

# starcoder研究

[TOC]

## StarCoder

### 省流

starcoder开源的很好

从预训练的数据集、预训练得到的模型

如何微调

vscode插件源码-都给你

长度为8192的token

### 简介

从它的简介开始

☞ [README.md · Bigcode/Starcoder at Main \(huggingface.co\)](#)

StarCoder 模型是 15.5B 参数模型，使用 The Stack (v1.2) 中的 80 多种编程语言进行训练，不包括选择退出请求。该模型使用 Multi Query Attention，一个包含 8192 个标记的上下文窗口，并使用 Fill-in-the-Middle（中间填空）目标对 1 万亿个标记进行训练。

不包括选择退出请求，指的是根据数据提供者或利益相关者的明确要求，出于隐私或知识产权等原因，将特定数据从训练过程中排除的做法。

☞ [bigcode/the-stack · Datasets at Hugging Face](#)

The Stack (v1.2)是一个数据集，6TB，huggingface可下载



# The Stack

6 TB of permissive code data



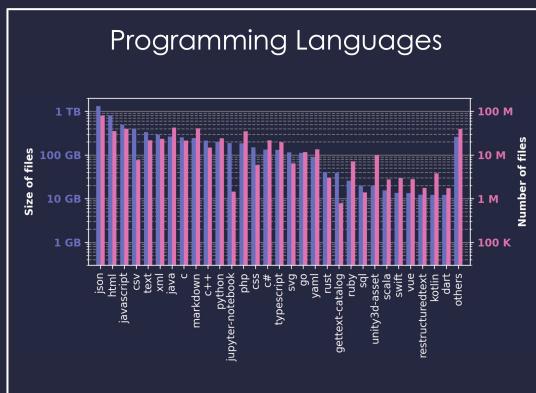
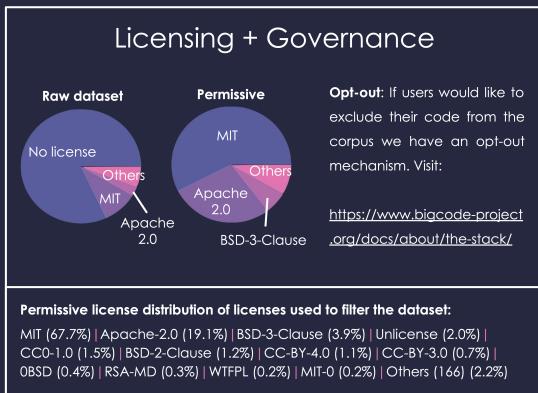
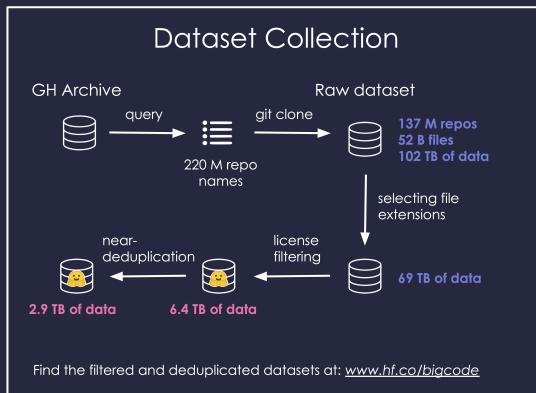
@BigCodeProject



<https://www.bigcode-project.org/>



contact@bigcode-project.org



**Evaluation**

We trained several **GPT-2 models (350M parameters)** on different parts of the dataset both with and without near-deduplication. The models trained on the Python subset of The Stack performed on par with CodeX and CodeGen of similar size when using near-deduplication.

Dataset	Filtering	pass@1	pass@10	pass@100
Codex (300M)	unknown	13.17	20.17	36.27
CodeGen (350M)	unknown	12.76	23.11	35.19
Python all-license	None	13.11	21.77	36.67
	Near-dedup	<b>17.34</b>	<b>27.64</b>	<b>45.52</b>
Python permissive-license	None	10.99	15.94	27.21
	Near-dedup	<b>12.89</b>	<b>22.26</b>	<b>36.01</b>

\*results obtained with The Stack v1.0

**Usage**

```

pip install datasets
from datasets import load_dataset
# full dataset (3TB of data)
ds = load_dataset("bigcode/the-stack", split="train")
# near-duplicated dataset (1.5TB of data)
ds = load_dataset("bigcode/the-stack-dedup", split="train")
# specific language (e.g. Dockerfiles)
ds = load_dataset("bigcode/the-stack", data_dir="data/dockerfile", split="train")
# dataset streaming (will only download the data as needed)
ds = load_dataset("bigcode/the-stack", streaming=True, split="train")
for sample in iter(ds):
    print(sample['content'])
    
```

For more info visit about the Datasets library visit: [hf.co/docs/datasets](https://hf.co/docs/datasets)

**Dataset Trivia**

- There are 1,467,018 files containing "hello world" in the dataset.
- The message "Your mom ate the database" (in German) appears in a file.
- It would take a single person **75,000 years** to type all the code from scratch.
- The Stack is **320x larger** than the English Wikipedia.

Printed double sided on A4 paper the stacked pile would almost reach **25x** the **Mount everest** and you could cover 18x the distance between earth and moon if the paper is aligned side-by-side.

18x

## 技术架构

架构: GPT-2模型，具有多查询注意力和填充中间目标

预训练步骤: 250,000步

预训练令牌数: 1万亿

精度: bfloat16

编排 (Orchestration) : Megatron-LM

神经网络 (Neural networks) : PyTorch

下面介绍其中的点

## 架构: GPT-2模型，具有多查询注意力和填充中间目标

GPT和BERT的区别

GPT (Generative Pre-trained Transformer) 和BERT (Bidirectional Encoder Representations from Transformers) 是两个非常流行的自然语言处理模型，它们在一些关键方面存在区别。

### 1. 架构:

- GPT: GPT采用了Transformer架构，它是一种基于自注意力机制的深度神经网络结构。GPT模型由多个相同的自注意力层和前馈神经网络层组成，其中自注意力层用于捕捉输入文本的上下文信息。
- BERT: BERT也采用了Transformer架构，但它引入了双向 (bidirectional) 的训练方式。BERT模型包含两个阶段：预训练和微调。在预训练阶段，BERT通过遮蔽语言模型和下一句预测任务，从大量未标记的文本中学习通用的语言表示。在微调阶段，BERT根据特定任务的标记数据进行微调，以适应具体的任务。

### 2. 训练目标:

- GPT: GPT是一个生成模型，它在训练过程中通过预测下一个词来学习文本序列的概率分布。这使得GPT在生成文本方面非常擅长，如对话生成、文章摘要等。
- BERT: BERT是一个判别模型，它的目标是学习出一个句子的固定表示。BERT在预训练过程中使用了遮蔽语言模型和下一句预测任务，但不涉及生成文本。

### 3. 上下文处理:

- GPT: GPT模型在生成文本时，只能依赖前面的上文信息，不能直接利用后面的上下文。因此，它适合于生成式任务，例如文本生成和语言建模。
- BERT: BERT模型采用双向训练，可以同时利用前后的上下文信息。这使得BERT在理解句子和文本分类等任务中表现出色。

总体而言，GPT和BERT的区别在于架构、训练目标和上下文处理方式。GPT适用于生成式任务，而BERT适用于理解和分类任务。选择适合特定任务的模型取决于任务的要求和数据的性质。

## 预训练令牌数：1万亿

在自然语言处理任务中，令牌是对文本进行分割和编码的基本单位，可以是一个单词、一个字母或者一个符号。

训练过程中，模型接收一定数量的文本输入，并根据这些文本进行学习和参数更新，以提高其在各种语言任务上的性能。

使用更多的令牌进行训练可以带来一些优势，例如模型能够学习更长的上下文依赖关系、更准确的语言模式和更广泛的知识。然而，训练大型模型需要大量的计算资源和时间，因此一万亿tokens的训练是非常庞大的规模，并且很可能需要使用大规模分布式系统来完成。

ps: codegeex预训练是约8500亿tokens，1536个昇腾GPU训练了两个月。starcoder 1万亿tokens 在512个Tesla A100训练了24天。

## 编排（Orchestration）：Megatron-LM

在这里，“Megatron-LM进行编排”指的是使用Megatron-LM这个工具包来协调和管理模型训练过程的操作。Megatron-LM是一个用于训练大型语言模型的工具，它提供了一些功能和工具，使得大规模训练任务可以进行分布式处理和并行计算。

通过使用Megatron-LM进行编排，可以实现以下功能：

1. 分布式训练：Megatron-LM支持将模型训练任务分发到多个计算设备或计算节点上，以提高训练速度和效率。它可以处理分布式训练的各个方面，例如数据并行、模型并行、梯度聚合等。
2. 并行计算：Megatron-LM利用多个计算设备的并行计算能力，通过将模型参数划分为多个部分，并同时计算它们，以加速训练过程。这种并行计算可以提高大型模型训练的效率。
3. 资源管理：Megatron-LM帮助管理和优化计算资源的使用，以确保训练过程的高效性和可扩展性。它可以自动分配和管理计算设备、内存和存储资源，以最大程度地利用可用资源并避免资源浪费。

通过Megatron-LM的编排功能，可以更好地组织和管理模型训练过程，以提高训练效率、加速模型收敛并处理大规模语言模型训练任务。

## PlayGround

huggingface构建了在线体验，这里也是体现出一些功能

### 1. Prefixes 🌟 1.前缀🌟

For pure code files, use any combination of the following prefixes:

对于纯代码文件，请使用以下前缀的任意组合：

```
<repename>REPONAME<filename>FILENAME<gh_stars>STARS  
code<|endoftext|>
```

STARS can be one of: 0, 1-10, 10-100, 100-1000, 1000+

STARS 可以是以下之一：0、1-10、10-100、100-1000、1000+

### 2. Commits 🎉 2. 承诺🎉

The commits data is formatted as follows:

提交数据的格式如下：

```
<commit_before>code<commit_msg>text<commit_after>code<|endoftext|>
```

### 3. Jupyter Notebooks 📈 3.Jupyter 笔记本💻

The model is trained on Jupyter notebooks as Python scripts and structured formats like:

该模型在Jupyter笔记本上作为Python脚本和结构化格式进行训练，例如：

```
<start_jupyter><jupyter_text>text<jupyter_code>code<jupyter_output>output<jupyter_text>
```

### 4. Issues 🐛 4.问题🐛

We also trained on GitHub issues using the following formatting:

我们还使用以下格式对GitHub问题进行了培训：

```
<issue_start><issue_comment>text<issue_comment>...<issue_closed>
```

### 5. Fill-in-the-middle ❄️ 5. 中间填空❄️

Fill in the middle requires rearranging the model inputs. The playground handles this for you - all you need is to specify where to fill:

中间填充需要重新排列模型输入。playground会为你处理这个——你只需要指定填充的位置：

```
code before<FILL_HERE>code after
```

尝试了第五个功能

Code

```
int a = 10;<FILL_HERE>;int c = 30;
```

Generate

Code

```
1 int a = 10; int b=20;int c = 30;
```

...

## 指令微调

这不是一个指令模型。如需指导和聊天，您可以在StarChat Playground与模型的微调版本聊天

该模型是在 GitHub 代码上训练的。因此，它不是一个指令模型和命令，如“编写一个计算平方根的函数”。效果不佳。但是，通过使用 Tech Assistant 提示，您可以将其变成一个有能力的技术助手。

## bigcode/ta-prompt · Datasets at Hugging Face --- bigcode/ta-prompt · Hugging Face 数据集

此存储库专用于使用 starcoder 执行上下文学习的提示。事实上，该模型是一种自回归语言模型，在代码和自然语言文本上都进行了训练。通过将对话添加到其 8192 令牌上下文窗口，可以将其转变为人工智能技术助手。



Code-to-text:

What is the purpose of the following code?<code>  
What is the bug in the following code?<code>

Text-to-code:

Write/Design/Implement a function to <task>

Code-to-code:

Translate this <code> from <programming language> to <programming language>.

Text-to-text:

What is <technical concept>

General-purpose Q&A 通用问答

What are you?

What is your purpose?

微调文件只有20kb! !

<a href="#">TA_prompt_v0.txt</a>	10.2 kB	<a href="#">Upload TA_prompt_v0.txt</a>	29 days ago
<a href="#">TA_prompt_v1.txt</a>	11.8 kB	<a href="#">Update TA_prompt_v1.txt</a>	28 days ago

## 开源协议

### [BigCode Model License Agreement – a Hugging Face Space by bigcode](#)

## 部署和使用

全网并没找到部署的傻瓜式一键包

我们只能拿到最基本的模型后，自己写代码调用！

[bigcode-project/starcoder: Home of StarCoder: fine-tuning & inference! \(github.com\)](#)

模型约64GB

merges.txt	442 kB		add model	
pytorch_model-00001-of-00007.bin	9.9 GB			upload correct starcoder
pytorch_model-00002-of-00007.bin	9.86 GB			upload correct starcoder
pytorch_model-00003-of-00007.bin	9.85 GB			upload correct starcoder
pytorch_model-00004-of-00007.bin	9.86 GB			upload correct starcoder
pytorch_model-00005-of-00007.bin	9.85 GB			upload correct starcoder
pytorch_model-00006-of-00007.bin	9.86 GB			upload correct starcoder
pytorch_model-00007-of-00007.bin	4.08 GB			upload correct starcoder

## 补充的链接和资料

### 模型

- [论文](#): 关于 StarCoder 的技术报告。
- [GitHub](#): 你可以由此获得有关如何使用或微调 StarCoder 的所有信息。
- [StarCoder](#): 基于 Python 数据集进一步微调 StarCoderBase 所得的模型。
- [StarCoderBase](#): 基于来自 The Stack 数据集的 80 多种编程语言训练而得的模型。
- [StarEncoder](#): 在 The Stack 上训练的编码器模型。
- [StarPii](#): 基于 StarEncoder 的 PII 检测器。

### 工具和应用演示

- [StarCoder Chat](#): 和 StarCoder 聊天!
- [VSCode Extension](#): 使用 StarCoder 补全代码的 VSCode 插件!
- [StarCoder Playground](#): 用 StarCoder 写代码!
- [StarCoder Editor](#): 用 StarCoder 编辑代码!

### 数据与治理

- [StarCoderData](#): StarCoder 的预训练数据集。
- [Tech Assistant Prompt](#): 使用该提示，你可以将 StarCoder 变成技术助理。
- [Governance Card](#): 有关模型治理的卡片。
- [StarCoder License Agreement](#): 该模型基于 BigCode OpenRAIL-M v1 许可协议。
- [StarCoder Search](#): 对预训练数据集中的代码进行全文搜索。

- ↗ [StarCoder Membership Test](#): 快速测试某代码是否存在于预训练数据集中。

你可以在 ↗ [huggingface.co/bigcode](https://huggingface.co/bigcode) 找到所有资源和链接!

## 论文解读

---

4096个token, 就gpt-3.5-turbo-0301而言



gpt-3.5-turbo 的标记限制为 4096 个标记, 而 gpt-4 和 gpt-4-32k 的标记限制分别为 8192 和 32768。

从 the stack (6.4TB的代码数据集) 的358种编程语言, 选择了86种, 同时还有人工审查

Language	After dedup		After filters and decont.		Weight	Percentage
	Num. files	Volume (GB)	Num. files	Volume (GB)		
ada	31,291	0.30	30,934	0.26	0.26	0.034
agda	17,608	0.07	17,554	0.07	0.07	0.009
alloy	5,374	0.01	5,368	0.01	0.01	0.001
antlr	7,983	0.05	7,917	0.05	0.05	0.007
applescript	4,906	0.01	4,737	0.01	0.01	0.001
assembly	248,396	1.58	247,919	1.56	1.56	0.203
augeas	195	0.00	180	0.00	0.00	0
awk	10,430	0.02	10,289	0.02	0.02	0.003
batchfile	252,514	0.29	239,568	0.23	0.23	0.03
bluespec	5,940	0.03	5,928	0.03	0.03	0.004
c	8,625,559	57.43	8,536,791	53.89	53.89	7.027
c-sharp	10,839,399	46.29	10,801,285	44.66	44.66	5.823
clojure	126,191	0.49	125,163	0.46	0.46	0.06
cmake	186,517	0.45	186,375	0.45	0.45	0.059
coffeescript	227,889	0.69	226,209	0.64	0.64	0.083
common-lisp	101,370	1.68	98,733	1.40	1.40	0.183
cpp	6,377,914	50.89	6,353,527	48.92	48.92	6.379
css	2,994,829	22.61	2,721,616	11.93	3.00	0.391
cuda	58,355	0.59	58,151	0.56	0.56	0.073
dart	932,583	3.86	928,415	3.66	3.66	0.477
dockerfile	572,186	0.42	571,506	0.42	0.42	0.055
elixir	282,110	0.74	281,016	0.71	0.71	0.093
elm	62,861	0.34	62,033	0.30	0.30	0.039
emacs-lisp	54,768	0.43	52,838	0.41	0.41	0.053
erlang	99,368	0.73	98,447	0.70	0.70	0.091
f-sharp	127,161	0.90	124,066	0.61	0.61	0.08
fortran	165,446	1.84	158,792	1.78	1.78	0.232
glsl	175,576	0.57	167,701	0.40	0.40	0.052
go	4,730,461	25.74	4,700,526	23.78	23.78	3.101
groovy	251,627	0.94	250,834	0.91	0.91	0.119
haskell	544,969	2.36	541,454	2.23	2.23	0.291
html	9,533,367	146.76	3,299,965	29.36	29.36	3.828
idris	8,060	0.03	8,042	0.03	0.03	0.004
isabelle	5,086	0.09	5,001	0.08	0.08	0.01
java	20,151,565	89.30	20,071,773	86.94	86.94	11.336
java-server-pages	214,133	1.03	210,816	0.98	0.98	0.128
javascript	21,108,587	141.65	19,544,285	64.71	64.71	8.437
json	17,012,912	338.34	4,751,547	5.62	1.00	0.13
julia	298,672	1.54	295,364	1.31	1.31	0.171
kotlin	2,242,771	5.77	2,239,354	5.68	5.68	0.741
lean	16,891	0.10	16,870	0.09	0.09	0.012
literate-agda	523	0.01	523	0.01	0.01	0.001
literate-coffeescript	1,138	0.01	1,133	0.01	0.01	0.001
literate-haskell	6,135	0.05	6,104	0.05	0.05	0.007
lua	558,861	3.28	549,459	2.87	2.87	0.374
makefile	661,424	1.49	657,349	1.31	1.31	0.171
maple	1,259	0.01	1,152	0.01	0.01	0.001
markdown	21,045,171	75.25	21,029,287	74.93	74.93	9.77
mathematica	26,895	1.72	22,653	1.25	1.25	0.163
matlab	967	0.04	93	0.00	0.00	0

Language	After dedup		After filters and decont.		Weight	Percentage
	Num. files	Volume (GB)	Num. files	Volume (GB)		
ocaml	159,734	1.11	158,356	1.03	1.03	0.134
pascal	118,675	1.71	110,981	1.68	1.68	0.219
perl	392,108	2.63	365,491	2.23	2.23	0.291
php	15,904,518	66.84	15,683,017	60.89	60.89	7.939
powershell	271,487	1.25	267,627	1.12	1.12	0.146
prolog	1,023	0.01	968	0.01	0.01	0.001
protocol-buffer	98,246	0.44	97,167	0.31	0.31	0.04
python	12,962,249	64.30	12,866,649	60.40	60.40	7.875
r	39,194	0.30	39,042	0.30	0.30	0.039
racket	4,201	0.04	3,688	0.03	0.03	0.004
restructuredtext	905,679	3.42	896,880	3.32	3.32	0.433
rmarkdown	5,389	0.06	5,386	0.06	0.06	0.008
ruby	3,405,374	7.14	3,390,320	6.81	6.81	0.888
rust	1,386,585	9.53	1,380,468	9.11	9.11	1.188
sas	9,772	0.13	9,226	0.12	0.12	0.016
scala	1,362,426	4.86	1,355,788	4.69	4.69	0.612
scheme	44,261	0.30	41,890	0.20	0.20	0.026
shell	2,236,434	3.38	2,206,327	3.09	3.09	0.403
smalltalk	592,999	0.74	587,748	0.58	0.58	0.076
solidity	164,242	1.21	153,194	0.85	0.85	0.111
sparql	14,173	0.04	13,716	0.04	0.04	0.005
sql	994,019	12.22	975,420	11.09	11.09	1.446
stan	5,441	0.01	5,429	0.01	0.01	0.001
standard-ml	48,995	0.52	19,630	0.19	0.19	0.025
stata	31,282	0.41	24,208	0.33	0.33	0.043
systemverilog	46,915	0.41	46,270	0.39	0.39	0.051
tcl	50,579	0.40	49,335	0.35	0.35	0.046
tcsh	4,911	0.02	4,806	0.02	0.02	0.003
tex	547,888	5.44	522,778	5.20	5.20	0.678
thrift	4,663	0.01	4,661	0.01	0.01	0.001
typescript	10,637,070	28.82	10,547,331	26.52	26.52	3.458
verilog	77	0.001	75	0.001	0.001	0
vhdl	60,027	1.12	58,208	0.94	0.94	0.123
visual-basic	163,291	1.49	161,239	1.42	1.42	0.185
xslt	43,095	0.56	6,513	0.05	0.05	0.007
yacc	25,775	0.41	7,451	0.11	0.11	0.014
yaml	5,282,081	28.36	3,995,948	3.76	1.00	0.13
zig	15,913	0.18	15,850	0.18	0.18	0.023
GitHub issues		~ 30,900,000	54.40	54.40	7.093	
Git commits		7,674,345	64.00	32.00	4.172	
notebook scripts		914,000	7.12	7.12	0.928	
notebook structured		668,743	6.00	6.00	0.782	
		305,929,658	815.68	799.37	100	

StarCoder是StarCodeBase的微调版本经过35B的python tokens训练过

## 与CodeGeex对比

Model	HumanEval	MBPP
LLaMA-7B	10.5	17.7
LaMDA-137B	14.0	14.8
LLaMA-13B	15.8	22.0
CodeGen-16B-Multi	18.3	20.9
LLaMA-33B	21.7	30.2
CodeGeeX	22.9	24.4
LLaMA-65B	23.7	37.7
PaLM-540B	26.2	36.8
CodeGen-16B-Mono	29.3	35.3
StarCoderBase	30.4	49.0
code-cushman-001	33.5	45.9
StarCoder	33.6	<b>52.7</b>
StarCoder-Prompted	<b>40.8</b>	49.5

	StarCoder	Codegeex
参数量	15.5B	13B
预训练数据集	the track v1.2	Pile、CodeParrot
预训练数据集大小	6.4TB(清洗前)	约900GB
预训练数据集是否开源	是	是
预训练数据集内容	86种编程语言代码、jupyter笔记、github issue、git 提交等	23种编程语言代码、Pile(Common Crawl,PMC,Arxiv,Books3,OpenWebText2 )
预训练tokens数	1000B	850B (5+遍158B)
预训练耗时	512个Tesla A100训练24天	1536个Ascend 910 训练60天
HumanEval评分	33.6	22.9
MBPP评分	52.7	24.4
是否提供finetune步骤	是	否
编程IDE插件	是	是

HumanEval评分是python基准

4. **code-cushman-001**是OpenAI的12B参数模型，是GitHub Copilot的初始模型([Chen et al., 2021](#))。它的训练集的细节是未知的。此模型已被OpenAI弃用，但在撰写本文时，可以从Microsoft Azure OpenAI服务中获得。<sup>13</sup>

论文提到了8K上下文窗口的好处

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	<b>1.79</b>	<b>1.66</b>	<b>1.61</b>	<b>1.21</b>	<b>1.54</b>	<b>1.68</b>	<b>1.43</b>	<b>1.48</b>	<b>2.02</b>	<b>1.65</b>

Table 20: 当在10种语言的存储库中使用2K或8K标记的窗口大小时，StarCoderBase在评估区域(大小为1K标记)上的困惑。更大的窗口大小大大减少了困惑，展示了StarCoder的8K令牌窗口的好处。

## 创新研讨会

简介：HuggingFace开源代码补全模型StarCoder 及其衍生对话模型StarChat

议题类型：新技术研究

价值：

StarCoder 模型是 15.5B 参数模型，使用 The Stack (v1.2) 中的 80 多种编程语言进行训练。该模型支持8K上下文窗口，并使用中间填空目标对 1 万亿个标记进行训练。

StarCoder 自发布到现在一个月的时间，已经在HuggingFace上有57.6K的下载量。StarChat也有10.2K的下载量。

StarCoder 的HumanEval评分不仅超越Codegeex，甚至与Code-cushman-001齐平。

StarCoder 拥有8192个上下文窗口（论文也证实了更大的上下文窗口可以提升生成结果的质量。gpt-3.5-turbo 的上下文窗口为 4096，而 gpt-4 和 gpt-4-32k 分别为 8192 和 32768。）

StarChat是在StarCoder基础上通过极少文本学习而激发得到的对话模型，能够回答编程方面的大部分问题，包括但不限于代码注释生成、代码跨语言翻译、单元测试生成等。

单个模型约占用30+GB显存，可部署到单张A100上。

相对于行内采购的aiXcoder，StarCoder开源了更多的技术细节，从预训练的数据集、数据整理，预训练过程、如何微调、如何都提供了相应的介绍，研究StarCoder对于我行智能研发领域有极大的提升作用。

应用场景：智能研发。

附件：

重点是，StarCoder是通过训练代码而来的，经过微调后激发出对话能力，

**StarChat**

## ☞ [HuggingFaceH4/starchat-alpha at main](#)

有两个模型

merges.txt	442 kB		Duplicate from HuggingFaceH4/starcod...	
model-00001-of-00004.safetensors	9.96 GB			Duplicate from HuggingFaceH4/starcod...
model-00002-of-00004.safetensors	9.86 GB			Duplicate from HuggingFaceH4/starcod...
model-00003-of-00004.safetensors	9.86 GB			Duplicate from HuggingFaceH4/starcod...
model-00004-of-00004.safetensors	1.36 GB			Duplicate from HuggingFaceH4/starcod...
model.safetensors.index.json	36.2 kB		Duplicate from HuggingFaceH4/starcod...	
pytorch_model-00001-of-00004.bin	9.96 GB			Duplicate from HuggingFaceH4/starcod...
pytorch_model-00002-of-00004.bin	9.86 GB			Duplicate from HuggingFaceH4/starcod...
pytorch_model-00003-of-00004.bin	9.86 GB			Duplicate from HuggingFaceH4/starcod...
pytorch_model-00004-of-00004.bin	1.97 GB			Duplicate from HuggingFaceH4/starcod...
pytorch_model.bin.index.json	36.3 kB		Duplicate from HuggingFaceH4/starcod...	

硬件要求：在 FP32 中，模型需要超过 60GB 的 RAM，您可以在 FP16 或 BF16 中将其加载到 ~30GB，或者在 20GB RAM 下的 8bit 中

看了篇知乎的介绍：[知 使用 StarCoder 创建一个编程助手 - 知乎 \(zhihu.com\)](#)

其他可参考的文章：

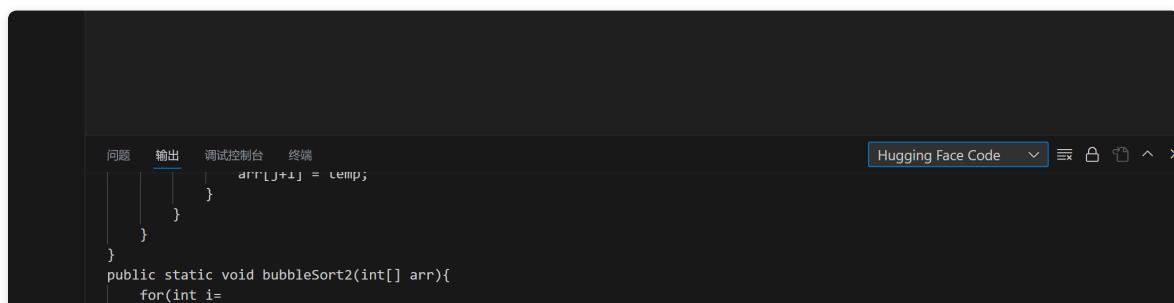
[51 秒杀自动编码Copilot! 「动嘴编程」神器StarChat开源，码农狂喜-51CTO.COM](#)

webservice

[huggingface/text-generation-inference: Large Language Model Text Generation Inference --- huggingface/text-generation-inference: 大型语言模型文本生成推理 \(github.com\)](#)

## 插件的使用

如果想看插件的输出，可以如图设置



插件的源码：

⌚ [huggingface-vscode/src at master · huggingface/huggingface-vscode · GitHub](#)

插件的请求都在这了：

⌚ [huggingface-vscode-endpoint-server/main.py at main · LucienShui/huggingface-vscode-endpoint-server \(github.com\)](#)

插件是模仿的tabnine：

⌚ [codota/tabnine-vscode: Visual Studio Code client for Tabnine.](#)

[https://marketplace.visualstudio.com/items?itemName=TabNine.tabnine-vscode \(github.com\)](https://marketplace.visualstudio.com/items?itemName=TabNine.tabnine-vscode (github.com))

## HuggingFace的使用

使用

☒ [Getting Started with Repositories --- 开始使用存储库 \(huggingface.co\)](#)



cmd

```
python -m pip install huggingface_hub  
huggingface-cli login
```

这个文解释了如何真的本地化部署

知 [huggingface transformers预训练模型如何下载至本地，并使用？ - 知乎 \(zhihu.com\)](#)

知 [如何优雅的下载huggingface-transformers模型 - 知乎 \(zhihu.com\)](#)