relies on **21 seed tasks as few-shot examples** to generate new code instructions using an identical prompt template.

## Prompt Template for Self-Instruct

You are asked to come up with a set of 20 diverse code generation task instructions. These task instructions will be given to a GPT model and we will evaluate the GPT model for completing the instructions.

Here are the requirements:

1. Try not to repeat the verb for each instruction to maximize diversity.

...

List of 20 tasks:

{seeds}

# Distill synthetic code instructions

- **Code Evol-Instruct (WizardCoder)**
  - Takes Code Alpaca as seeds and use **5 heuristics** to increase the complexity of the data

## Prompt Template for Code Evol-Instruct

Please increase the difficulty of the given programming test question a bit.

You can increase the difficulty using, but not limited to, the following methods:

**{heuristic}**

{question}

# Distill synthetic code instructions

- Limitations of Self-Instruct & Code Evol-Instruct
  - Depends on a fixed set of seed tasks and heuristics
  - Generated data can **inherit the LLM's system bias**

## Prompt Template for Self-Instruct

You are asked to come up with a set of 20 diverse code generation task instructions. These task instructions will be given to a GPT model and we will evaluate the GPT model for completing the instructions.

Here are the requirements:

...

List of 20 tasks:  
**{seeds}**

## Prompt Template for Code Evol-Instruct

Please increase the difficulty of the given programming test question a bit.

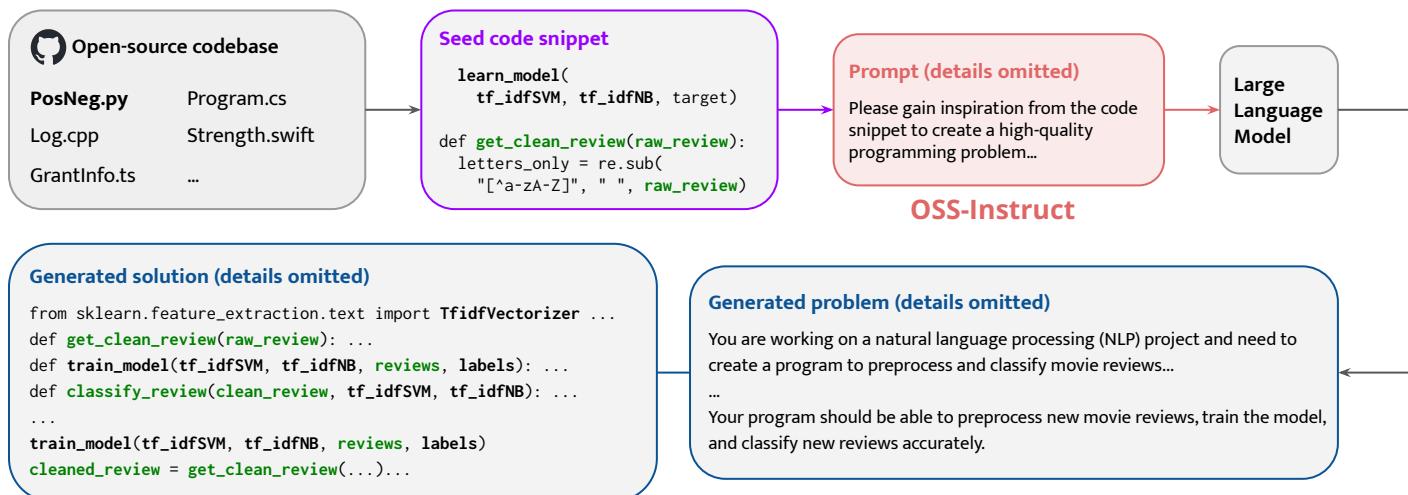
You can increase the difficulty using, but not limited to, the following methods:  
**{heuristic}**

{question}



# OSS-Instruct

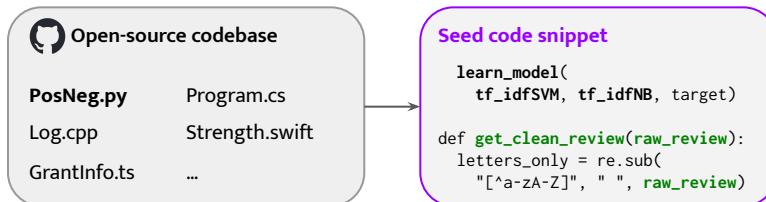
- LLMs get inspired from open source
  - Create **high-quality** and **low-bias** coding problems
  - Orthogonal to existing methods





# OSS-Instruct: data collection

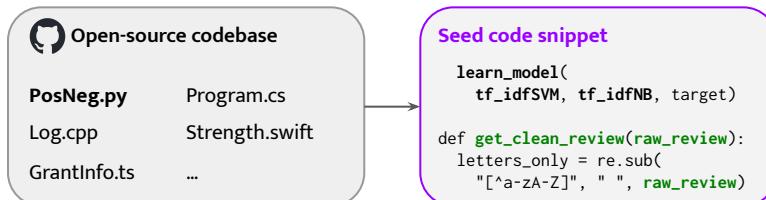
- Codebase: StarCoder training data (starcoderdata)
  - 783GB of code in 86 programming languages
  - Includes 54GB GitHub Issues + 13GB Jupyter notebooks
  - 32GB of GitHub commits





# OSS-Instruct: data collection

- Seed snippet collection
  - Randomly extract 1–15 consecutive lines from each document
  - 80K intotal (75K after cleaning): 40K from Python, and 5K from each of C++, Java, TypeScript, Shell, C#, Rust, PHP, and Swift respectively





# OSS-Instruct: prompt design

- Ask ChatGPT (gpt-3.5-turbo) to get inspired from the code

You are exceptionally skilled at crafting high-quality programming problems and offering precise solutions.

Please gain inspiration from the following random code snippet to create a high-quality programming problem. Present your output in two distinct sections: **[Problem Description]** and **[Solution]**.

Code snippet for inspiration:

{code}

Guidelines for each section:

- [Problem Description]** : This should be **\*\*completely self-contained\*\***, providing all the contextual information one needs to understand and solve the problem. Assume common programming knowledge, but ensure that any specific context, variables, or code snippets pertinent to this problem are explicitly included.
- [Solution]** : Offer a comprehensive, **\*\*correct\*\*** solution that accurately addresses the **[Problem Description]** you provided.

## Seed: method definition

```
render() {
```

### Problem

Your task is to complete the `render` method to generate the rendered shape as a string...

### Code

```
class ShapeRenderer {  
    constructor(vertices) {  
        this.vertices = vertices;  
    }  
    render() {  
        let renderedShape = "";  
        for (let i = 0; i < this.vertices.length; i++) {  
            const vertex = this.vertices[i];  
            renderedShape += `(${vertex.x}, ${vertex.y})`;  
            if (i < this.vertices.length - 1) {  
                renderedShape += " - ";  
            }  
        }  
        return renderedShape;  
    }  
}
```

## Seed: shell script

```
python3 makeErrorFile.py data/test_dataset_14 14
```

### Problem

Create a Python program that generates an **error file** based on a given **dataset**...

### Code

```
def generate_error_file(dataset_file, ...):  
    error_lines = []  
    with open(dataset_file, 'r') as file:  
        for line in file:  
            ...  
    with open(error_file_name, 'w') as error_file:  
        for error_line in error_lines:  
            error_file.write(error_line + '\n')  
if __name__ == "__main__":  
    if len(sys.argv) != 3:  
        print("Usage: ...")  
    else:  
        dataset_file = sys.argv[1]  
        dataset_number = sys.argv[2]  
        generate_error_file(...)
```

## Seed: library imports

```
import numpy as np
import gym_electric_motor as gem
import matplotlib.pyplot as plt
```

## Problem

Create a **reinforcement learning agent** to control an electric motor using the OpenAI **Gym** environment...

## Code

```
import numpy as np
import gym_electric_motor as gem
...
env = gem.make("DcSeriesCont-v1")
class DQNAgent:
    def __init__(self, state_dim, action_dim): ...
    def build_model(self): ...
    def act(self, state): ...
    def train(self, state, action, reward, ...): ...
...
for episode in range(episodes):
    state = env.reset()
    state = np.reshape(state, [1, state_dim])
    ...

```

## Seed: class signature

```
@SpringBootApplication
@Import({ AxonConfig.class })
public class AxonbankApplication {
    public static void main(String[] args) {
```

## Problem

Create a simple Java **Spring Boot application** for a **banking system**...

## Code

```
import org.axonframework.commandhandling.CommandBus;
import org.axonframework.config.Configuration;
...
@SpringBootApplication
@Import({ AxonConfig.class })
public class AxonbankApplication {...}
public class BankAccount {...}
public class CreateAccountCommand {...}
public class DepositFundsCommand {...}
public class WithdrawFundsCommand {...}
public class AccountCreatedEvent {...}
public class FundsDepositedEvent {...}
public class FundsWithdrawnEvent {...}
```

### Seed: code statements

```
cutoff_range = np.ptp(cutoffs)
if cutoff_range == 0: cutoff_range = 1
cutoff_colors = plt.get_cmap('plasma')(
    (cutoffs - np.min(cutoffs)) / cutoff_range
)
```

### Problem

Implement a function that calculates the color values for a given set of **cutoff** values based on a specified **color map**...

### Code

```
import numpy as np
import matplotlib.pyplot as plt
def calculate_cutoff_colors(cutoffs, cmap_name):
    cutoff_range = np.ptp(cutoffs)
    if cutoff_range == 0:
        cutoff_range = 1
    cmap = plt.get_cmap(cmap_name)
    normalized_cutoffs = ...
    cutoff_colors = ...
    return cutoff_colors
```

### Seed: comments

```
# Set degrees
```

### Problem

Implement a Python class that represents a **temperature in degrees**...

### Code

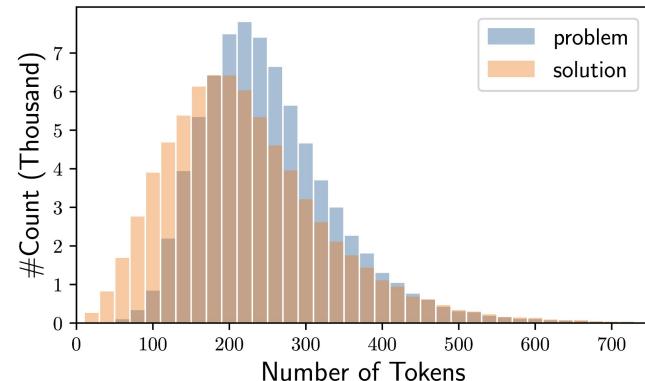
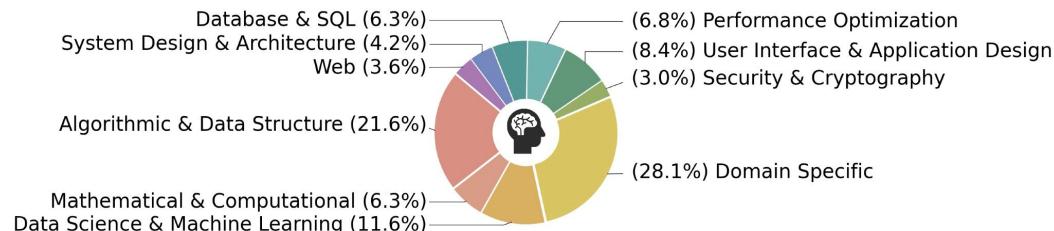
```
class TemperatureConverter:
    def __init__(self): ...
    def set_celsius(self, degrees): ...
    def set_fahrenheit(self, degrees): ...
    def set_kelvin(self, degrees): ...
    def get_celsius(self): ...
    def get_fahrenheit(self): ...
    def get_kelvin(self): ...
    def convert_to(self, unit):
        if unit == 'C':
            return self.get_celsius()
        elif unit == 'F':
            return self.get_fahrenheit()
        elif unit == 'K':
            return self.get_kelvin()
        ...
```

# Training Magicoder and Magicoder-S

- Base models: CodeLlama-Python-7B and DeepSeek-Coder-6.7B-Base
- Base model + 75K OSS-Instruct  Magicoder
  - Generated by gpt-3.5-turbo-1106
- Magicoder + 110K Evol-Instruct  Magicoder-S
  - Decontaminated from [evol-codealpaca-v1](#), an open-source reproduction of Evol-Instruct

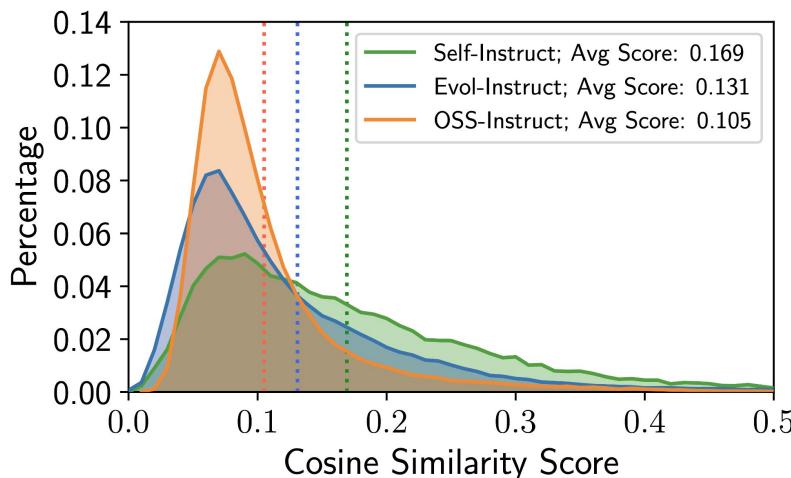
# Overview of OSS-Instruct dataset

- **Problem types:** Categorized using the SOTA instruction-tuned embedding model [INSTRUCTOR](#) with manually designed queries
- **Length distribution** of the generated problems and solutions



# Similarity with HumanEval

- Measured using TF-IDF embedding
- Similarity to HumanEval: OSS-Instruct < Evol-Instruct < Self-Instruct



# Pass@1 results on HumanEval+

- Magicoder-S outperforms ChatGPT on HumanEval+

Model (Size)	HumanEval+	Model (Size)	#Training tokens	HumanEval+
GPT-3.5-Turbo (?)	65.9	DeepSeek-Coder Base (6.7B)	2T	39.6
GPT-4-Turbo (?)	81.7	DeepSeek-Coder Instruct (33B)	+2B	72.6
WizardCoder-SC (15B)	45.1	DeepSeek-Coder Instruct (6.7B)	+2B	70.1
WizardCoder-CL (7B)	40.9			
CodeLlama-Python (7B)	34.1			
Magicoder-CL (7B)	55.5	<b>Magicoder-DS (6.7B)</b>	+90M	<b>60.4</b>
Magicoder-S-CL (7B)	66.5	<b>Magicoder-S-DS (6.7B)</b>	+240M	<b>70.7</b>

# Multilingual evaluation on MultiPL-E

- Magicoder-CL significantly outperforms the base model
- Magicoder-S-CL (**7B**) reaches WizardCoder-CL (**34B**)

Model (Size)	Java	JavaScript	C++	PHP	Swift	Rust
CodeLlama-Python (34B)	40.2	41.7	41.4	40.4	35.3	38.7
<b>WizardCoder-CL (34B)</b>	<b>44.9</b>	<b>55.3</b>	<b>47.2</b>	<b>47.2</b>	<b>44.3</b>	<b>46.2</b>
WizardCoder-SC (15B)	35.8	41.9	39.0	39.3	33.7	27.1
CodeLlama-Python (7B)	29.1	35.7	30.2	29.0	27.1	27.0
Magicoder-CL (7B)	36.4	45.9	36.5	39.5	33.4	30.6
<b>Magicoder-S-CL (7B)</b>	<b>42.9</b>	<b>57.5</b>	<b>44.4</b>	<b>47.6</b>	<b>44.1</b>	<b>40.3</b>

# Impact of the language distribution

- Training on different languages boosts **overall** performance
  - Python data can boost non-Python performance and vice versa
  - Combining data from both sources achieves the best result!

Model (7B)	Finetuning Data	Python (HumanEval+)	Others (MultiPL-E)
CodeLlama-Python	-	34.1	29.6
Magicoder-CL	Python (43K)	47.6	32.7
Magicoder-CL	Others (32K)	44.5	<b>38.3</b>
Magicoder-CL	Both (75K)	<b>55.5</b>	37.8

# OSS-Instruct vs. direct finetuning

- Directly finetuning on open-source code may **harm** performance
  - 75K comment-function pairs data following CodeSearchNet
  - Data factuality is essential to code instruction tuning!

Base model:

CodeLlama-Python-7B

Finetuning Data	HumanEval+	MultiPL-E
Base model w/o finetuning	34.1	29.6
Comment-function pairs (75K)	47.6	24.1
OSS-Instruct (75K)	<b>55.5</b>	<b>37.8</b>



# Summarizing 🎩 Magicoder

- Empower Code Generation with 🪄 OSS-Instruct
  - Inspire LLMs with open-source code snippets for data generation
  - Generate high-quality and low-bias data

Google



IBM

∞ Meta AI2 Allen Institute for AI

Kwai

snowflake

★ 2,000

⬇ 300K

**Problem description generation:** First, we generate a large collection of programming problem descriptions that span a diverse range of topics, including those in the long tail distribution. To achieve this diversity, we sample random code snippets from various sources and prompt the model to generate programming problems inspired by these examples. This allowed us to tap into a wide range of topics and create a comprehensive set of problem descriptions (Wei et al., 2024).

∞ Meta  
Llama 3.1

# SelfCodeAlign: powering StarCoder2-Instruct

Check out the blog post: <https://huggingface.co/blog/sc2-instruct>

The screenshot shows the GitHub repository page for StarCoder2-Instruct. The title of the repository is "StarCoder2-Instruct: Fully Transparent and Permissive Self-Alignment for Code Generation". It was published on April 29, 2024. The page features a dark background with a starry space theme, including a yellow star with code symbols (</>) and planet icons.

**StarCoder2-Instruct: Fully Transparent and Permissive Self-Alignment for Code Generation**

Published April 29, 2024

Update on GitHub

built by **BigCode**

Contributors:

- yuxiang630 Yuxiang Wei guest
- cassanof Federico Cassano guest
- ganlex Jiawei Liu guest
- YifengDing Yifeng Ding guest
- StringChaos Naman Jain guest
- harmdevries Harm de Vries guest
- lvwerra Leandro von Werra guest
- arjunguha Arjun Guha guest
- lingming Lingming Zhang guest

Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro Von Werra, Arjun Guha, and Lingming Zhang, *Fully Transparent Self-Alignment for Code Generation*, [NeurIPS'24]

# How to *build* code LMs through *pretraining*

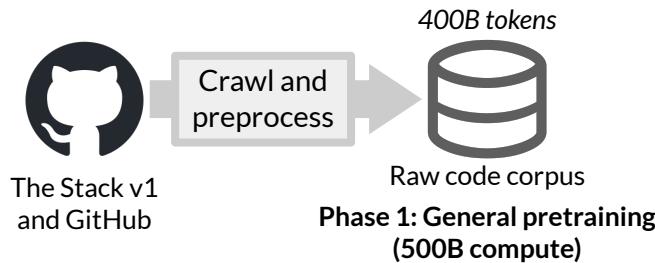
- **SnowCoder** [[arXiv](#)]
  - We all know that high-quality data is important, but what does it mean?

# Prerequisites of (code) model pretraining

- **Infra**
  - Hardware
    - Many A/H100s + memory + CPUs and a stable GPU cluster
    - Large volumes of cloud storage (e.g., aws s3 buckets and database)
  - Software
    - Job scheduler (e.g., kubernetes and slurm)
    - Efficient pretraining library (e.g., Megatron-DeepSpeed and Megatron-LM)
    - Training environment/code with a multi-node setup (e.g., with NCCL)
- **Architecture**
  - Llama architecture w/ some hyperparameter changes
  - A trained tokenizer
- **Raw data**
  - The Stack + GitHub crawls

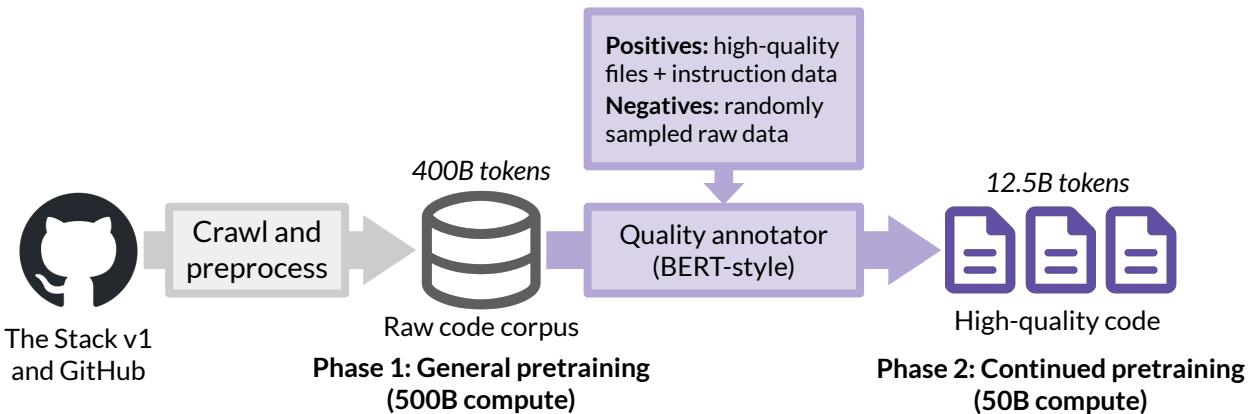
# Three phases of *progressively refined* data

1. 500B compute using 400B preprocessed raw tokens
  - a. Throughput: 1.5 days with 16 nodes



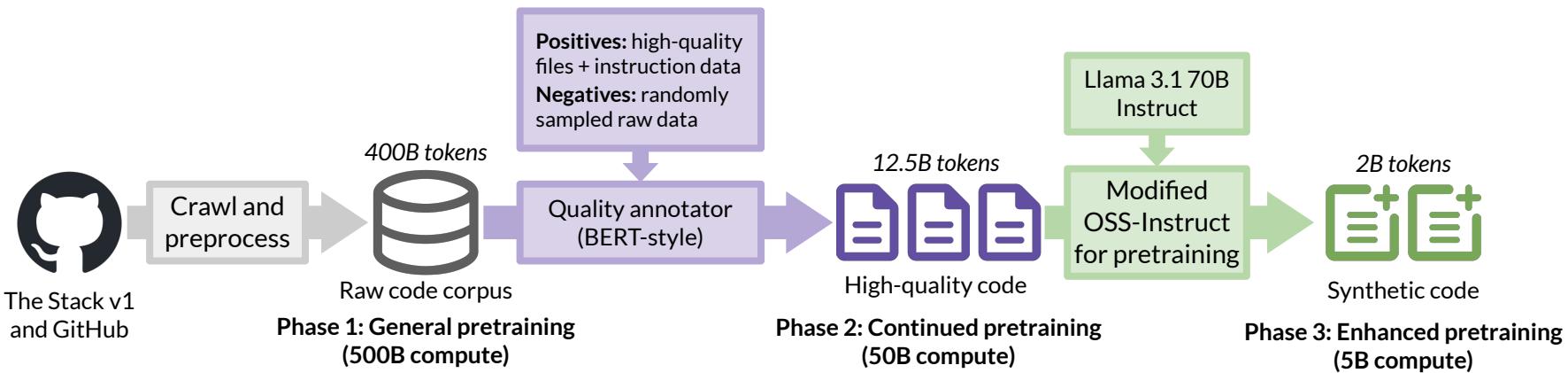
# Three phases of *progressively refined* data

1. 500B compute using 400B preprocessed raw tokens
2. 50B compute using 12.5B high-quality tokens (repeat x4), scored with a BERT-style annotator
  - a. Positive data: Magicoder + StarCoder2-Instruct + synthetic high-quality code files
  - b. Negative data: random sampled raw code



# Three phases of progressively refined data

1. 500B compute using 400B preprocessed raw tokens
2. 50B compute using 12.5B high-quality tokens (repeat x4), scored with a BERT-style annotator
3. 5B compute using 2B synthetic tokens generated w/ OSS-Instruct x Llama 3.1
  - a. Modified to generate high-quality code files using phase 2 data as seeds



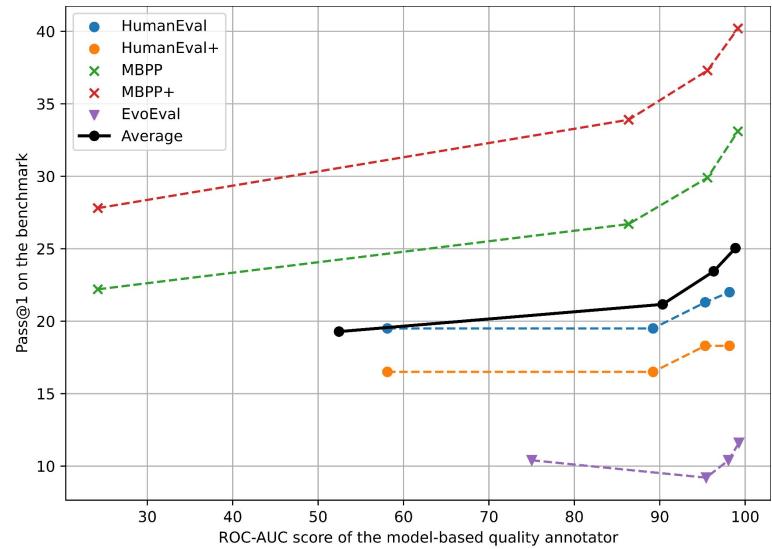
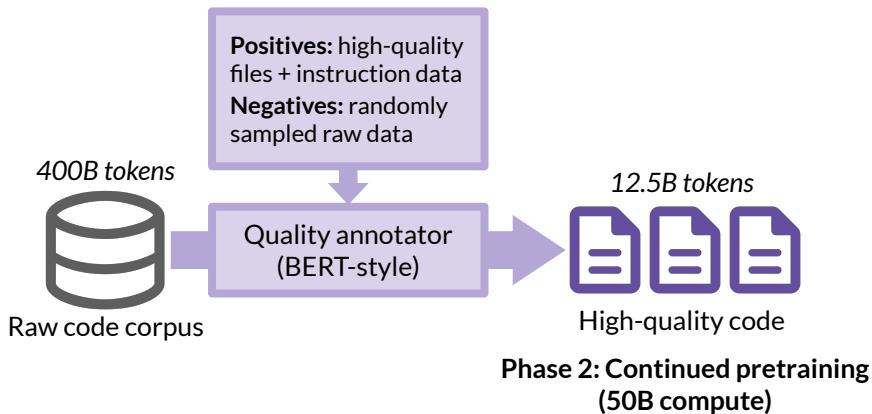
# Baseline comparison

- Each phase improves over the previous
- Decent performance across the 4 codegen benchmarks

Model	Training compute	HumanEval+	MBPP+	EvoEval	BigCodeBench
StableCode-3B [31]	1.3T	26.2	43.9	18.6	25.9
StarCoder2-3B [23]	3.3T to 4.3T	27.4	<b>49.2</b>	19.0	21.4
Granite-Code-Base-3B [27]	4.5T	29.3	45.8	<b>19.8</b>	20.0
CodeGemma-2B-v1.0 [35]	3T + 1T	18.3	46.3	15.4	23.9
CodeGemma-2B-v1.1 [35]	3T + 500B	<b>32.3</b>	48.9	<b>19.8</b>	<b>28.0</b>
Qwen1.5-1.8B <sup>1</sup> [43]	3T	19.5	28.3	5.0	6.3
Qwen2-1.5B <sup>1</sup> [43]	7T	31.1	38.4	17.2	16.5
DeepSeek-Coder-1.3B [14]	2T	28.7	<b>48.1</b>	19.2	22.2
StarCoderBase-3B [19]	1T	17.7	36.8	11.6	5.9
SmolLM-1.7B [2]	1T	15.9	34.7	10.0	2.5
Phi-1.5-1.3B [20]	150B	<b>31.7</b>	<b>43.7</b>	<b>20.6</b>	<b>14.3</b>
SnowCoder-alpha-1.3B	500B	14.0	27.8	7.4	10.3
SnowCoder-beta-1.3B	500B + 50B	21.3	34.7	12.8	12.3
SnowCoder-1.3B	550B + 5B	<b>28.0</b>	<b>42.9</b>	<b>18.0</b>	<b>19.4</b>

# High-quality = In-domain

- 4 BERT annotators trained with different data recipes
- X-axis: ROC-AUC score of each annotator on the corresponding benchmark
  - TL;DR, ROC-AUC evaluates how well a scorer can rank one distribution higher than the other
- Y-axis: benchmark pass@1



# Conclusion: how to get a *good* code pretraining corpus

- A huge raw code corpus from the stack, expecting 10x the final code tokens
- A small annotation model to obtain the 10%
  - The annotator should match the downstream distribution
- Synthetic data is all you need?
- More details in the paper: <https://arxiv.org/abs/2409.02326>

# Building Code Intelligence with Language Models

- How to *use* code LMs?
  - Repilot [FSE'23]
  - APR-LLM [ICSE'23]
- How to *build* code LMs through *posttraining*?
  - Magicoder [ICML'24]
  - SelfCodeAlign [NeurIPS'24]
- How to *build* code LMs through *pretraining*?
  - SnowCoder [arXiv]

 Yuxiang Wei, 3rd year PhD student at UIUC

 @YuxiangWei9

 <https://yuxiang.cs.illinois.edu>