

Data (and some modeling)

Daniel Fried

11-891: Neural Code Generation

<https://cmu-codegen.github.io/s2024/>

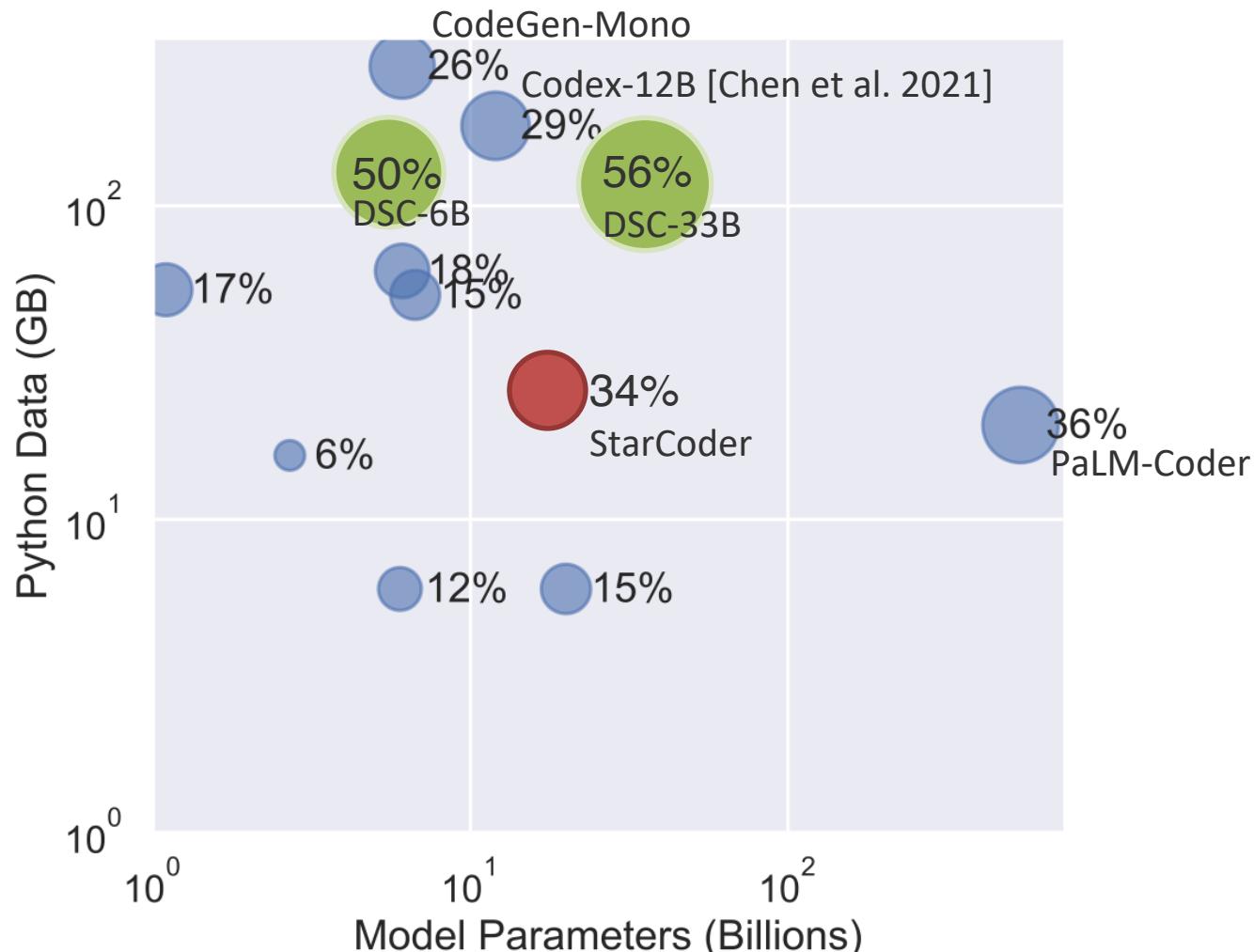


Language
Technologies
Institute

With slides from the BigCode project

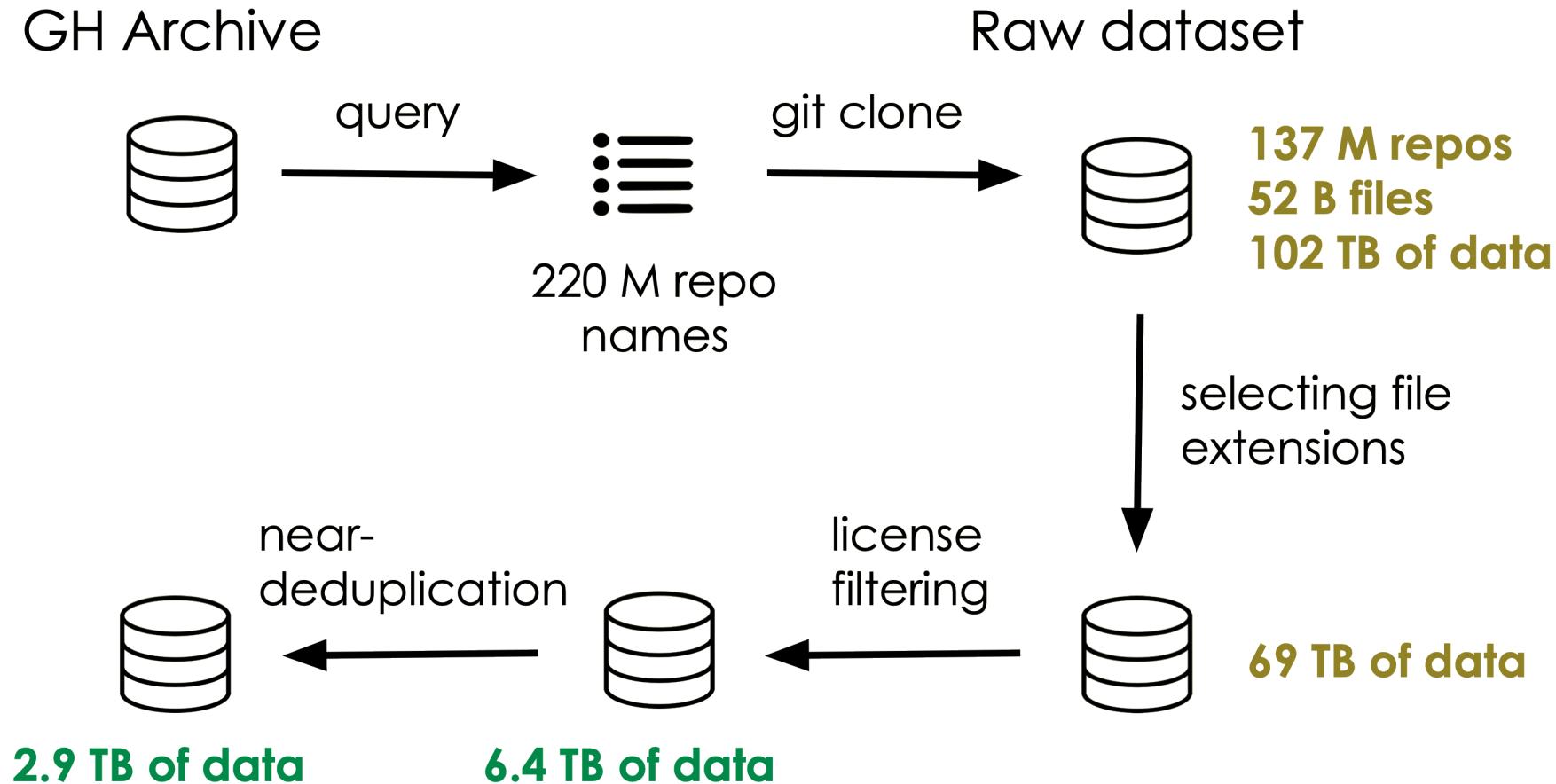
What Makes a Model Good?

Function pass rate on HumanEval Python [Chen et al. 2021] by amount of Python data & model scale:

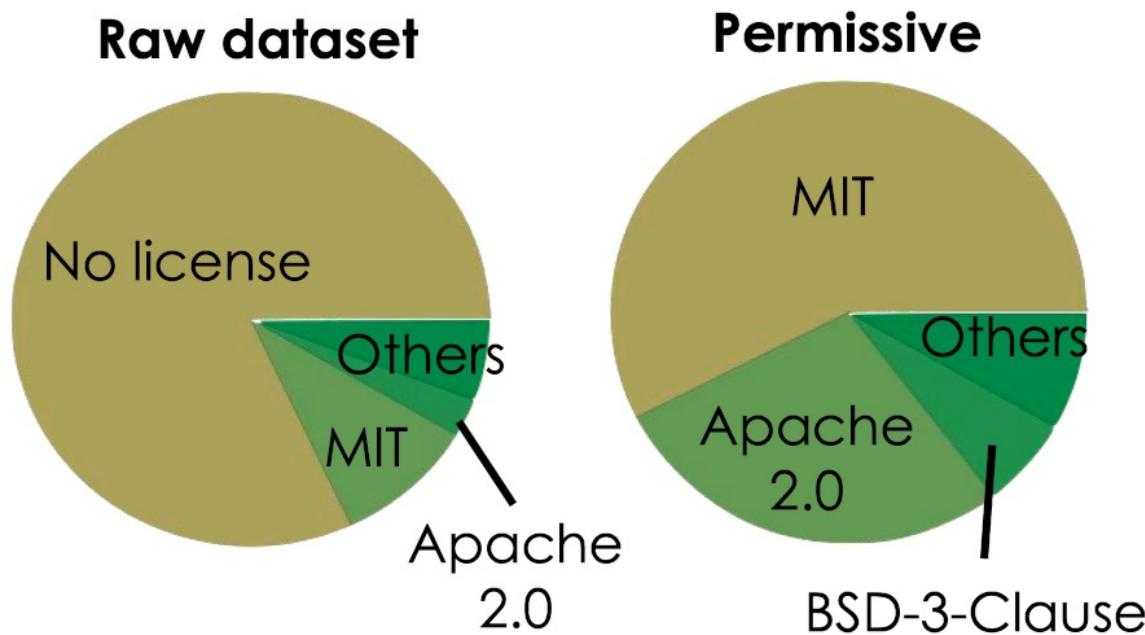


The Stack, SantaCoder, and StarCoder

The Stack: Dataset



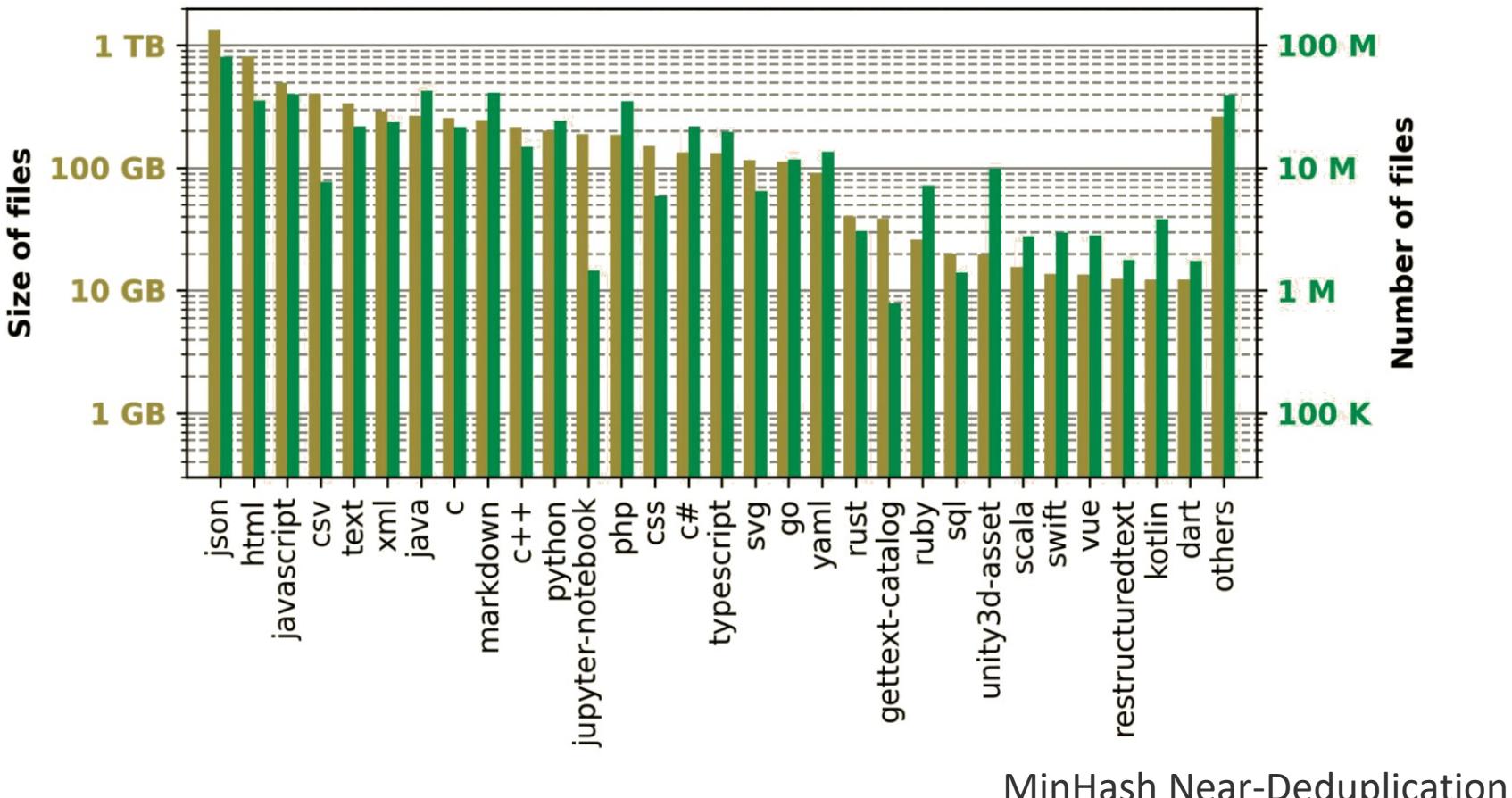
The Stack: Dataset



Permissive license distribution of licenses used to filter the dataset:

MIT (67.7%) | Apache-2.0 (19.1%) | BSD-3-Clause (3.9%) | Unlicense (2.0%) |
CC0-1.0 (1.5%) | BSD-2-Clause (1.2%) | CC-BY-4.0 (1.1%) | CC-BY-3.0 (0.7%) |
OBSD (0.4%) | RSA-MD (0.3%) | WTFPL (0.2%) | MIT-0 (0.2%) | Others (166) (2.2%)

The Stack: Dataset



The Stack: Python Models

- ▶ Possible to approximate Codex-12B performance with permissively licensed data? Train 350M models on Python
- ▶ ***Deduplication always improves performance***
(<https://huggingface.co/blog/dedup>)
- ▶ License filtering hurts, but there's enough data available to match Chen et al. 2021

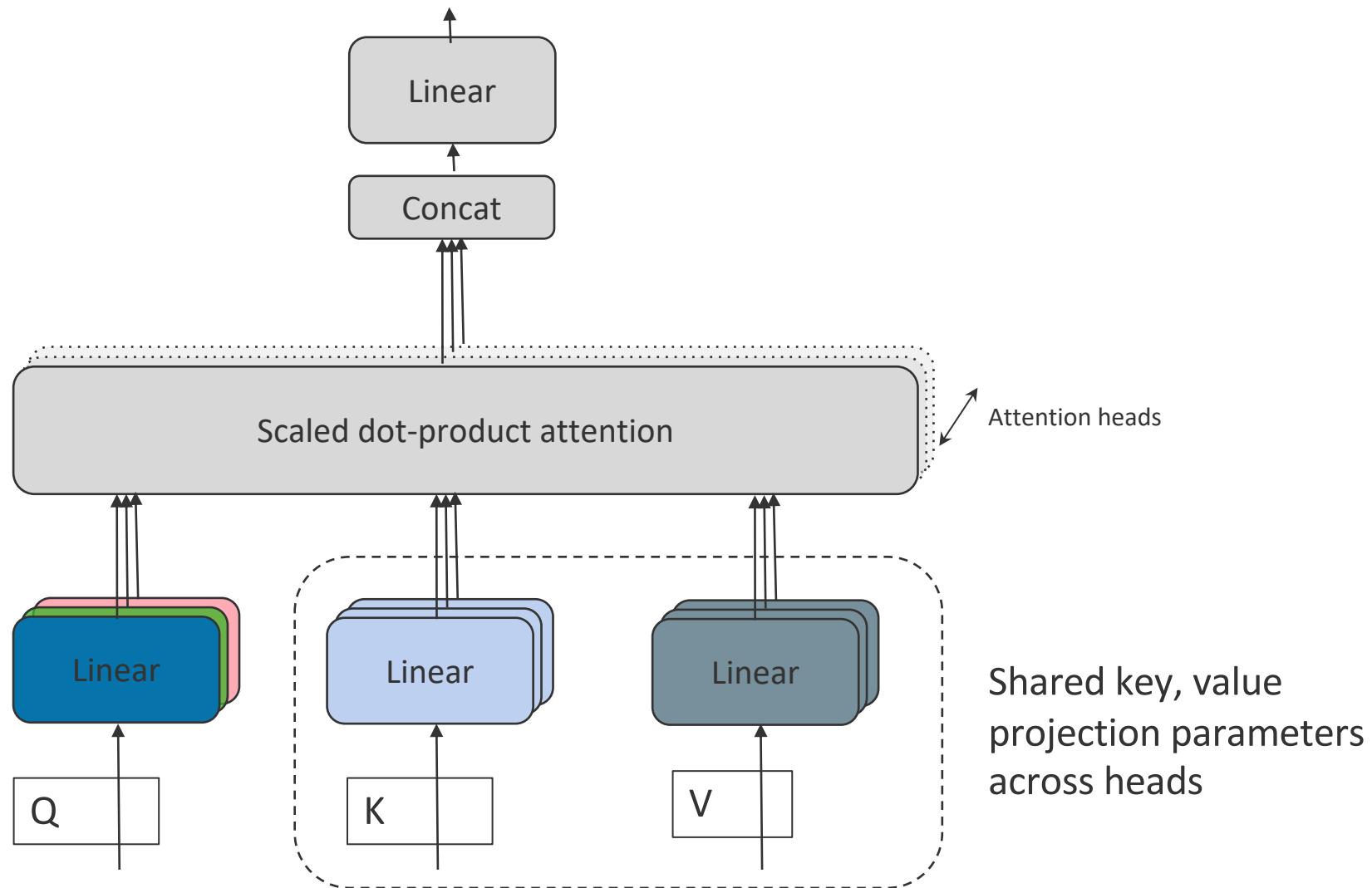
Dataset	Filtering	pass@1	pass@10	pass@100	Python Data
Codex (300M)	Exact-dedup?	13.17	20.17	36.27	180 GB
CodeGen (350M)	unknown	12.76	23.11	35.19	
Python all-license	None	13.11	21.77	36.67	740 GB
	Near-dedup	17.34	27.64	45.52	
Python permissive-license	None	10.99	15.94	27.21	191 GB
	Near-dedup	12.89	22.26	36.01	80 GB

SantaCoder: Overview

- ▶ Preparation for a big run: explorations at 1B scale
- ▶ Data: The Stack
- ▶ Tokenizer: BPE following GPT-2 recipe; use a digit splitter
- ▶ Ablations
 - ▶ Multi-query attention and infilling (FIM, Bavarian et al. 2022)
 - ▶ Data filtering

Multi-Query Attention

- Designed to reduce memory-bandwidth cost to speed up inference



SantaCoder: Model Ablations

- ▶ Infilling (FIM) and MQA “for cheap”

Language	Attention	FIM	HumanEval	MBPP
Java	Multi Query Attention	✓	0.35	0.54
	Multi Head Attention	✓	0.36	0.55
	Multi Query Attention	✗	0.37	0.55
JavaScript	Multi Query Attention	✓	0.33	0.64
	Multi Head Attention	✓	0.37	0.67
	Multi Query Attention	✗	0.37	0.65
Python	Multi Query Attention	✓	0.36	0.67
	Multi Head Attention	✓	0.38	0.70
	Multi Query Attention	✗	0.39	0.68

Table 5: Pass@100 results for the architecture ablations on HumanEval and MBPP.

SantaCoder: Data Filtering Ablations

- ▶ Remove repos with < 5 stars
 - **Hurts substantially!**
- ▶ Remove files with low (or very high) comment-to-code ratio
 - ~ Mixed effects
- ▶ More aggressive near-duplicate filtering
 - + **Very slight improvements**
- ▶ Remove files with low character-to-token ratios
 - + **Very slight improvements**

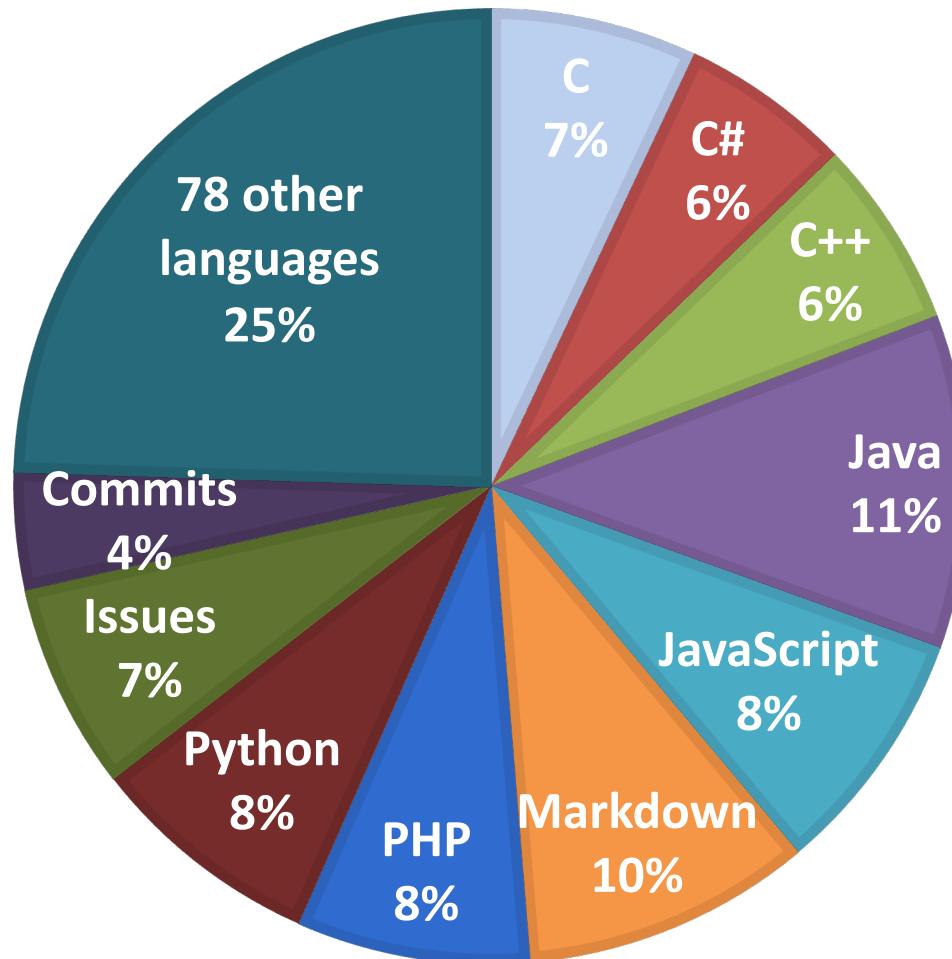
SantaCoder: Final Model

- ▶ 1B parameter, with infilling (FIM) and multi-query attention
- ▶ 268GB of data: 118B tokens. Java, JavaScript, Python
- ▶ ~6 days on 96 V100s

Model	Size	Left-to-right pass@100			Fill-in-the-middle ex. match		
		Java	JavaScript	Python	Java	JavaScript	Python
InCoder	6.7B	0.36	0.38	0.47	0.49	0.51	0.31
CodeGen-multi	2.7B	0.42	0.39	0.39	✗	✗	✗
CodeGen-mono	2.7B	✗	✗	0.57	✗	✗	✗
Codex ¹²	2.5B	✗	✗	0.60	✗	✗	✗
SantaCoder	1.1B	0.41	0.47	0.49	0.62	0.60	0.44

StarCoder: A Large-Scale Multilingual Model

We follow the natural distribution and sample data from 86 languages proportionally to their volume. **800GB total.** Lots of natural language (~20%)!

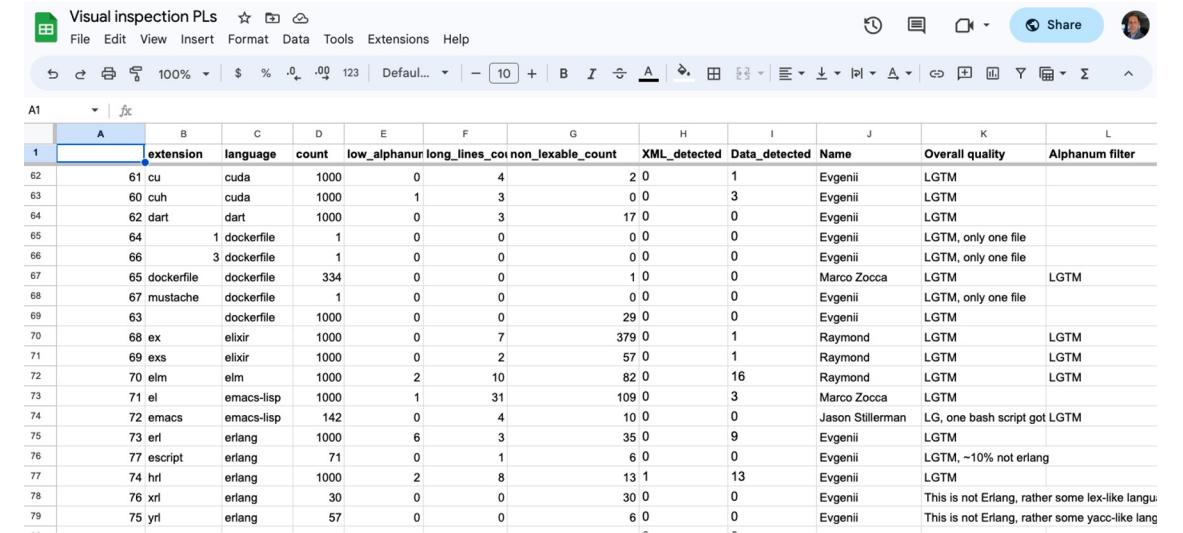


StarCoder: Data Selection

- ▶ Selected 86 languages to be used in training out of the 358 in the Stack
- ▶ **Included:**
 - ▷ Languages with more than 500 MB of data
 - ▷ Languages ranked within the top 50 by one of two commonly used rankings of language popularity.
- ▶ **Excluded:**
 - ▷ languages that are no longer actively supported
 - ▷ configuration languages
 - ▷ D and Swift (human error!)
- ▶ **Limited:**
 - ▷ Data formats like JSON and YAML
 - ▷ Long lines
 - ▷ Low alpha-numeric characters
 - ▷ HTML

StarCoder: Data Inspection

- ▶ Randomly selected 1000 files for each programming language extension
- ▶ 18 BigCode members verified 300 extensions
 - ▷ Excluded 36 extensions
 - ▷ Decide whether to keep or remove long line filter and alpha filter for each extension



The screenshot shows a spreadsheet titled "Visual inspection PLs" with a tab labeled "fjc". The table has 18 columns and 79 rows of data. The columns are labeled A through L, with headers including "extension", "language", "count", "low_alpha", "long_lines", "non_lexable_count", "XML_detected", "Data_detected", "Name", "Overall quality", and "Alphanum filter". The data includes various file types like cuda, dart, dockerfile, mustache, elixir, exs, elm, el, emacs-lisp, erl, escript, hrl, xrl, and yrl, along with their counts and detection metrics. Some rows have notes in the "Name" column, such as "LGTM, only one file" and "This is not Erlang, rather some yacc-like lang". The "Overall quality" column shows mostly "LGTM" entries.

A	B	C	D	E	F	G	H	I	J	K	L
1	extension	language	count	low_alpha	long_lines	non_lexable_count	XML_detected	Data_detected	Name	Overall quality	Alphanum filter
62	61	cu	cuda	1000	0	4	2 0	1	Evgenii	LGTM	
63	60	cuh	cuda	1000	1	3	0 0	3	Evgenii	LGTM	
64	62	dart	dart	1000	0	3	17 0	0	Evgenii	LGTM	
65	64	1	dockerfile	1	0	0	0 0	0	Evgenii	LGTM, only one file	
66	66	3	dockerfile	1	0	0	0 0	0	Evgenii	LGTM, only one file	
67	65	dockerfile	dockerfile	334	0	0	1 0	0	Marco Zocca	LGTM	LGTM
68	67	mustache	dockerfile	1	0	0	0 0	0	Evgenii	LGTM, only one file	
69	63	dockerfile	dockerfile	1000	0	0	29 0	0	Evgenii	LGTM	
70	68	ex	elixir	1000	0	7	379 0	1	Raymond	LGTM	LGTM
71	69	exs	elixir	1000	0	2	57 0	1	Raymond	LGTM	LGTM
72	70	elm	elm	1000	2	10	82 0	16	Raymond	LGTM	LGTM
73	71	el	emacs-lisp	1000	1	31	109 0	3	Marco Zocca	LGTM	
74	72	emacs	emacs-lisp	142	0	4	10 0	0	Jason Stillerman	LG, one bash script got LGTM	
75	73	erl	erlang	1000	6	3	35 0	9	Evgenii	LGTM	
76	77	escript	erlang	71	0	1	6 0	0	Evgenii	LGTM, ~10% not erlang	
77	74	hrl	erlang	1000	2	8	13 1	13	Evgenii	LGTM	
78	76	xrl	erlang	30	0	0	30 0	0	Evgenii	This is not Erlang, rather some lex-like langu	
79	75	yrl	erlang	57	0	0	6 0	0	Evgenii	This is not Erlang, rather some yacc-like lang	

Spreadsheet available!

PII dataset annotations

- ▶ Data composition
 - ▷ 12,000 code files
 - ▷ 7,000 pre-filtered to probably have PII, 5,000 random
 - ▷ 31 programming languages
- ▶ PII Annotation
 - ▷ 7 entities: Names, Usernames, Emails, IP addresses, keys, passwords, and IDs
 - ▷ 1,399 crowd-workers from Toloka

StarEncoder

- ▶ Model
 - ▷ Bidirectional Transformer similar to BERT-base
 - ▷ Same vocabulary as StarCoder
 - ▷ ~125M params
 - ▷ <https://huggingface.co/bigcode/starencoder>
- ▶ Pre-training
 - ▷ Follows data mix of StarCoder
 - ▶ Commits and Issues included
 - ▷ Trained for 400B tokens
 - ▶ Masked language modeling + next “sentence” prediction objective
 - ▷ <https://github.com/bigcode-project/bigcode-encoder>

PII Models

Named entity recognition (NER) training

Fine-tune StarEncoder with a linear classification
tagging layer on 6 PII target classes

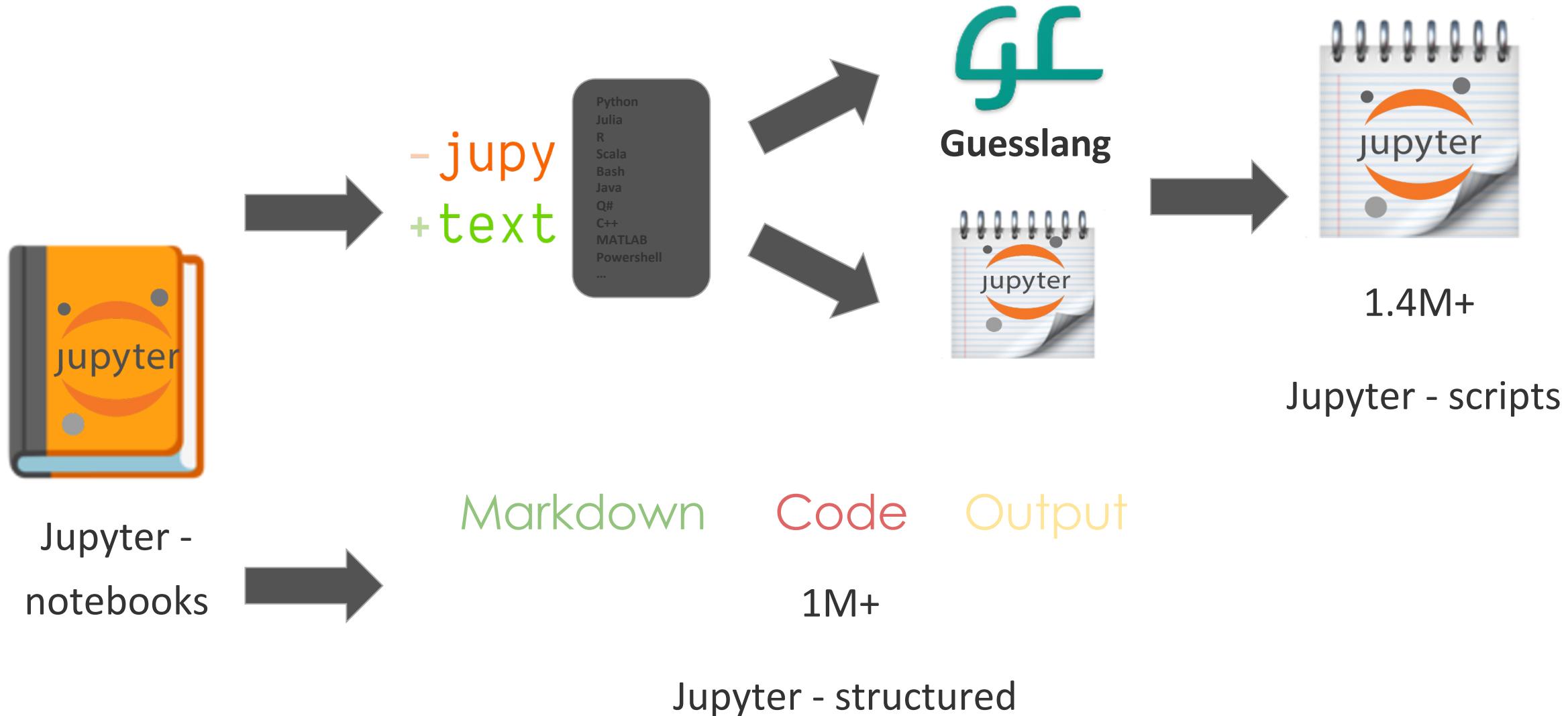
Pseudo-labeling

Take 18 000 files and label them (noisily) with an
ensemble of two NER models: 113 000 entities
(x7 times increase of labeled entities)

Entity type	Train	Test
EMAIL	4721	1742
NAME	3847	1298
IP_ADDRESS	1941	521
USERNAME	1320	346
PASSWORD	390	148
KEY	171	118

Method	Email address			IP address			Key		
	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
Regex	96.20%	97.47%	96.83%	71.29%	87.71%	78.65%	3.62%	49.15%	6.74%
NER	94.01%	98.10%	96.01%	88.95%	94.43%	91.61%	60.37%	53.38%	56.66%
+ pseudo labels	97.73%	98.94%	98.15%	90.10%	93.86%	91.94%	62.38%	80.81%	70.41%

Jupyter Notebooks



Jupyter Notebooks

Model input:

```
<jupyter_text>Let's test our 'is_prime' function:<jupyter_code>
    print(is_prime(3))
print(is_prime(4))
print(is_prime(29))
print(is_prime(33))<jupyter_output>
```

Model output:

```
True
False
True
False
```

Model input:

```
<jupyter_code>numbers = [1, 9, 8, 3, 27]
print([n*2 for n in numbers])<jupyter_output>
```

Model output:

```
[2, 18, 16, 6, 54]
```

GitHub Data

Issues (discussion threads)

```
<issue_start>title + USERID: comment  
<issue_comment>USERID: comment ...  
<issue_closed> <eos>
```

Model input:

```
<issue_start><issue_comment>username_0: I am trying to train a  
text classifier based on BERT using 'transformers' but I  
get the error: 'RuntimeError: CUDA Out of memory'.<  
issue_comment>
```

Model output:

```
username_1: Hi,  
  
This is not an issue with transformers. You should look at your  
GPU usage and see if you can free up some space before  
training starts (or use smaller batch sizes).
```

Commits and Commit Messages

Model input:

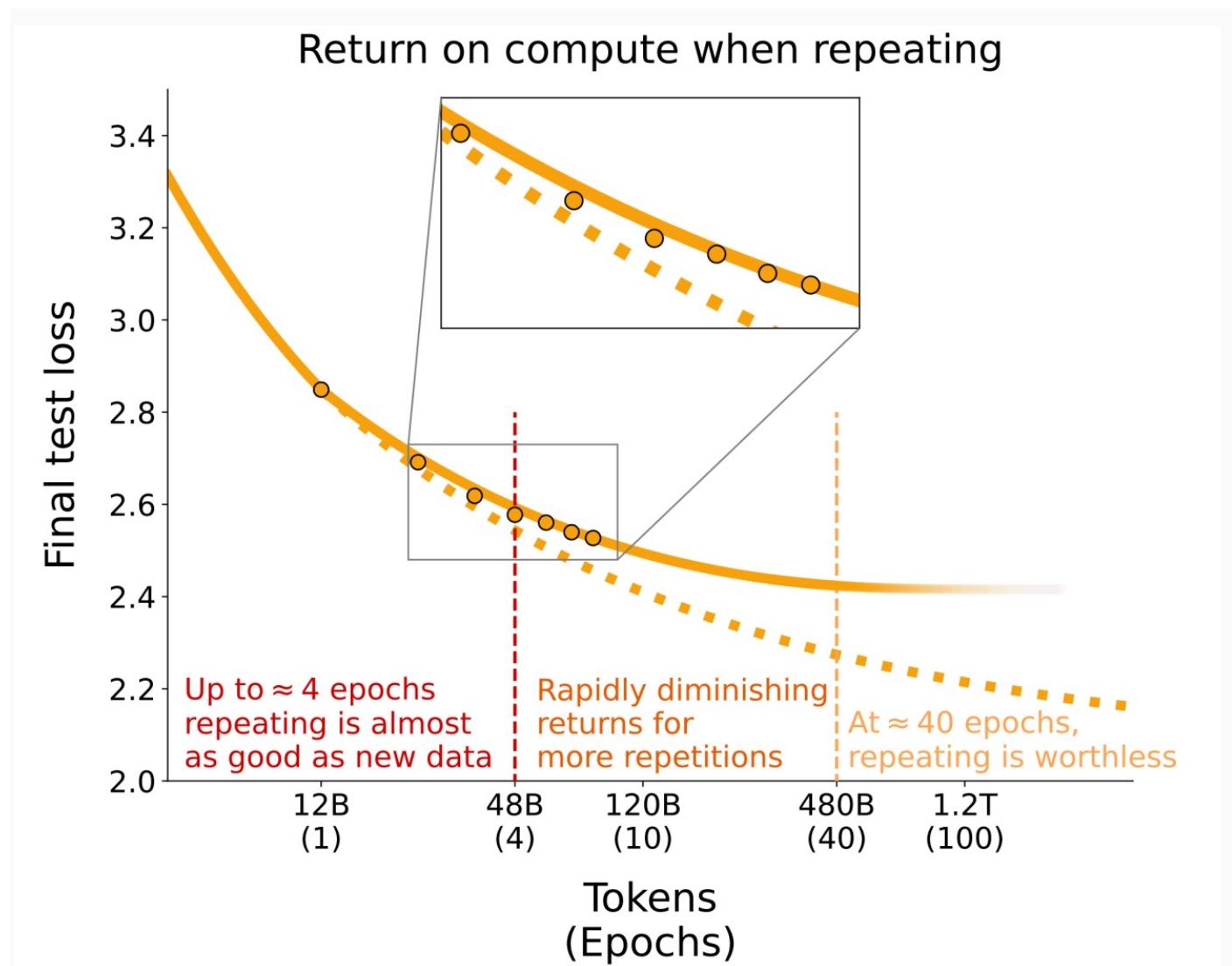
```
<commit_before>def fibonacci(n):<commit_msg>add type hints to  
function<commit_after>def
```

Model output:

```
def fibonacci(n: int) -> list[int]:
```

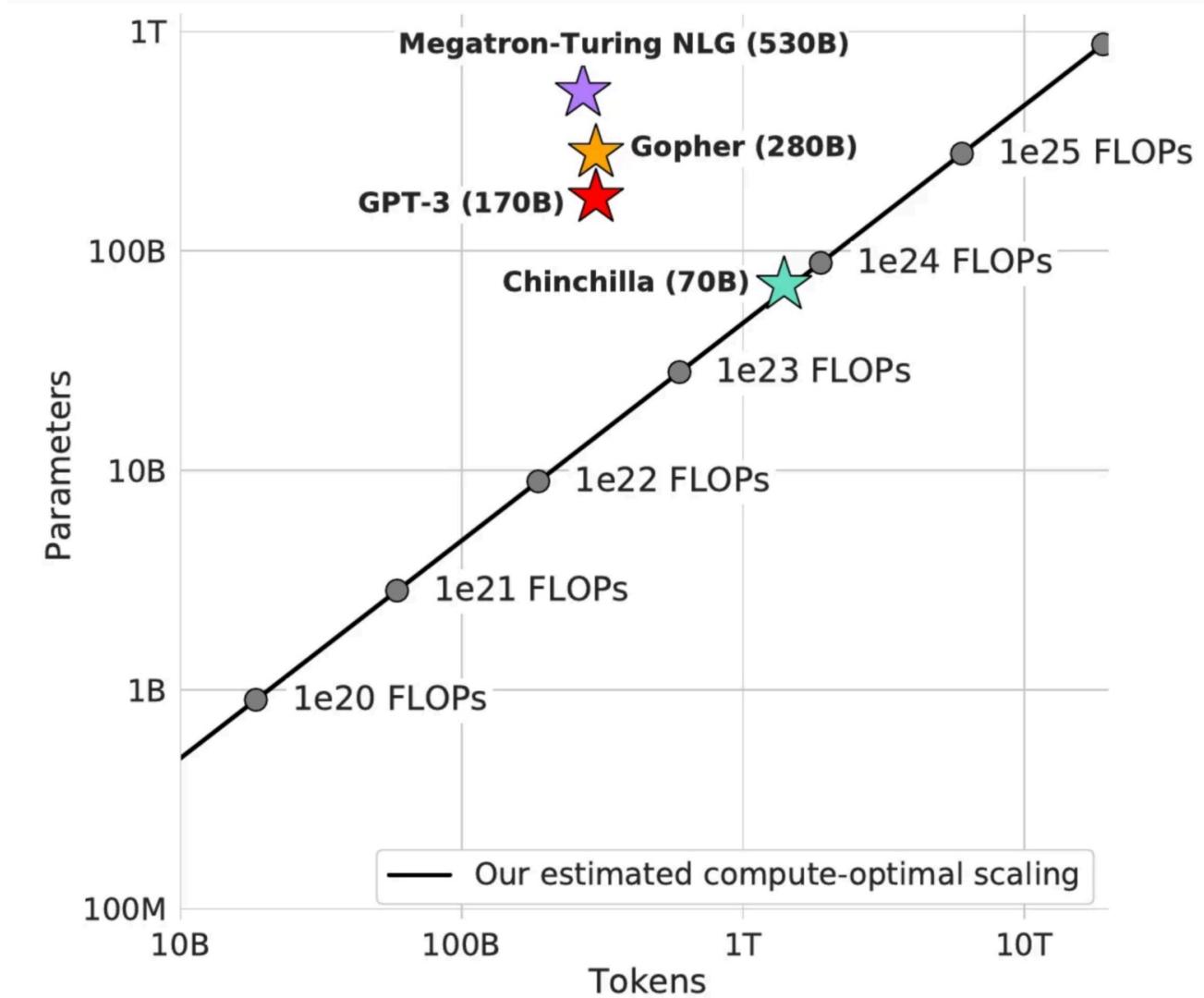
Recap: What About Data Reuse?

- ▶ Mueninghoff et al. were able to train up to 4 epochs on fixed data before seeing significant degradation relative to using new data



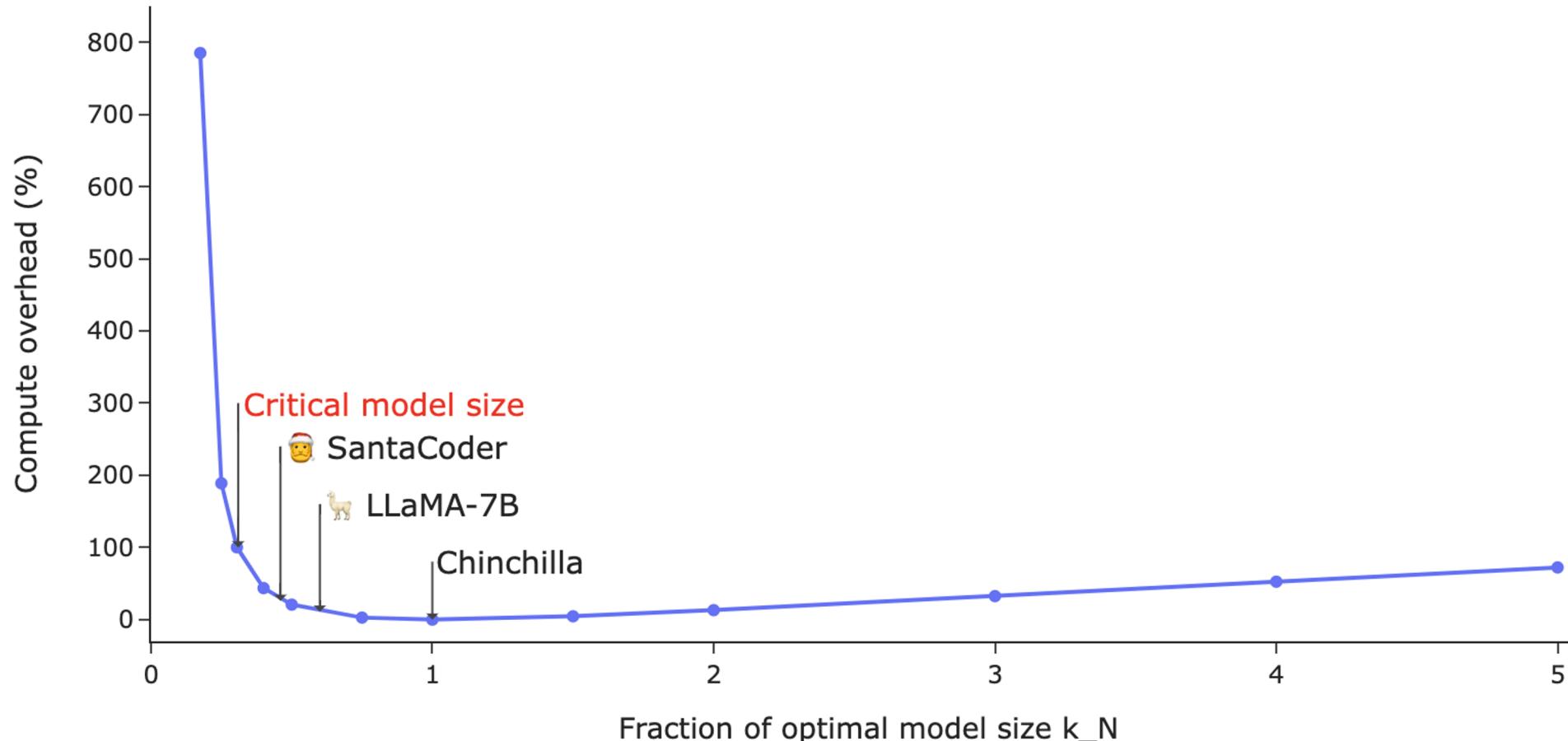
Recap: Scaling Laws

- With a fixed compute budget (number of GPU days), can train a larger model on fewer tokens, or a smaller model on more tokens
- Scaling laws* predict (for a given pre-training dataset, and compute budget), what size Transformer and number of tokens will produce the best loss

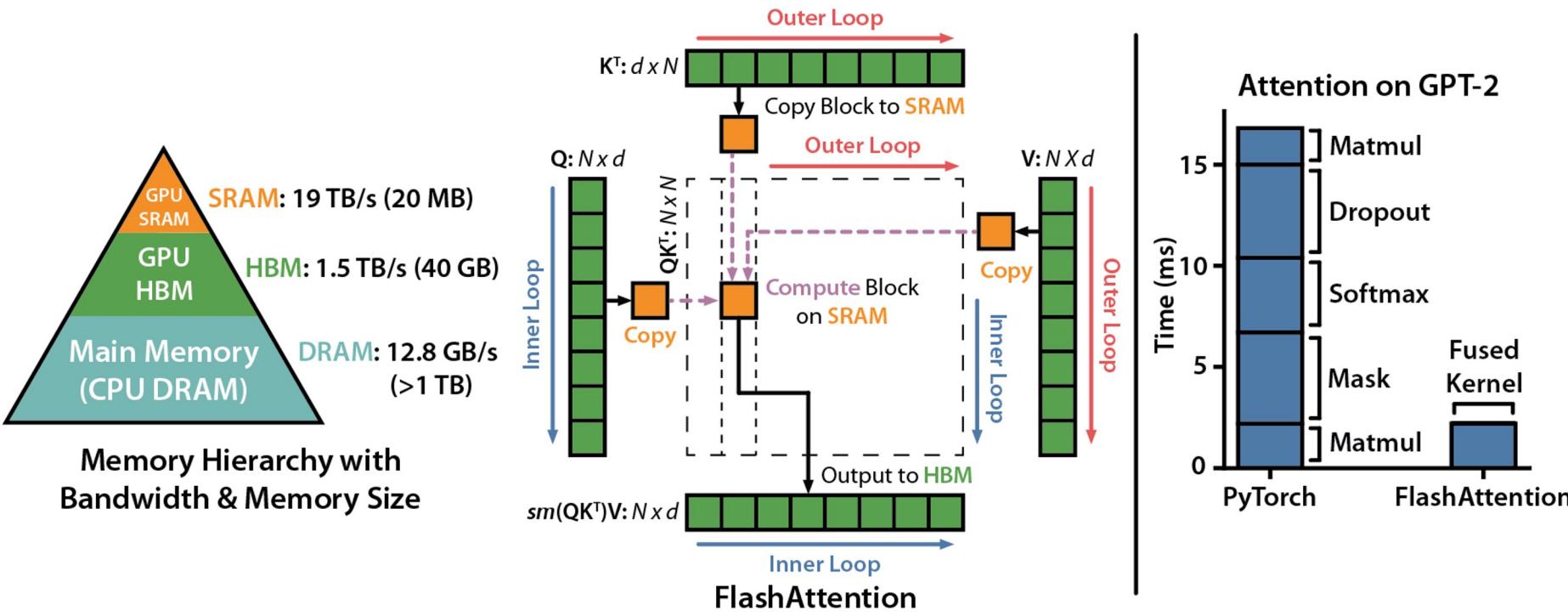


Chinchilla Optimality Ignores Inference

- ▶ Using a smaller model than recommended by the Chinchilla scaling laws comes at a cost to training compute, but it can be small



Flash Attention



→ up to 4x speedup over standard attention
→ scale sequence length up to 8192 tokens.

StarCoder Models

- ▶ StarCoderBase
 - ▷ 15.5B parameters, trained on 1T tokens (~3 epochs)
 - ▶ This is much smaller than Chinchilla optimal, but we were aiming for inference efficiency
 - ▶ Multiple epochs didn't seem to hurt
 - ▷ ~1 month on 512 80GB A100s
 - ▷ Megatron-LM with BF16 and FlashAttention
- ▶ StarCoder
 - ▷ Continued training on 35B tokens of Python (two epochs)

Evaluation Harness: Unified framework for efficient code evaluation

- ▶ Data parallelism for fast text generation with accelerate
- ▶ Unified framework for 7+ code benchmarks: HumanEval, MultiPL-E in 18 programming languages, DS-1000, PaL ...
- ▶ Docker containers for code execution

Official version: <https://github.com/bigcode-project/bigcode-evaluation-harness>
VLLM fork (may be faster): <https://github.com/iNeil77/vllm-code-harness/tree/main>

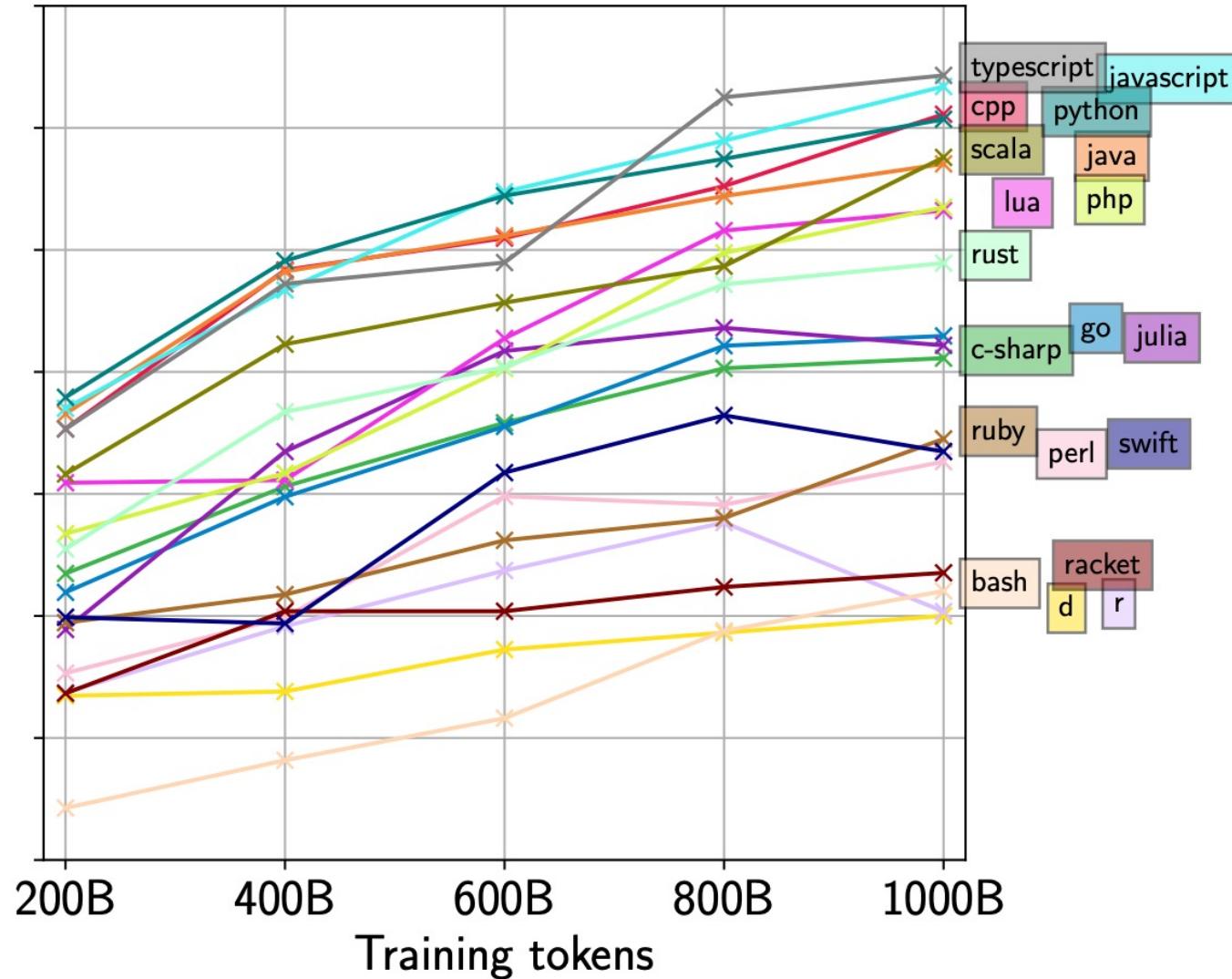
MultiPL-E

- ▶ Translations of the HumanEval benchmark into other programming languages.
- ▶ Together, StarCoderBase and StarCoder outperform OpenAI's code-cushman-001 on HumanEval in 12 languages.
- ▶ Surprisingly, StarCoder outperforms StarCoderBase on 9 languages in addition to Python.

Language	code-cushman-001	StarCoder	StarCoderBase
cpp	30.59	31.55	30.56
c-sharp	22.06	21.01	20.56
d	6.73	13.57	10.01
go	19.68	17.61	21.47
java	31.90	30.22	28.53
julia	1.54	23.02	21.09
javascript	31.27	30.79	31.70
lua	26.24	23.89	26.61
php	28.94	26.08	26.75
perl	19.29	17.34	16.32
python	30.71	33.57	30.35
r	10.99	15.50	10.18
ruby	28.63	1.24	17.25
racket	7.05	0.07	11.77
rust	25.22	21.84	24.46
scala	27.62	27.61	28.79
bash	11.74	10.46	11.02
swift	22.12	22.74	16.74
typescript	31.26	32.29	32.15

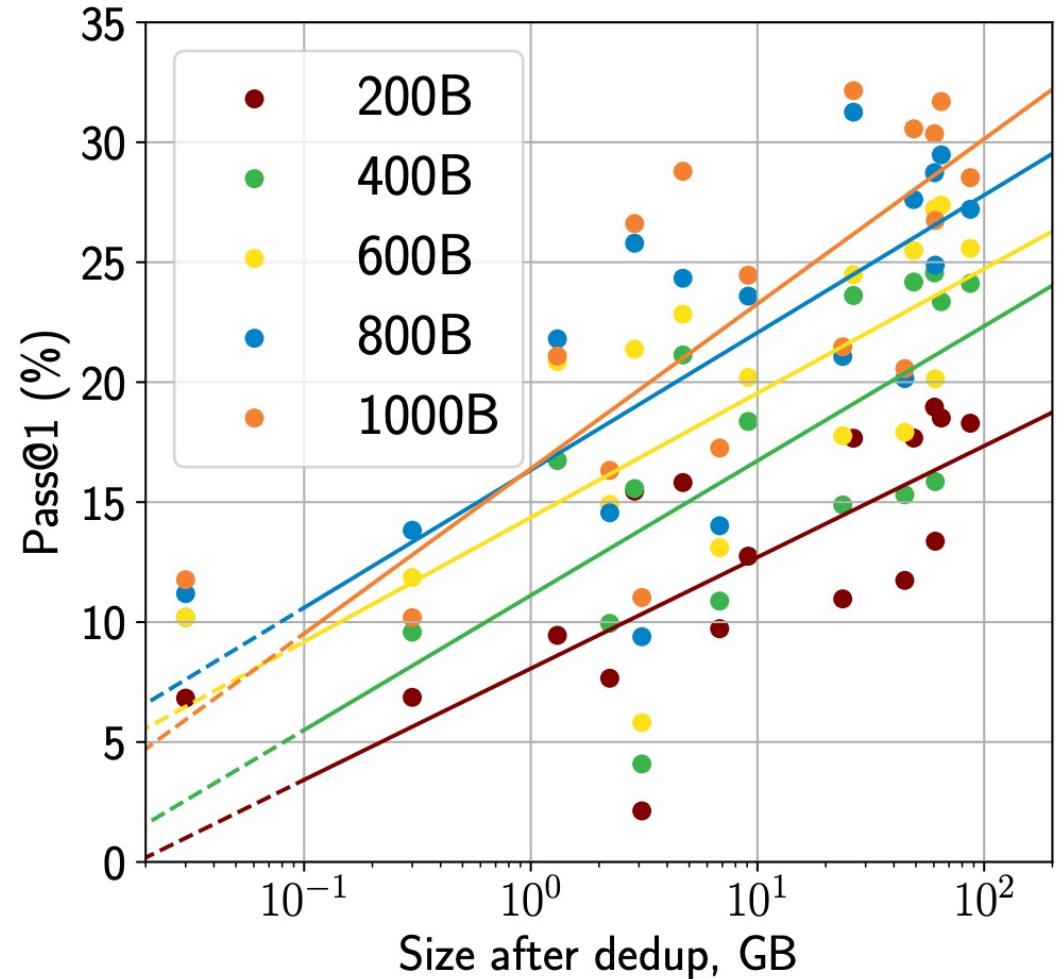
MultiPL-E translated HumanEval results

StarCoderBase: Performance Over Training



StarCoderBase: Performance By Data

- ▶ How correlated is code completion performance for a language with the amount of data available for a language?
- ▶ Train model for 200B tokens (on all languages). Evaluate on all languages, getting a dot for each language. Observe a strong correlation.
- ▶ Continue training, evaluate again at 400B tokens. The correlation remains strong, and line shifts upward.



Language Competition?

- ▶ InCoder saw slight competition between languages at the 1.3B param scale:

#	Size (B)	Obj.	Training Data	Data Size	Train Tokens	Train Compute	HumanEval Pass@1	MBPP Pass@1
1)	6.7	CM	multi lang + SO	204 GB	52 B	3.0 Z	15	19.4
2)	1.3	CM	multi lang + SO	204 GB	52 B	0.6 Z	8	10.9
3)	1.3	LM	multi lang + SO	204 GB	52 B	0.6 Z	6	8.9
4)	1.3	LM	Python + SO	104 GB	25 B	0.3 Z	9	9.8
5)	1.3	LM	Python	49 GB	11 B	0.1 Z	5	6.1
6)	2.3	LM	multi lang + SO	204 GB	52 B	1.1 Z	9	12.7

- ▶ But is there competition among languages in these large models?

Scaling Laws for Mixed-Modal Models

- In multi-modal settings, modalities compete when models are small; can synergize when models are large.

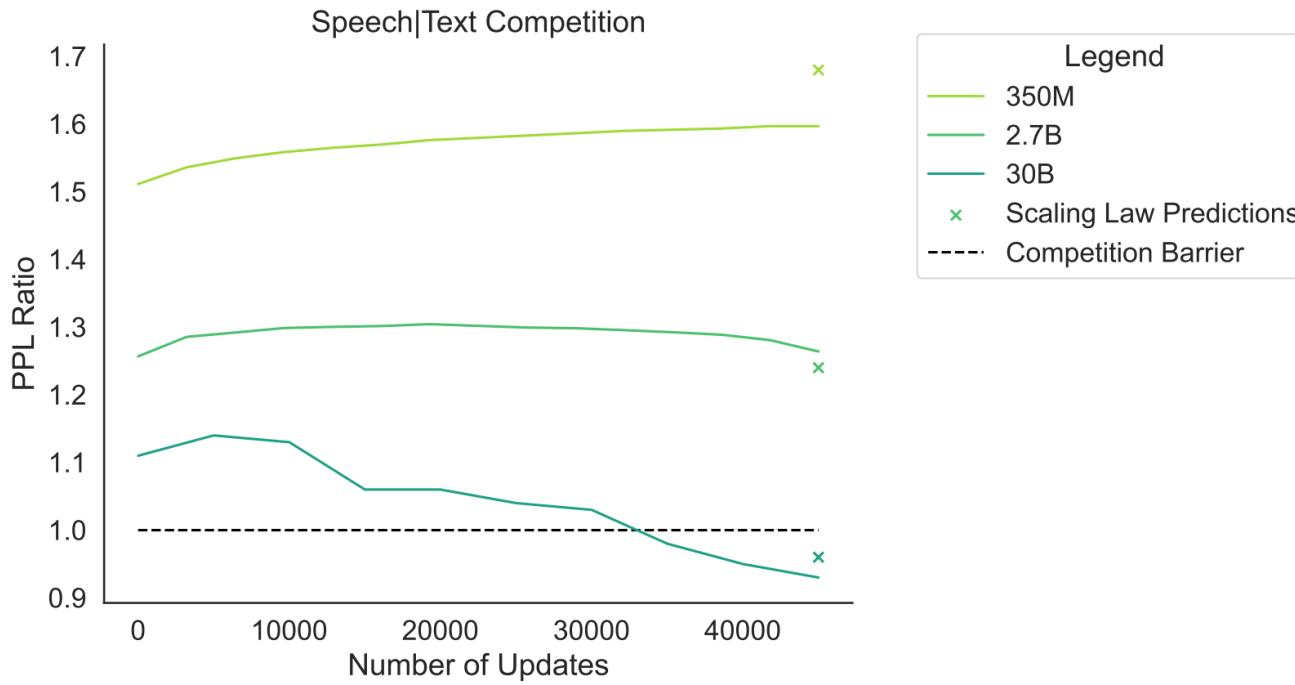


Figure 5: We plot $\frac{0.5 * (\mathcal{L}(N, \text{Text}) + \mathcal{L}(N, \text{Speech}))}{\mathcal{L}(N, [\text{Speech}, \text{Text}])}$ throughout the training process. If this ratio is below 1, we have broken through the competition barrier. Additionally, we add the predictions for the final ratio as predicted from our scaling laws.

DS-1000: Practical data tasks requiring API use

Here is a sample dataframe:

Problem

```
df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
```

I'd like to add inverses of each existing column to the dataframe and name them based on existing column names with a prefix, e.g. inv_A is an inverse of column A and so on.

The resulting dataframe should look like so:

```
result = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6], "inv_A": [1/1, 1/2, 1/3], "inv_B": [1/4, 1/5, 1/6]})
```

Obviously there are redundant methods like doing this in a loop, **but there should exist much more pythonic ways of doing it ...** [omitted for brevity]

Code Context

A:

```
<code>
import pandas as pd
df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
</code>
BEGIN SOLUTION
<code>
[insert]
</code>
END SOLUTION
<code>
print(result)
</code>
```

Reference Solution

```
result = df.join(df.apply(lambda x: 1/x).add_prefix("inv_"))
```

Format	Model	Matplotlib	NumPy	Pandas	PyTorch	SciPy	Scikit-Learn	TensorFlow	Overall
	Number of problems:	155	220	291	68	106	115	45	1,000
Completion	InCoder-6B	28.3	4.4	3.1	4.4	2.8	2.8	3.8	7.4
Completion	CodeGen-16B-Mono	31.7	10.9	3.4	7.0	9.0	10.8	15.2	11.7
Completion	code-cushman-001	40.7	21.8	7.9	12.4	11.3	18.0	12.2	18.1
Completion	StarCoderBase	47.0	27.1	10.1	19.5	21.7	27.0	20.5	23.8
Completion	StarCoder	51.7	29.7	11.4	21.4	20.2	29.5	24.5	26.0

Evaluating Infilling

Model	Java	JavaScript	Python
InCoder-6B	0.49	0.51	0.31
SantaCoder	0.62	0.60	0.44
StarCoder	0.73	0.74	0.62

Single-line code completion for three languages
(SantaCoder/InCoder benchmarks)

Packages type check		
	✓	Total
InCoder	30	128
StarCoderBase	49	128

TypeScript type inference (TypeWeaver benchmarks)

Model	BLEU
InCoder-6B	18.27
SantaCoder	19.74
StarCoderBase	21.38
StarCoder	21.99

Python docstring generation
(CodeXGLUE / InCoder benchmark)

Model	Non-None F1	All F1
InCoder-6B	59.1	46.8
SantaCoder	66.9	78.5
StarCoderBase	77.4	86.6
StarCoder	77.1	86.4

Python return-type prediction
(InCoder/TypeWriter benchmarks)

Testing 8K Window: Perplexity with Long Contexts

Window Size	Language									
	cpp	c-sharp	c	go	java	javascript	php	r	ruby	rust
2K tokens	2.01	1.90	1.71	1.35	1.65	1.98	1.73	1.72	2.16	1.84
8K tokens	1.79	1.66	1.61	1.21	1.54	1.68	1.43	1.48	2.02	1.65

- ▶ Derived test data from GPL repositories on GitHub. GPL was excluded from training data.
- ▶ Demonstrates StarCoder can benefit from information within long files or repositories.
- ▶ Longer contexts provides noticeable decreases in perplexity.

Non-Trivial Natural Language Abilities

- ▶ Surprisingly reasonable performance on some natural language reasoning tasks
- ▶ CodeGen < StarCoderBase < LLaMA

Problem: Beth bakes 4, 2 dozen batches of cookies in a week. If these cookies are shared amongst 16 people equally, how many cookies does each person consume?

Solution: Beth bakes 4 2 dozen batches of cookies for a total of $4 \times 2 = \text{<<} 4*2=8 \text{>>}$ 8 dozen cookies

There are 12 cookies in a dozen and she makes 8 dozen cookies for a total of $12 \times 8 = \text{<<} 12*8=96 \text{>>}$ 96 cookies

She splits the 96 cookies equally amongst 16 people so they each eat $96 / 16 = \text{<<} 96/16=6 \text{>>}$ 6 cookies

Final Answer: 6

Model	Size	GSM8K CoT	+maj1@100	GSM8K PAL	+maj1@40
StarCoderBase	15.5B	8.4	—	21.5	31.2
CodeGen-Multi	16B	3.18	—	8.6	15.2
CodeGen-Mono	16B	2.6	—	13.1	22.4
LLaMA	7B	11.0	18.1	10.5	16.8
	13B	17.8	29.3	16.9	28.5
	33B	35.6	53.1	38.7	50.3
	65B	50.9	69.7	—	—

Reasoning Tasks in HELM

Model	Size	Open Access	Synth. Reason. (AS)	Synth. Reason. (NL)	bAbI	Dyck	GSM8K	MATH	MATH (CoT)	LSAT	Legal Support
code-davinci-002	175B		54.0	68.4	68.6	80.5	56.8	41.0	43.3	—	—
text-davinci-003	175B		50.2	73.4	65.3	75.1	50.6	39.0	44.9	23.3	62.2
Luminous Supreme	70B		31.2	—	50.4	72.9	11.2	14.9	5.7	21.2	53.0
StarCoderBase	15.5B	✓	44.0	21.0	50.4	85.4	8.4	15.1	7.0	19.0	53.2
Cohere Command Beta	52.4B		24.3	24.5	47.3	42.1	13.8	13.3	7.5	22.9	60.6
J1-Jumbo v1	178B		26.3	17.4	54.3	44.5	5.4	8.9	3.3	23.2	48.4
J1-Grande v2 beta	17B		28.6	13.9	47.0	61.7	9.6	12.7	6.8	19.1	56.2
code-cushman-001	12B		34.1	16.4	48.1	45.1	4.9	9.9	7.2	—	—
OPT	175B	✓	22.5	24.8	50.7	49.4	4.0	6.5	2.6	22.0	53.2
GPT-NeoX	20B	✓	20.4	16.7	46.8	74.7	5.3	14.1	7.1	19.1	51.5
BLOOM	176B	✓	30.4	19.7	44.7	54.5	9.5	4.3	5.5	20.9	54.3
GLM	130B	✓	25.2	25.4	44.3	54.9	6.1	0	5.9	19.3	45.1
UL2	20B	✓	20.5	21.7	50.1	14.0	2.4	0	0	20.7	50.6
OPT	66B	✓	19.3	21.3	40.8	47.1	1.8	4.8	2.9	17.5	52.7
YaLM	100B	✓	5.6	6.1	34.6	63.3	0	0	0	2.3	48.4
T5	11B	✓	19.6	10.1	41.2	34.7	2.3	0	0	15.9	55.8

Membership Checking and Indexing

Submit

Multiple levels of data attribution, documentation tools!

Am I in The Stack?

Stack: Data Portrait
(stack.dataportraits.org)

StarCoder: Dataset Search 

Lightweight checks for other demos/plugin!

We can analyze the data; end users can interact

StarCoder might be one of the *most* documented LLMs + dataset combos

Source: [chilin0525/model-layer-profiling/test.py](#) | Language: python | License: MIT

```
from transformers import AutoTokenizer, AutoModel
import torch
import torch.cuda.profiler as profiler
import nvidia_dlprof_pytorch_nvtx
```

```
# call the magic code generation model
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
inputs = tokenizer("Hello world!", return_tensors="pt")
```

Matching Text

Found spans are in grey. The longest span is in blue. Hovering over a character highlights the longest span that includes that character (there may be overlapping shorter spans). Clicking shows the component substrings below.

```
# call the magic code generation model
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
inputs = tokenizer("Hello world!", return_tensors="pt")
```

Substring Hashes

DeepSeek Coder

DeepSeek Coder

- ▶ 1.3B, 6.7B, and 33B parameter models
- ▶ Trained from scratch on 2 Trillion tokens of code from 87 languages
- ▶ FIM loss, and 16K context length

DeepSeek Coder: Data

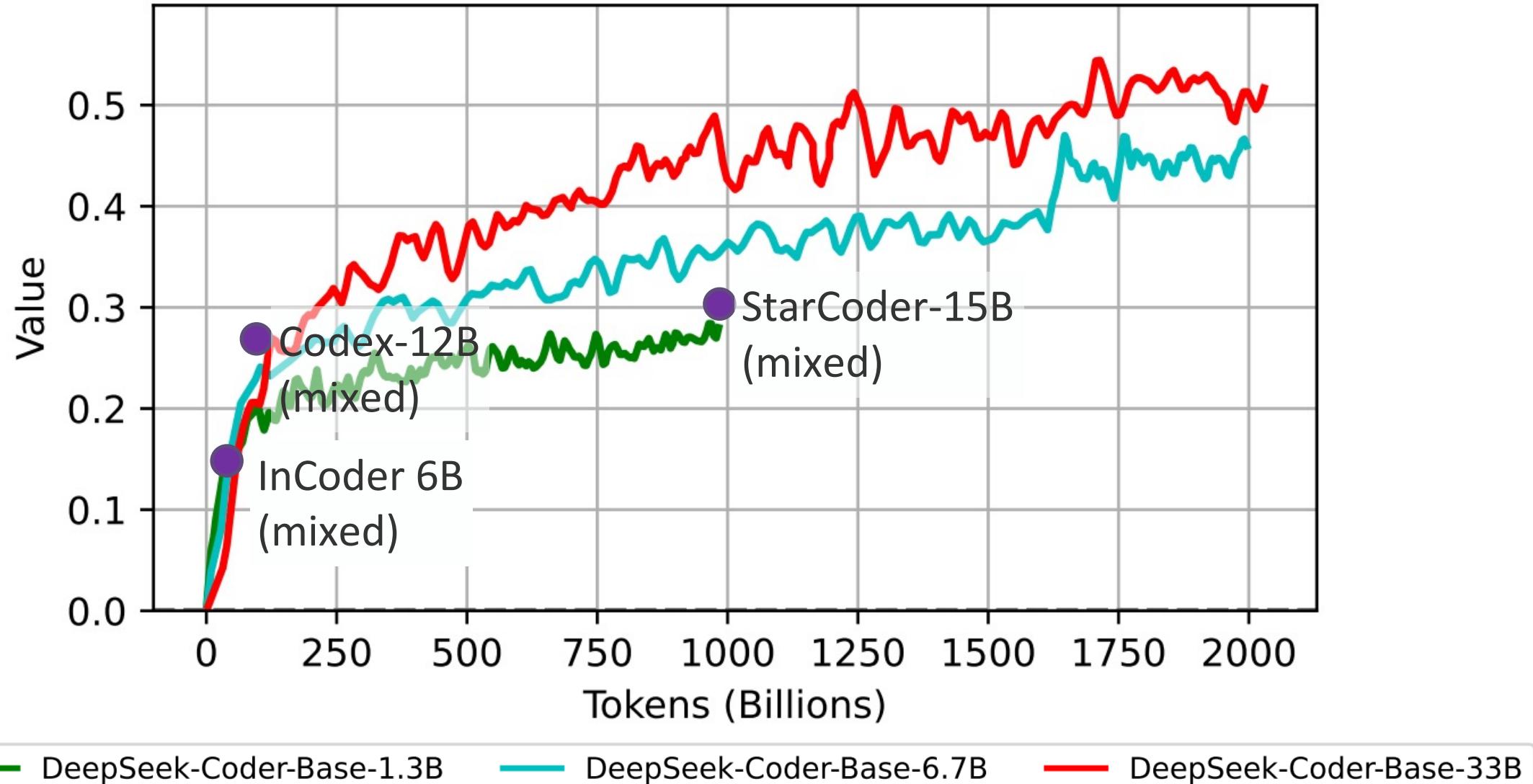
- ▶ 87% code, 10% code-related English NL, 3% code-unrelated Chinese NL
- ▶ Pre-training: 800GB, 2 Trillion tokens.
 - ▷ StarCoder filtering and less aggressive deduplication (repo-level)
 - ▷ Also remove code with syntax errors, poor readability, low modularity
 - ▷ *May have up-sampled Python relative to the natural distribution?*
 - ▷ *Probably not license-filtered?*

DeepSeek Coder: Repo-Level Context

- ▶ Parse file dependencies and arrange repo files in the context window using a topological ordering.
- ▶ Theoretically can handle 64K tokens, but “empirical observations suggest that the model delivers its most reliable outputs within a 16K token range”

DeepSeek Coder: Data, Data, Data

HumanEval-Pass@1



DeepSeek Coder: Results

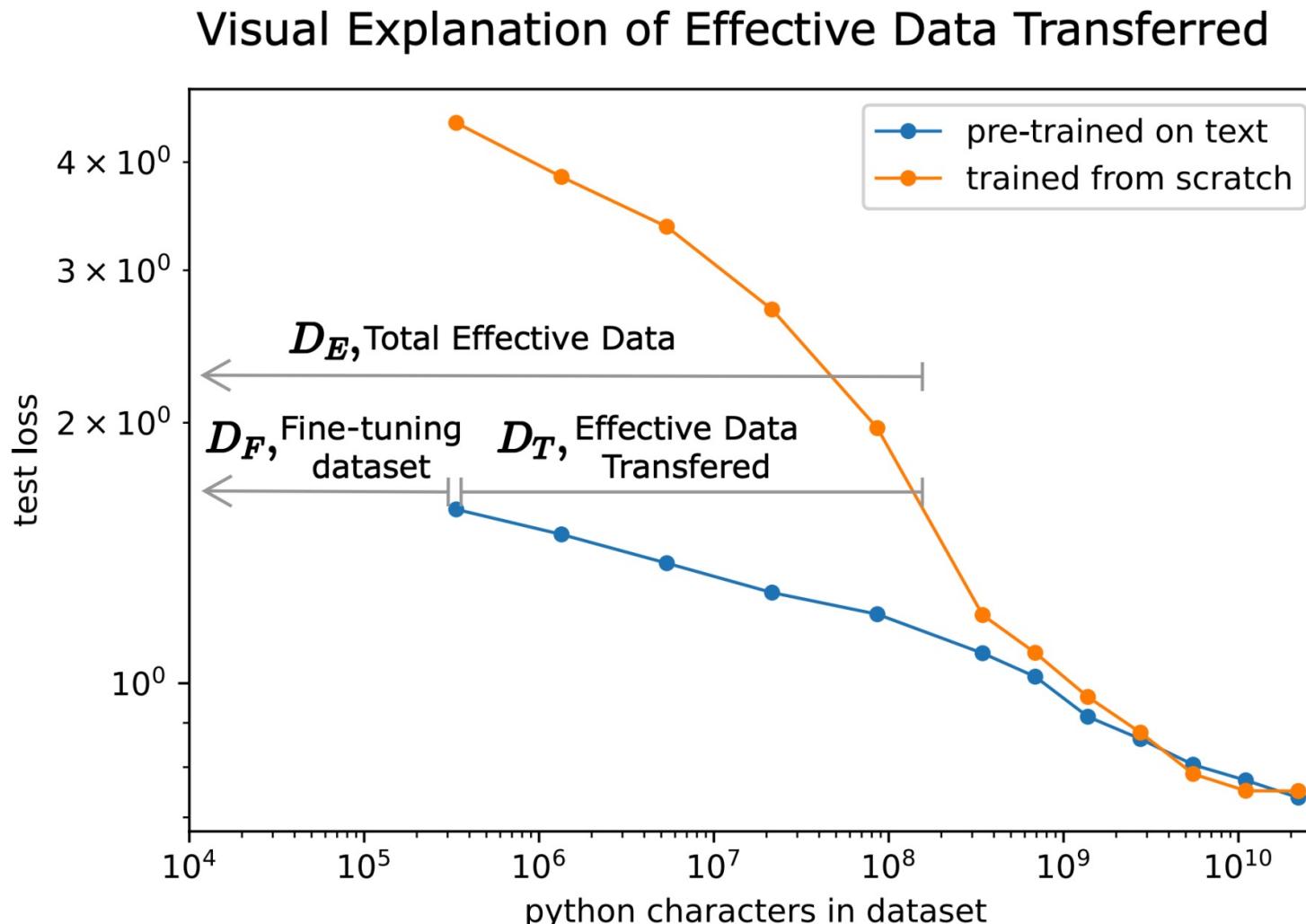
► MultiPL-E HumanEval and MBPP

Model	Size	Python	C++	Java	PHP	TS	C#	Bash	JS	Avg	MBPP
Multilingual Base Models											
code-cushman-001	12B	33.5%	31.9%	30.6%	28.9%	31.3%	22.1%	11.7%	-	-	-
CodeGeeX2	6B	36.0%	29.2%	25.9%	23.6%	20.8%	29.7%	6.3%	24.8%	24.5%	36.2%
StarCoderBase	16B	31.7%	31.1%	28.5%	25.4%	34.0%	34.8%	8.9%	29.8%	28.0%	42.8%
CodeLlama	7B	31.7%	29.8%	34.2%	23.6%	36.5%	36.7%	12.0%	29.2%	29.2%	38.6%
CodeLlama	13B	36.0%	37.9%	38.0%	34.2%	45.2%	43.0%	16.5%	32.3%	35.4%	48.4%
CodeLlama	34B	48.2%	44.7%	44.9%	41.0%	42.1%	48.7%	15.8%	42.2%	41.0%	55.2%
DeepSeek-Coder-Base	1.3B	34.8%	31.1%	32.3%	24.2%	28.9%	36.7%	10.1%	28.6%	28.3%	46.2%
DeepSeek-Coder-Base	6.7B	49.4%	50.3%	43.0%	38.5%	49.7%	50.0%	28.5%	48.4%	44.7%	60.6%
DeepSeek-Coder-Base	33B	56.1%	58.4%	51.9%	44.1%	52.8%	51.3%	32.3%	55.3%	50.3%	66.0%

► DS-1000

Model	Size	Matplotlib	Numpy	Pandas	Pytorch	Scipy	Scikit-Learn	Tensorflow	Avg
CodeGeeX2	6B	38.7%	26.8%	14.4%	11.8%	19.8%	27.0%	17.8%	22.9%
StarCoder-Base	16B	43.2%	29.1%	11.0%	20.6%	23.6%	32.2%	15.6%	24.6%
CodeLlama-Base	7B	41.9%	24.6%	14.8%	16.2%	18.9%	17.4%	17.8%	22.1%
CodeLlama-Base	13B	46.5%	28.6%	18.2%	19.1%	18.9%	27.8%	33.3%	26.8%
CodeLlama-Base	34B	50.3%	42.7%	23.0%	25.0%	28.3%	33.9%	40.0%	34.3%
DeepSeek-Coder-Base	1.3B	32.3%	21.4%	9.3%	8.8%	8.5%	16.5%	8.9%	16.2%
DeepSeek-Coder-Base	6.7B	48.4%	35.5%	20.6%	19.1%	22.6%	38.3%	24.4%	30.5%
DeepSeek-Coder-Base	33B	56.1%	49.6%	25.8%	36.8%	36.8%	40.0%	46.7%	40.2%

Recap: Scaling laws of transfer



DeepSeek Coder: Effects of Transfer

- ▶ V-1.5: start with a model trained on lots of NL text (DeepSeek-LLM), then fine-tune on 2T tokens (70% code, 30% NL)
- ▶ Instruct: fine-tune on 2B tokens of instructions (details unclear)

Models	Size	Programming		Math Reasoning		Natural Language				
		HumanEval	MBPP	GSM8K	MATH	MMLU	BBH	HellaSwag	WinoG	ARC-C
DeepSeek-Coder-Base	6.7B	44.7%	60.6%	43.2%	19.2%	36.6%	44.3%	53.8%	57.1%	32.5%
DeepSeek-Coder-Base-v1.5	6.9B	43.2%	60.4%	62.4%	24.7%	49.1%	55.2%	69.9%	63.8%	47.2%
DeepSeek-Coder-Instruct	6.7B	66.1%	65.4%	62.8%	28.6%	37.2%	46.9%	55.0%	57.6%	37.4%
DeepSeek-Coder-Instruct-v1.5	6.9B	64.1%	64.6%	72.6%	34.1%	49.5%	53.3%	72.2%	63.4%	48.1%

DeepSeek Coder: Example from Instruct-33B

Q1: Write a game snake using pygame



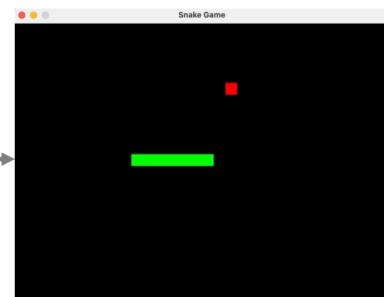
Sure, here is a simple implementation of the classic Snake game using Pygame. This code will create a window with a snake that can be controlled using the arrow keys. The snake will grow in length each time it eats the food. The game will end if the snake hits the edge of the window or its own tail.

```
'''python
import pygame
import random

...
running = True
While running:
    ...
    # Draw everything
    screen.fill(BLACK)
    snake.draw()
    food.draw()
    pygame.display.flip()
    # Cap the frame rate
    clock.tick(SPEED)
# Quit Pygame
pygame.quit()
'''
```

This code sets up a basic game loop where the snake moves in response to key presses, and the game ends if the snake hits the edge of the window or its own tail. The snake grows in length each time it eats the food. The game continues until the player closes the window.

Run Code



Q2: Add a scoring system in the top left corner

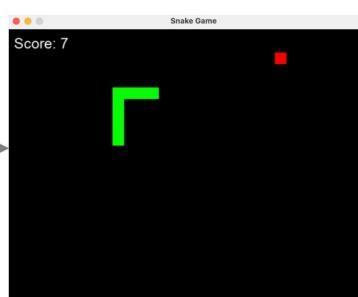


To add a scoring system to the top left corner of the game window, you can create a function to display the score and increment it each time the snake eats the food. Here's how you can modify the code to include a scoring system:

```
'''python
...
running = True
While running:
    ...
    # Draw everything
    screen.fill(BLACK)
    snake.draw()
    food.draw()
    display_score(score) # Display the score
    pygame.display.flip()
    # Cap the frame rate
    clock.tick(SPEED)
# Quit Pygame
pygame.quit()
'''
```

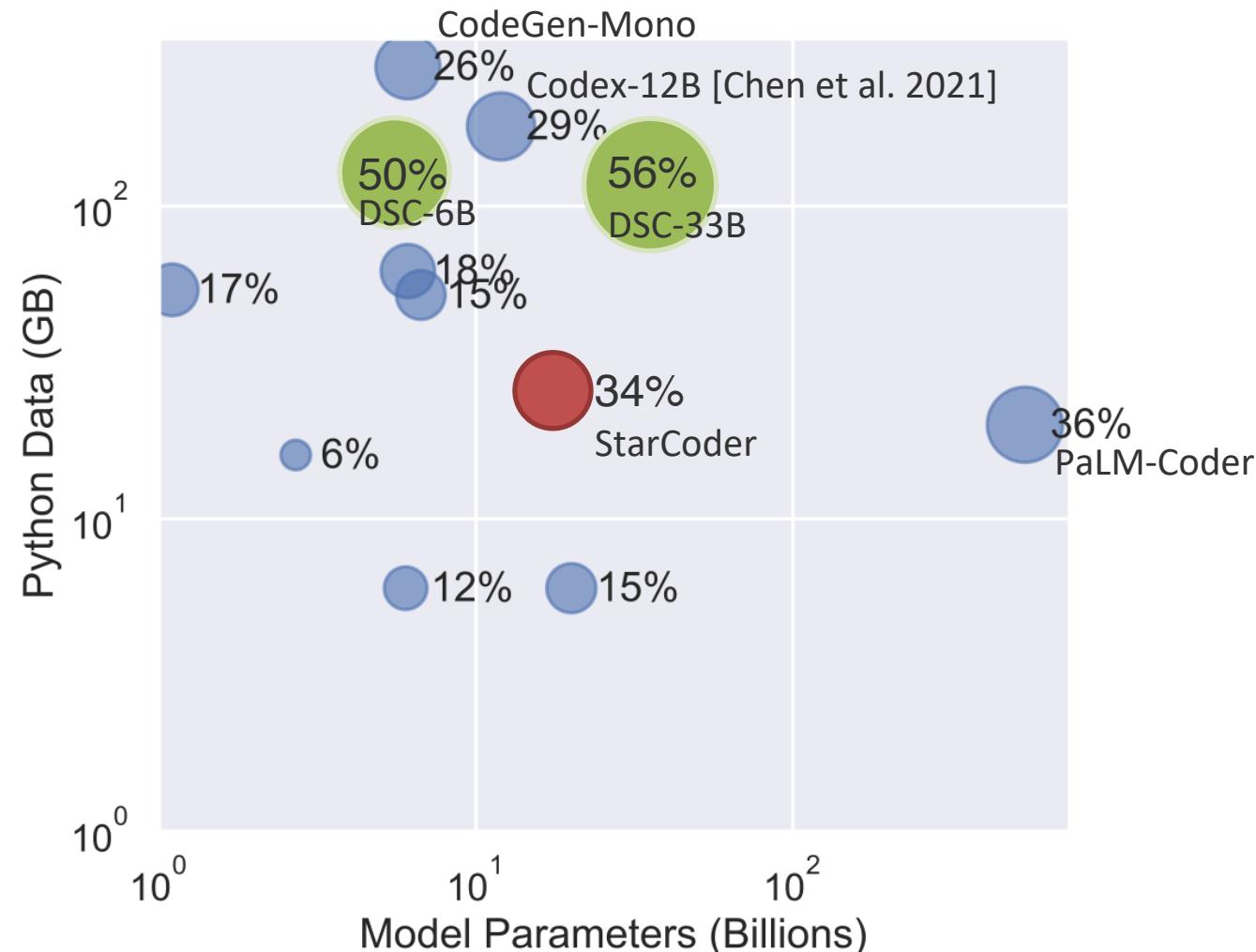
In this updated code, we've added a `score` variable to keep track of the player's score, which is incremented each time the snake eats the food. The `display_score` function is used to render the score text and display it in the top left corner of the window. The `display_score` function is called after the snake has moved and the screen has been filled with the black color, but before the `pygame.display.flip()` call, so that the score is drawn on top of everything else.

Run Code



What Makes a Model Good?

Data size, model size, data filtering, optimization quality, and competition/synergies among training data...



Questions?