

DeepSeek - Coder : 当大型语言模型遇到编程时 - 代码智能的兴起

郭大雅^{* 1}, 朱启浩^{* 1,2}, 杨德建¹, 谢振达¹, 启东¹, 张文涛¹
陈官厅¹, 小碧¹, Y. Wu¹, Y.K. Li¹, 罗富力¹, 熊英飞², 梁文峰¹

¹ DeepSeek - AI
² HCST (PKU) 教育部重点实验室 ; 北京大学 SCS
{zhuqh, guodaya} @ deepseek. com <https://github.com/deepseek-ai/DeepSeek-Coder>

Abstract

大型语言模型的快速发展已彻底革新了软件开发中的代码智能。然而，主要依赖闭源模型限制了广泛的研究与开发。为解决这一问题，我们推出了DeepSeek-Coder系列开源代码模型，其规模从13亿到330亿不等，并从2万亿个标记中从头开始训练。这些模型预先在高质量的项目级代码语料库上进行训练，并采用16K窗口的填空任务来增强代码生成和填补。我们的广泛评估表明，DeepSeek-Coder不仅在多个基准测试中实现了开源代码模型的最佳性能，还超越了现有的闭源模型如Codex和GPT-3.5。此外，DeepSeek-Coder模型处于宽松的许可之下，允许同时进行研究和无限制的商业使用。

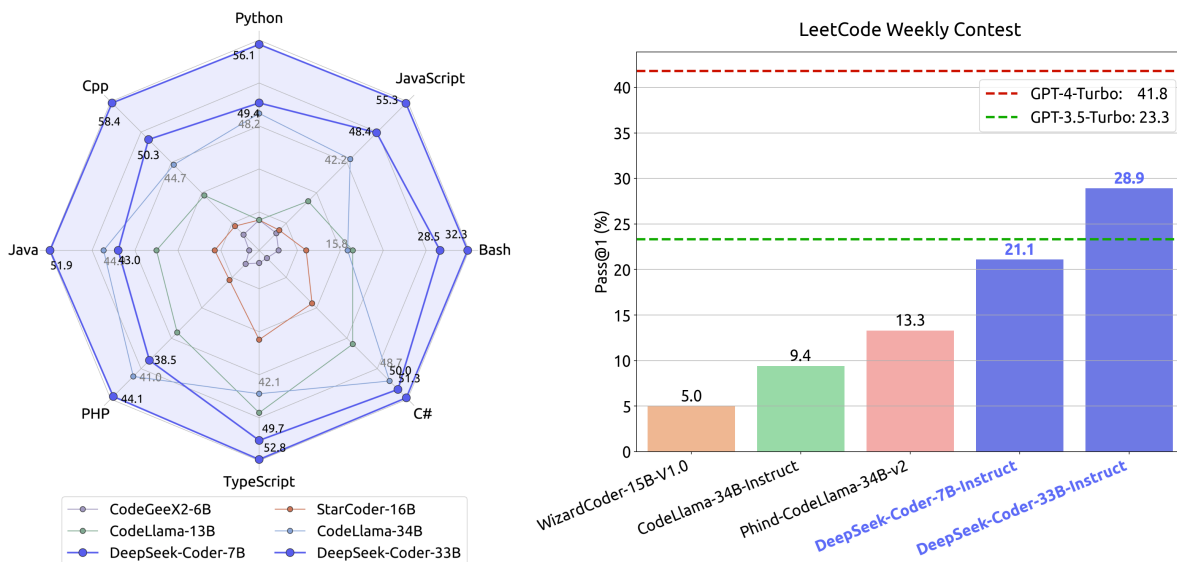


图 1 | DeepSeek - Coder 的性能

* 核心贡献者，按名称字母顺序排序。

1. Introduction

软件开发领域因大型语言模型的迅速进步 (OpenAI, 2023; Touvron et al., 2023) 而得到了显著转型, 这些模型带来了代码智能的新时代。这些模型有可能自动化和简化许多编码方面的工作, 从错误检测到代码生成, 从而提高生产力并减少人为错误的可能性。然而, 该领域的重大挑战之一是在开源模型 (Li et al., 2023; Nijkamp et al., 2022; Rozier et al., 2023; Wang et al., 2021) 与闭源模型 (Gemini Team, 2023; OpenAI, 2023) 之间的性能差距。尽管巨型闭源模型非常强大, 但由于其专有性质, 许多研究人员和开发者仍然难以访问这些模型。

为了应对这一挑战, 我们推出了DeepSeek-Coder系列。该系列包括一系列开源代码模型, 规模从13亿到33亿不等, 涵盖每个规模的基版本和指令版本。每个系列中的模型均从87种编程语言中提取的2万亿个标记重新训练, 确保对编程语言和语法有全面的理解。此外, 我们尝试在仓库级别组织预训练数据, 以增强模型在仓库内部跨文件上下文中的理解能力。除了在预训练过程中使用下一个标记预测损失外, 我们还引入了Fill-In-Middle (FIM) 方法 (Bavarian et al., 2022; Li et al., 2023)。这种方法旨在进一步提升模型的代码完成能力。为满足处理更长代码输入的要求, 我们将上下文长度扩展到16K。这一调整使我们的模型能够处理更加复杂和广泛的编码任务, 从而增强了其在各种编码场景中的适用性和灵活性。

我们使用多种公开的代码相关基准进行了全面的实验。研究发现, 在开源模型中, DeepSeek-Coder-Base 33B 在所有基准测试中均表现出优越的性能。此外, *OpenAI GPT - 3.5 Turbo* Coder - Instruct 33B 超越 在大多数评估基准中, *OpenAI GPT - 4* 显著缩小两者之间的性能差距 并且采用了开源模型。令人惊讶的是, 尽管参数量较少, DeepSeek-Coder-Base 7B 在与参数量大五倍的模型 (如 CodeLlama-33B) 相比时仍能展现出竞争力的表现 (Roziere et al., 2023)。总结来说, 我们的主要贡献是:

- 我们引入了DeepSeek-Coder-Base和DeepSeek-Coder-Instruct, 这两种先进的代码集中型大型语言模型 (LLMs)。通过广泛训练于庞大的代码语料库, 这些模型展示了对87种编程语言的理解能力。此外, 它们还提供了多种模型规模以满足不同计算和应用需求。我们首次尝试在模型构建过程中纳入仓库级别的数据。
- 我们的模型预训练阶段。我们发现这可以显著提升跨文件代码生成的能力。
- 我们的研究严格分析了FIM训练策略对代码模型预训练阶段的影响。这些全面的研究结果揭示了FIM配置的一些有趣方面, 提供了宝贵的见解, 极大地促进了代码预训练模型的改进和发展。
- 我们对我们的代码LLM进行了广泛的评估, 涵盖了多种代码相关任务的广泛基准。研究发现表明, DeepSeek-Coder-Base 在这些基准上超越了所有现有的开源代码LLM。此外,

通过使用教学数据进行细致的微调，DeepSeek - Coder - Instruct 实现了 *OpenAI GPT - 3.5 Turbo* 更好的性能相比与代码相关的任务中的模型。

2. 数据收集

DeepSeek-Coder 的训练数据集由87%的源代码、10%的英语代码相关自然语言语料库和3%的代码无关的中文自然语言语料库组成。英语语料库来自GitHub的Markdown和StackExchange材料。¹，这些数据用于增强模型对代码相关概念的理解，并提高其处理库使用和错误修复等任务的能力。与此同时，中文语料库由高质量的文章组成，旨在提高模型理解中文语言的能力。在本节中，我们将概述我们构建代码训练数据的过程。这一过程包括数据爬取、基于规则的过滤、依赖解析、仓库级别去重以及质量筛选，如图2所示。接下来，我们将逐步描述数据创建流程。

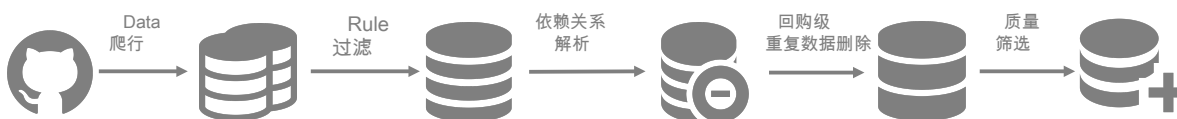


图 2 | 创建数据集的过程

2.1. GitHub 数据抓取和过滤

我们在GitHub上收集了截至2023年2月之前创建的公共仓库，并仅保留了87种编程语言（如表1所示）。为了减少处理的数据量，我们应用了与StarCoder项目（Li等，2023）中使用的规则类似的过滤规则，初步筛选掉质量较低的代码。通过应用这些过滤规则，我们将数据总量减少了至原始大小的32.8%。为了使论文更加自包含，我们简要描述了StarCoder Data项目中使用的过滤规则：

首先，我们过滤掉平均行长度超过100字符或最大行长度超过1000字符的文件。此外，我们去除包含少于25%字母字符的文件。除了XSLT编程语言外，我们进一步过滤掉其他类型的文件

。"`<? xml 版本 =`"

其中字符串 出现在前100个字符中。对于HTML文件，我们考虑可见文本与HTML代码的比例。我们保留那些可见文本至少占代码20%且不少于100个字符的文件。对于通常包含更多数据的JSON和YAML文件，我们仅保留字符计数在50到5000之间的文件。这有效地移除了大多数数据密集型文件。

2.2. Dependency Parsing

在之前的研究中（Chen et al., 2021；Li et al., 2023；Nijkamp et al., 2022；Rozier et al., 2023），大型语言模型主要在文件级别源代码上进行预训练，忽略了项目中不同文件之间的依赖关系。然而，在实际应用中，这类模型难以有效地扩展以处理整个项目级别的代码场景。因此，我们

¹ <https://stackexchange.com>

算法 1 依赖分析的拓扑排序

```
1: procedure TOPOLOGICAL SORT (files)
2:   graphs  $\leftarrow \{\}$                                 ▶ 初始化空邻接列表
3:   inDegree  $\leftarrow \{\}$                             ▶ 初始化 in - degrees 的空字典
4:   对于每个 file in files do
5:     graphs[file]  $\leftarrow []$ 
6:     inDegree[file]  $\leftarrow 0$ 
7:   结束
8:
9:   对于每个 fileA in files do
10:    对于每个 fileB in files do
11:      if HAS DEPENDENCY (fileA, fileB) then          ▶ 如果 fileA 依赖于 fileB
12:        graphs[fileB]. append (fileA)              ▶ 将边从 B 添加到 A
13:        inDegree[fileA]  $\leftarrow$  inDegree[fileA] + 1  ▶ A 程度的增量
14:      如果结束
15:    结束
16:  结束
17:
18:  subgraphs getDisconnectedSubgraps ( graphs)      ▶ 识别断开的子图
19:  allResults  $\leftarrow []$ 
20:  对于每个 subgraph in subgraphs do
21:    results  $\leftarrow []$ 
22:    while 长度 ( results)  $\times$  NumberOfNodes (subgraph) do
23:      file = argmin ( { inDegree[file] | file  $\in$  subgraph and file  $\notin$  results })
24:      对于每个 node in graphs[file] do
25:        inDegree[node]  $\leftarrow$  inDegree[node] - 1
26:      结束
27:      results. append (file)
28:    结束时
29:    allResults. append (results)
30:  结束
31:
32:  返回 allResults
33: 结束程序
```

我们将考虑如何在同一仓库内利用文件之间的依赖关系。具体来说，我们首先解析文件间的依赖关系，然后按照确保每个文件所依赖的上下文出现在该文件之前的顺序排列这些文件。通过根据依赖关系对文件进行对齐，我们的数据集更准确地反映了实际的编码实践和结构。这种增强的对齐不仅使数据集更具相关性，还可能提高模型在处理项目级代码场景时的实用性和适用性。值得注意的是，我们仅考虑文件之间的调用关系，并使用正则表达式提取这些关系，例如"导入"在 Python 中，"使用"在 C # 中，和"包括" in C.

该算法1描述了一个项目内的文件列表中进行依赖分析的拓扑排序。初始步骤是设置两个数据结构：一个名为空邻接表。"图形"表示文件和空字典之间的依赖关系"学位"用于存储每个文件的 in - degrees。然后，算法迭代每个文件对，以识别

驻留,更新 "图形" and "学位" 相应地,接下来识别整体依赖图中的任何孤立子图。对于每个子图,算法采用修改后的拓扑排序。与标准方法选择入度为零的节点不同,该算法选择入度最小的节点,这使其能够处理图中的循环。选定的节点被添加到 "结果" 列表中的节点及其连接节点的入度减少。这一过程将持续进行,直到为每个子图生成一个拓扑排序序列。算法最终返回这些排序序列的列表,并将每个序列中的文件连接起来形成单个训练样本。为了包含文件路径信息,在每个文件开头添加一条注释以指示文件的路径。这种方法确保了路径信息保留在训练数据中。

2.3. Repo 级重复数据删除

最近的研究表明,通过去重训练数据集可以显著提高大型语言模型 (LLMs) 的性能。李等人 (2022) 发现语言模型训练语料库中往往包含大量的近似重复内容,并且通过移除长篇重复子字符串可以提升LLM的性能。科切托夫等人 (2022) 应用了近似去重方法到训练数据上,取得了显著的改进,并强调近似去重是实现竞争性性能的关键预处理步骤。在我们的数据集中,我们也采用了近似去重的方法。然而,与以往的工作相比,我们采取了不同的方法。我们是在代码仓库级别进行去重,而不是在文件级别进行,因为后者可能会过滤掉仓库内的某些文件,从而可能破坏仓库结构。具体而言,我们将仓库级别的所有代码拼接成一个样本,并应用相同的近似去重算法以确保仓库结构的完整性。

2.4. 质量筛选和去污

除了应用第2.1节中提到的过滤规则外,我们还使用了一个编译器和一个质量模型,并结合启发式规则进一步过滤掉低质量的数据。这包括包含语法错误、可读性差和低模块化的代码。我们在表1中提供了源代码的统计摘要,其中包括87种语言的总览,详细列出了每种语言的磁盘大小、文件数量以及所占百分比。总数据量为798 GB,共有6亿个文件。为了确保我们的代码训练数据不会受到来自测试集的信息污染,这些信息可能存在于GitHub上,我们实施了n-gram过滤过程。该过程涉及移除符合特定标准的任何代码片段。具体而言,我们排除了包含来自HumanEval (Chen等, 2021)、MBPP (Austin等, 2021)、GSM8K (Cobbe等, 2021) 和MATH (Hendrycks等, 2021) 等来源的文档字符串、问题和解决方案文件。对于过滤标准,我们应用了以下规则:如果一段代码包含与测试数据中任何一个10-gram字符串完全相同的字符串,则将其从训练数据中排除。当测试数据由短于10-gram但不少于3-gram的字符串组成时,我们采用精确匹配的方法进行过滤。

语言	大小 (GB)	文件 (k)	Prop. (%)	语言	大小 (GB)	文件 (k)	Prop. (%)
Ada	0.91	126	0.11	Literate Haskell	0.16	20	0.02
Agda	0.26	59	0.03	Lua	0.82	138	0.10
合金	0.07	24	0.01	Makefile	0.92	460	0.12
ANTLR	0.19	38	0.02	枫木	0.03	6	0.00
AppleScript	0.03	17	0.00	Mathematica	0.82	10	0.10
装配	0.91	794	0.11	MATLAB	0.01	1	0.00
Augeas	0.00	1	0.00	OCaml	0.91	139	0.11
AWK	0.09	53	0.01	帕斯卡	0.79	470	0.10
Batchfile	0.92	859	0.12	Perl	0.81	148	0.10
Bluespec	0.10	15	0.01	PHP	58.92	40,627	7.38
C	28.64	27,111	3.59	PowerShell	0.91	236	0.11
C#	58.56	53,739	7.34	Prolog	0.03	5	0.00
Clojure	0.90	295	0.11	协议缓冲区	0.92	391	0.12
CMake	0.90	359	0.11	Python	120.68	75,188	15.12
CoffeeScript	0.92	361	0.12	R	0.92	158	0.11
Common Lisp	0.92	105	0.11	球拍	0.09	13	0.01
C++	90.87	36,006	11.39	RMarkdown	6.83	1,606	0.86
CSS	5.63	11,638	0.71	Ruby	15.01	18,526	1.88
CUDA	0.91	115	0.11	Rust	0.61	692	0.08
Dart	0.89	264	0.11	SAS	0.92	70	0.11
Dockerfile	0.04	48	0.00	Scala	0.81	971	0.10
长生不老药	0.91	549	0.11	方案	0.92	216	0.12
Elm	0.92	232	0.12	壳体	13.92	10,890	1.74
Emacs Lisp	0.91	148	0.11	Smalltalk	0.92	880	0.12
Erlang	0.92	145	0.12	Solidity	0.85	83	0.11
F#	0.91	340	0.11	Sparql	0.10	88	0.01
Fortran	1.67	654	0.21	SQL	15.14	7,009	1.90
GLSL	0.92	296	0.11	Stan	0.20	41	0.03
Go	2.58	1,365	0.32	标准 ML	0.74	117	0.09
Groovy	0.89	340	0.11	Stata	0.91	122	0.11
Haskell	0.87	213	0.11	SystemVerilog	0.91	165	0.11
HTML	30.05	14,998	3.77	TCL	0.90	110	0.11
伊德里斯	0.11	32	0.01	Tcsh	0.17	53	0.02
伊莎贝尔	0.74	39	0.09	Tex	20.46	2,867	2.56
Java	148.66	134,367	18.63	节俭	0.05	21	0.01
Java 服务器页	0.86	1072	0.11	TypeScript	60.62	62,432	7.60
JavaScript	53.84	71,895	6.75	Verilog	0.01	1	0.00
JSON	4.61	11956	0.58	VHDL	0.85	392	0.11
朱莉娅	0.92	202	0.12	Visual Basic	0.75	73	0.09
Jupyter 笔记本	14.38	2,555	1.80	XSLT	0.36	48	0.04
Kotlin	6.00	3,121	0.75	Yacc	0.72	67	0.09
Lean	0.52	68	0.07	YAML	0.74	890	0.09
识字 Agda	0.05	4	0.01	Zig	0.81	70	0.10
Literate CoffeeScript	0.01	3	0.00	Total	797.92	603,173	100.00

表 1 | 所选编程语言的清理训练数据的摘要。

3. 培训政策

3.1. 培训策略

3.1. 1. Next Token 预测

下一个令牌预测

我们模型的第一个训练目标被称为 在这一过程中，各种文件被连接以形成固定长度的条目。然后，使用这些条目训练模型，使其能够根据提供的上下文预测后续的标记。

3.1. 2. Fill - in - the - middle

填充中间。

我们模型的第二个训练目标被称为 在代码预训练场景中，通常需要根据给定的上下文和后续文本生成相应的插入内容。由于编程语言中的特定依赖关系，仅依靠下一个标记预测不足以学习这种中间填充能力。因此，一些方法（如Bavarian等，2022；Li等，2023）提出了Fill-in-the-Middle（FIM）的预训练方法。该方法涉及随机将文本分为三部分，然后打乱这三部分的顺序并用特殊字符连接它们。这种方法旨在在训练过程中融入一种填空预训练任务。在FIM方法中，使用了两种不同的模式：Prefix-Suffix-Middle（PSM）和Suffix-Prefix-Middle（SPM）。在PSM模式下，训练语料库按以下顺序组织：，，你：相反，SPM模式将段落排列方式设置为由前缀和后缀 flank 中间的部分。，，呈现了不同的结构性挑战。这些模式对于增强模型处理代码中各种结构排列的能力至关重要，并为高级代码预测任务提供了稳健的训练框架。

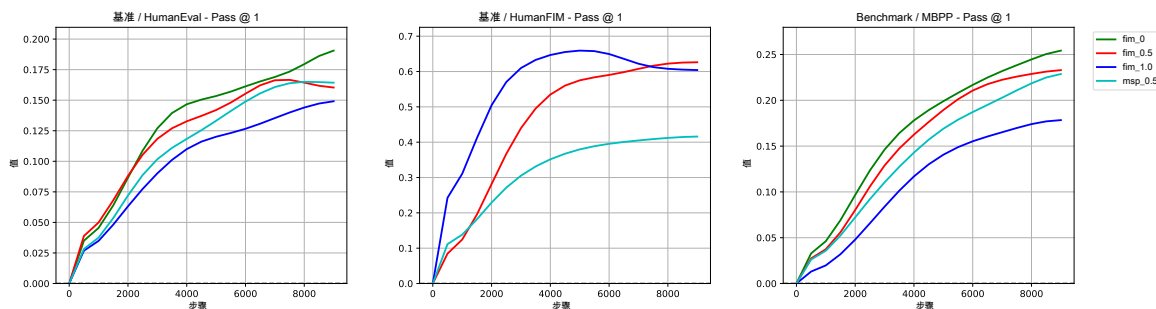


图 3 | 使用 FIM 目标的有效性。

为了确定FIM方法中各种超参数的有效性，我们进行了系列消融实验。

实验设置: 在本次实验中，我们采用DeepSeek-Coder-Base 1.3B作为模型架构。我们专注于训练数据集中的一部分Python子集，以简化实验过程。我们的主要目标是评估Fill-in-the-Middle（FIM）技术的有效性，并使用HumanEval-FIM基准（Fried等人，2022年）。该基准专门针对Python的一行FIM任务，在此任务中，随机遮蔽HumanEval解决方案中的一行代码，测试模型预测缺失部分的能力。我们假设PSM模式可能与传统的下一个标记预测目标存在细微差异。这主要是因为PSM涉及重新排列原始文本的顺序，可能会对模型的学习动态产生影响。因此，我们在四种不同的配置下实施了PSM模式进行FIM：0% FIM率、50% FIM率、100% FIM率和50% MSP率。屏蔽跨度预测（MSP）策略最初是在T5（Raffel等人，2023年）中引入的，它隐藏多个文本片段并训练模型重建这些片段。根据CodeGen2.5（Nijkamp等人，2023年），MSP可能在FIM性能上优于PSM。因此，我们将这种方法纳入了我们的比较分析中。

Results: 我们的实验结果如图3所示。虽然模型在HumanEval-FIM上表现出最佳性能，达到100%的FIM率，但这种配置也导致了最弱的代码完成能力。这表明FIM和代码完成能力之间存在权衡关系。

代码补全能力。此外，我们观察到当PSM（可能的手术治疗）率设定为50%时，模型的表现优于MSP（基于状态的策略）策略。为了在FIM（功能独立性测量）效率与代码补全熟练度之间取得平衡，我们最终选择了50%的PSM率为我们的首选训练策略。

在我们的实施中，我们特别引入了三种哨兵标记。对于每个代码文件，我们最初将其内容划分为三个部分，分别表示为 fim_start 、 fim_hole 和 fim_end 。

使用 PSM 模式，我们构造训练示例如下：

$\text{fim_start} | >_{f_{pre}} | \text{fim_hole} | >_{f_{mid}} | \text{fim_end} | >_{f_{middle}} | \text{eos_token} | >$

我们在打包过程之前，在文档级别实施了 Fill-in-the-Middle（FIM）方法，这与 Bavarian 等人的原始工作（2022 年）提出的建议一致。具体实施时采用 FIM 率为 0.5 的模式，遵循 PSM 模式。

3.2. Tokenizer

对于标记化过程，我们使用 HuggingFace Tokenizer 库² 为了训练字-piece 编码（BPE）分词器（Sennrich 等，2015），我们在训练语料库的一部分上根据 Sennrich 等（2015）的方法进行训练。最终，我们使用了一个词汇量大小为 32,000 的分词器。

3.3. 模型体系结构

我们开发了一系列具有不同参数的模型以满足多样化的应用需求，包括参数量为 1.3B、6.7B 和 33B 的模型。这些模型基于 DeepSeek-AI（2024）提出的 DeepSeek 大型语言模型（LLM）框架构建。每个模型均为解码器型 Transformer，采用了 Su 等人（2023）描述的旋转位置嵌入（RoPE）。值得注意的是，DeepSeek 33B 模型集成了分组查询注意力（GQA），分组大小为 8，从而提高了训练和推理效率。此外，我们还使用了 FlashAttention v2（Dao，2023）来加速注意力机制中的计算过程。我们的模型架构细节总结在表 2 中。

3.4. 优化

在 DeepSeek LLM (DeepSeek - AI，2024) 之后，我们使用 AdamW (Loshchilov 和 Hutter，2019) 作为优化器， β_1 and β_2 值分别为 0.9 和 0.95。我们通过以下方式调整批量大小和学习率

² <https://github.com/huggingface/tokenizers>

1 初始速率。值得注意的是，我们遵循 DeepSeek LLM (DeepSeek - AI，2024) 中确立的指导方针。我们实施了一个三阶段策略，包括 2000 个预热步长，并将最终学习率设置为 $\sqrt{\text{batch_size}}$ 的 10%。

3.5. Environments

我们的实验使用了 HAI-LLM (High-Flyer, 2023) 框架进行，该框架以其在训练大规模语言模型时的高效性和轻量级方法而闻名。该框架采用了多种并行策略来优化计算效率，包括张量并行 (Korthikanti 等，2023)、ZeRO 数据并行 (Rajbhandari 等，2020) 以及 PipeDream 管道并行 (Narayanan 等，2019)。我们的实验

² <https://github.com/huggingface/tokenizers>

超参数	DeepSeek - Coder 1.3B	DeepSeek - Coder 6.7B	DeepSeek - 编码器 33B
隐藏激活	SwiGLU	SwiGLU	SwiGLU
隐藏大小	2048	4096	7168
中间尺寸	5504	11008	19200
隐藏层数	24	32	62
注意力头数	16	32	56
Attention	多头	多头	分组查询 (8)
批量大小	1024	2304	3840
最大学习率	5.3e-4	4.2e-4	3.5e-4

表 2 | DeepSeek - Coder 的超参数。

利用配备NVIDIA A100和H800 GPU的集群。在A100集群中，每个节点配置有8块GPU，并通过NVLink桥连接成对。H800集群同样如此安排，每个节点也包含8块GPU。这些GPU通过NV Link和NVSwitch技术进行互联，确保节点内的高效数据传输。为了在A100和H800集群中实现无缝通信，我们采用InfiniBand互连技术，这种技术以其高吞吐量和低延迟而闻名。这种设置为我们的计算实验提供了坚固且高效的基础设施。

3.6. Long Context

为了增强DeepSeek-Coder在处理扩展上下文方面的能力，特别是在仓库级别代码处理等场景中，我们重新配置了RoPE (Su et al., 2023) 参数以扩展默认的上下文窗口。遵循先前的做法 (Chen et al., 2023 ; kaiokendev, 2023)，我们采用了一种线性缩放策略，将缩放因子从1增加到4，并将基本频率从10000调整到100000。模型额外进行了1000步训练，批大小为512，序列长度为16K。学习率保持与最终预训练阶段相同。理论上，这些修改使我们的模型能够在64K令牌的上下文中进行处理。然而，实证观察表明，该模型在其输出最为可靠的16K令牌范围内表现最佳。未来的研究将继续完善和评估长上下文适应方法，旨在进一步提高DeepSeek-Coder在处理扩展上下文方面的效率和用户友好性。

3.7. Instruction Tuning

我们通过基于指令的微调方法，使用高质量数据来增强DeepSeek-Coder-Base，从而开发出DeepSeek-Coder-Instruct。这些数据包括经过Alpaca指令格式 (Taori等，2023) 结构化的人类指令，以区分每一部分。

<|EOT|>

对话转向，我们使用了一个唯一的分隔符令牌 来表示每个段落的结束。在训练中，我们使用余弦调度策略，包含100步预热阶段，并且初始学习率为1e-5。我们还使用了4000万个令牌的批量大小和总共20亿个令牌。

一个使用DeepSeek-Coder-Instruct 34B的例子展示在图4中。这个示例是一个多轮对话场景，用于构建蛇游戏。首先，我们要求模型使用pygame编写一个蛇游戏。模型成功创建了一个可以正常运行且没有错误的基本蛇游戏。为了改进游戏，我们进一步请求在左上角添加一个得分系统。随后，模型引入了 "分数" 变量和一个 "display _ score" 功能，以及如何集成这些功能的说明。本例展示了DeepSeek-Coder-Instruct在多轮对话场景中提供完整解决方案的能力。更多案例详见附录A。



图 4 | DeepSeek - Coder - Instruct 33B 在多轮设置中的响应示例。

4. 实验结果

在本节中，我们评估了DeepSeek-Coder在四项任务上的表现，包括代码生成（第4.1节）、FIM代码完成（第4.2节）、跨文件代码完成（第4.3节）以及程序基础的数学推理（第4.4节）。我们将DeepSeek-Coder与之前的最先进的大型语言模型进行了比较：

- **CodeGeeX2** 郑等（2023）代表了多语言代码生成模型CodeGeeX的第二代。该模型基于ChatGLM2（杜等，2022）架构开发，并通过大量编程示例数据集进行了增强。
- **StarCoder** (Li等, 2023) 是一个可公开访问的模型，参数量达到150亿。该模型专门训练于精心筛选的Stack数据子集（Kocetkov等, 2022），涵盖了86种编程语言，确保其在广泛的编码任务中具备卓越的能力。
- **CodeLlama** 罗济耶等（2023）涵盖了系列以代码为中心的大语言模型（LLMs），这些模型是基于LLaMA2（托弗伦等，2023）衍生而来的。这些模型有三种规模——7B、13B和34B，并且持续在包含5000亿个token的庞大代码语料库上进行训练，建立在基础的LLaMA2架构之上。
- **代码 - 库斯曼 - 001** Chen 等人(2021)是由 OpenAI 开发的 120 亿个参数模型，并作为 Github Copilot 的初始模型。
- **GPT - 3.5 和 GPT - 4** (OpenAI, 2023) 是由 OpenAI 开发的先进生成式人工智能模型。虽然它们并非专门针对代码生成进行训练，但也展示了

在该领域表现出色。它们在处理代码生成任务方面的有效性主要归因于其参数量的巨大规模。

4.1. 代码生成

HumanEval 和 MBPP 基准 HumanEval (Chen等，2021) 和MBPP (Austin等，2021) 基准广泛用于评估代码LLM。HumanEval包含164个手工编写的Python问题，通过测试案例验证，以零样本设置评估代码生成能力；而MBPP基准则包括500个问题，采用少量样本设置。为了评估模型的多语言能力，我们将Humaneval基准中的Python问题扩展到七种常用编程语言，即C++、Java、PHP、TypeScript (TS)、C#、Bash和JavaScript (JS) (Cassano等，2023)。对于这两个基准，我们采用了贪婪搜索方法，并使用相同的脚本和环境重新实现了基线结果，以确保公平比较。

型号尺寸	Python	C++	Java	PHP	TS		C#	Bash	JS	Avg	MBPP
多语言基础模型											
代码 - 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%	36.2%
StarCoderBase 16B	31.7%	31.1%	28.5%	25.4%	34.0%	34.8%	8.9%	29.8%	28.0%	42.8%	42.8%
CodeLlama 7B	31.7%	29.8%	34.2%	23.6%	36.5%	36.7%	12.0%	29.2%	29.2%	38.6%	38.6%
CodeLlama 13B	36.0%	37.9%	38.0%	34.2%	45.2%	43.0%	16.5%	32.3%	35.4%	48.4%	48.4%
CodeLlama 34B	48.2%	44.7%	44.9%	41.0%	42.1%	48.7%	15.8%	42.2%	41.0%	55.2%	55.2%
DeepSeek - Coder - Base 1.3 B	34.8%	31.1%	32.3%	24.2%	28.9%	36.7%	10.1%	28.6%	28.3%	46.2%	46.2%
DeepSeek - Coder - Base 6.7 B	49.4%	50.3%	43.0%	38.5%	49.7%	50.0%	28.5%	48.4%	44.7%	60.6%	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%	66.0%
指令调整模型											
GPT - 3.5 - Turbo -	76.2%	63.4%	69.2%	60.9%	69.1%	70.8%	42.4%	67.1%	64.9%	70.8%	70.8%
GPT-4 -	84.1%	76.4%	81.6%	77.2%	77.4%	79.1%	58.2%	78.0%	76.5%	80.0%	80.0%
DeepSeek - Coder - Instruct 1.3 B	65.2%	45.3%	51.9%	45.3%	59.7%	55.1%	12.7%	52.2%	48.4%	49.4%	49.4%
DeepSeek - Coder - Instruct 6.7 B	78.6%	63.4%	68.4%	68.9%	67.2%	72.8%	36.7%	72.7%	66.1%	65.4%	65.4%
DeepSeek - Coder - Instruct 33B	79.3%	68.9%	73.4%	72.7%	67.9%	74.1%	43.0%	73.9%	69.2%	70.0%	70.0%

表 3 | 多语言 HumanEval 和 MBPP 基准方法的性能。

如表3所示，本研究的结果表明，DeepSeek-Coder-Base 在 HumanEval 和 MBPP 上分别取得了 50.3% 和 66.0% 的平均准确率，达到了最先进的性能。与同样规模的开源模型 CodeLlama-Base 34B 相比，我们的模型在准确率方面分别提高了 9% 和 11%。值得注意的是，即使是我们较小的模型 DeepSeek-Coder-Base 6.7B 也超过了 CodeLlama-Base 34B 的性能。经过指令微调后，我们的模型在 HumanEval 基准测试中超越了闭源的 GPT-3.5-Turbo 模型，显著缩小了与 OpenAI GPT-4 之间的性能差距。

DS - 1000 基准 HumanEval 和 MBPP 存在一个显著的缺点，即它们高度依赖于简单的编程任务，这可能无法准确反映大多数程序员通常编写的代码类型。相比之下，Lai 等人 (2023) 在工作中引入的 DS-1000 基准测试提供了涵盖七种不同库的 1,000 个实用且现实的数据科学工作流程的全面集合。该基准测试通过使用特定测试案例执行代码来评估代码生成能力。DS-1000 的独特之处在于其基于所涉及库对问题进行分类，这些库包括 Matplotlib、NumPy、Pandas、SciPy、Scikit-

学习、PyTorch和TensorFlow。基准评估了代码补全设置下基础模型的性能，并为每个库提供了pass@1结果以及总体得分。

DS-1000基准的结果如表4所示。从表中可以看出，DeepSeek-Coder模型在所有库中都实现了较高的准确性，这表明我们的模型不仅能够生成高质量的代码，还能够在实际的数据科学工作流程中更准确地使用库。

型号尺寸	Matplotlib	Numpy	熊猫	Pytorch	:	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 1.3 B	32.3%	21.4%	9.3%	8.8%	8.5%	16.5%	8.9%	16.2%
DeepSeek - Coder - Base 6.7 B	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%

表 4 | DS - 1000 任务上不同方法的性能。

LeetCode 竞赛基准 为了进一步验证该模型在实际编程问题中的能力，我们构建了LeetCode Contest基准。³ LeetCode⁴ 呈现了竞赛级别的问题，提供了显著的挑战，测试模型的问题理解和代码生成能力。我们从LeetCode Contest收集了最新的问题，以防止这些问题或其解决方案出现在预训练数据中。从2023年7月到2024年1月共收集了180个问题。对于每个问题，我们收集了100个测试案例以确保测试覆盖范围。我们使用模板“{问题描述}\n请在此完成代码以解决上述问题：\n```\npython\n{代码模板}\n```”构建指令提示。

评价结果如表5所示。在我们的评估中，DeepSeek-Coder模型在当前开源编码模型中表现出色。具体而言，DeepSeek-Coder-Instruct 6.7B和33B分别在这个基准测试中实现了19.4%和27.8%的Pass@1得分。这一性能显著超过了现有的开源模型，如Code-Llama-33B。DeepSeek-Coder-Instruct 33B是唯一一个在这项任务中优于OpenAI的GPT-3.5-Turbo的开源模型。然而，与更先进的GPT-4-Turbo相比，仍存在明显的性能差距。

我们的分析表明，实施Chain-of-Thought (CoT) 提示显著提升了DeepSeek-Coder-Instruct模型的能力。这一改进在更具挑战性的任务子集中尤为明显。通过在初始提示后添加指令：“首先编写一个逐步大纲，然后编写代码。”我们观察到性能有所提升。这一观察使我们相信，先制定详细的代码描述过程有助于模型更有效地理解并解决编码任务中的逻辑和依赖关系复杂性，尤其是那些更为复杂的任务。因此，我们在使用DeepSeek-Coder-Instruct模型应对复杂编码挑战时，强烈建议采用CoT提示策略。这种做法促进了更为系统和逻辑的问题解决框架，可能在代码生成任务中产生更加精确和高效的成果。

³ <https://github.com/deepseek-ai/DeepSeek-Coder/tree/main>
我们已经在
/ 评估 / LeetCode

⁴ <https://leetcode.com/>

型号尺寸	Easy (45)	中等 (91)	Hard (44)	总体(180)
WizardCoder - V1.0 15B	17.8%	1.1%	0.0%	5.0%
CodeLlama - Instruct 34B	24.4%	4.4%	4.5%	9.4%
Phind - CodeLlama - V2 34B	26.7%	8.8%	9.1%	13.3%
GPT - 3.5 - Turbo -	46.7%	15.4 %	15.9%	23.3%
GPT - 3.5 - Turbo + CoT -	42.2%	15.4%	20.5%	23.3%
GPT - 4 - Turbo -	73.3%	31.9%	25.0%	40.6%
GPT - 4 - Turbo + CoT -	71.1%	35.2%	25.0%	41.8%
DeepSeek - Coder - Instruct 1.3 B	22.2%	1.1%	4.5%	7.2%
DeepSeek - Coder - Instruct + CoT 1.3 B	22.2%	2.2%	2.3%	7.2%
DeepSeek - Coder - Instruct 6.7 B	44.4%	12.1%	9.1%	19.4%
DeepSeek - Coder - Instruct + CoT 6.7 B	44.4%	17.6%	4.5%	21.1%
DeepSeek - Coder - Instruct 33B	57.8%	22.0%	9.1%	27.8%
DeepSeek - Coder - Instruct + CoT 33B	53.3%	25.3%	11.4%	28.9%

表 5 | 不同模型在 LeetCode 竞赛基准上的表现。

尽管我们投入了大量努力以收集最新的代码问题用于模型测试，但仍无法完全排除数据污染的可能性。我们在7月和8月举办的LeetCode竞赛中观察到，GPT-4-Turbo和DeepSeek-Coder模型获得了更高的分数。我们建议研究社区在将来使用我们发布的LeetCode数据进行模型评估时考虑数据污染的潜在问题。

4.2. 填写中间代码完成

DeepSeek-Coder 模型在其预训练阶段使用 0.5 的 FIM (Fill-In-the-Middle) 率进行训练。这种专门的训练策略使模型能够根据给定代码片段前后文（前缀和后缀）填补空白，从而高效地生成代码。这一能力特别适用于代码补全工具。许多具有类似能力的开源模型已经涌现出来。其中值得一提的是 SantaCoder (Allal 等人, 2023 年)、StarCoder (Li 等人, 2023 年) 和 CodeLlama (Roziere 等人, 2023 年)。这些模型在代码生成和补全领域树立了先例。在评估 DeepSeek-Coder 模型的表现时，我们与上述模型进行了对比分析。此次比较的基准是单行填充基准，包括由 Allal 等人 (2023 年) 提出的三种不同编程语言的数据集。该基准使用行准确匹配精度作为评估指标。

型号尺寸	python	java	JavaScript	Mean
SantaCoder 1.1 B	44.0%	62.0%	74.0%	69.0%
StarCoder 16B	62.0%	73.0%	74.0%	69.7%
CodeLlama - Base 7B	67.6%	74.3%	80.2%	69.7%
CodeLlama - Base 13B	68.3%	77.6%	80.7%	75.5%
DeepSeek - Coder - Base 1B	57.4%	82.2%	71.7%	70.4%
DeepSeek - Coder - Base 7B	66.6%	88.1%	79.7%	80.7%
DeepSeek - Coder - Base 33B	65.4%	86.6%	82.5%	81.2%

表 6 | FIM 任务上不同方法的性能。

评价结果如表 6 所示。尽管是容量最小的模型

.of 1.3亿个参数，DeepSeek-Coder 在这些基准测试中表现优于其更大的同类模型StarCoder和CodeLlama。这种卓越性能可归因于DeepSeek-Coder所使用的高质量预训练数据。此外，观察到的一个显著趋势是模型大小与性能之间的关联性。随着模型规模的增加，相应的性能提升也变得更加合理。这一趋势强调了模型容量在代码补全任务中实现更高准确性的关键作用。基于这些发现，我们建议在代码补全工具中部署DeepSeek-Coder-Base 6.7B模型。这一建议是建立在该模型在效率与准确性之间取得平衡的基础上的。DeepSeek-Coder-Base 6.7B模型凭借其庞大的参数量，在代码补全领域已证明其高度有效性，使其成为将先进计算能力整合到编程环境中的理想选择。

4.3. 跨文件代码完成

在本节中，我们将评估现有开源模型在跨文件代码完成任务中的性能。与上一节讨论的代码生成不同，跨文件代码完成要求模型访问和理解跨越多个文件且具有众多跨文件依赖关系的仓库。我们使用CrossCodeEval (Ding等，2023) 来评估目前可用的大规模 (7B) 开源代码模型在跨文件完成任务中的能力。该数据集基于四个流行编程语言 (Python、Java、TypeScript和C#) 的一组多样化的实际开源且许可宽松的仓库构建而成。该数据集特别设计以严格要求跨文件上下文，以确保准确完成。值得注意的是，该数据集是从2023年3月至6月创建的仓库构建而来，而我们的预训练数据仅包括在2023年2月之前创建的代码，从而确保该数据集未包含在我们的预训练数据中，避免了数据泄露。

型号尺寸	Python Java TypeScript C#									
	EM	ES	EM	ES	EM	ES	EM	ES	EM	ES
CodeGeex2 6B	8.11%	59.55%	7.34%	59.60%	6.14%	55.50%	1.70%	51.66%		
+ 检索	10.73%	61.76%	10.10%	59.56%	7.72%	55.17%	4.64%	52.30%		
StarCoder - Base 7B	6.68%	59.55%	8.65%	62.57%	5.01%	48.83%	4.75%	59.53%		
+ 检索	13.06%	64.24%	15.61%	64.78%	7.54%	42.06%	14.20%	65.03%		
CodeLlama - Base 7B	7.32%	59.66%	9.68%	62.64%	8.19%	58.50%	4.07%	59.19%		
+ 检索	13.02%	64.30%	16.41%			64.64%	12.34%	60.64%	13.19%	63.04%
DeepSeek - Coder - Base 6.7 B	9.53%	61.65%	10.80%	61.77%	9.59%	60.17%	5.26%	61.32%		
+ 检索		16.14%	66.51%	17.72%	63.18%	14.03%	61.77%	16.23%	63.42%	
+ retrieval w / o Repo Pre - training	16.02%		66.65%	16.64%	61.88%	13.23%	60.92%	14.48%	62.38%	

表 7 | 不同模型对跨文件代码完成的性能。

在我们对各种模型的评估中，我们将最大序列长度设置为2048个令牌，最大输出长度设置为50个令牌，并将跨文件上下文的最大令牌数限制为512个。对于跨文件上下文，我们利用了Ding等 (2023) 提供的官方BM25搜索结果。评价指标包括精确匹配和编辑相似度。结果如表7所示，表明DeepSeek-Coder在多种语言的跨文件完成任务中始终优于其他模型，展示了其优越的实际应用能力。仅使用文件级代码语料库时 (你：)， w / o 回购预培训 为了预训练DeepSeek-Coder，在Java、TypeScript和C#语言中观察到性能下降，这表明仓库级别预训练的有效性。

4.4. 基于程序的数学推理

程序导向的数学推理涉及评估模型通过编程理解并解决数学问题的能力。这种类型的推理在数据解析和科学计算等领域至关重要。为了进行这项评估，我们采用了高晓等 (2023) 提出的程序辅助数学推理 (PAL) 方法。该方法应用于七个不同的基准测试中，每个基准测试都提供了独特的挑战和背景。这些基准包括GSM8K (Cobbe等，2021) 、 MATH (Hendrycks等，2021) 、 GSM-Hard (高晓等，2023) 、 SVAMP (Patel等，2021) 、 TabMWP (Lu等，2022) 、 ASDiv (Miao等，2020) 和MAWPS (Gou等，2023) 。在每个基准测试中，模型被要求交替地用自然语言描述解决方案步骤，然后用代码执行该步骤。如表8所示，DeepSeek-Coder模型在所有基准测试中表现出色，尤其是33B变体，这表明使用此类模型在需要复杂数学计算和解决问题能力的应用中具有巨大潜力。

型号尺寸	GSM8k	MATH	GSM - Hard	SVAMP	TabMWP	ASDiv	MAWPS	Avg
多语言基础模型								
CodeGeex - 2 7B	22.2%	9.7%	23.6%	39.0%	44.6%	48.5%	66.0%	36.2%
StarCoder - Base 16B	23.4%	10.3%	23.0%	42.4%	45.0%	54.9%	81.1%	40.0%
CodeLlama - Base 7B	31.2%	12.1%	30.2%	54.2%	52.9%	59.6%	82.6%	46.1%
CodeLlama - Base 13B	43.1%	14.4%	40.2%	59.2%	60.3%	63.6%	85.3%	52.3%
CodeLlama - Base 34B	58.2%	21.2%	51.8%	70.3%	69.8%	70.7%	91.8%	62.0%
DeepSeek - Coder - Base 1.3 B	14.6%	16.8%	14.5%	36.7%	30.0%	48.2%	62.3%	31.9%
DeepSeek - Coder - Base 6.7 B	43.2%	19.2%	40.3%	58.4%	67.9%	67.2%	87.0%	54.7%
DeepSeek - Coder - Base 33B	60.7%	29.1%	54.1%	71.6%	75.3%	76.7%	93.3%	65.8%

表 8 | 不同方法在程序辅助数学推理任务上的性能。

5. 继续从一般 LLM 的预培训

为了进一步增强DeepSeek-Coder模型在自然语言理解和数学推理方面的能力，我们从通用语言模型DeepSeek-LLM-7B Base (DeepSeek-AI, 2024) 中进行额外的预训练，使用了2万亿个令牌的数据，从而生成了DeepSeek-Coder-v1.5 7B。对于此次预训练，我们特别使用了表9中列出的数据源。与DeepSeek-Coder不同，DeepSeek-Coder-v1.5 在预训练阶段仅使用下一个令牌预测目标，并且上下文长度为4K。

数据源	百分比
源代码	70%
Markdown 和 StackExchange	10%
与代码相关的自然语言	7%
与数学相关的自然语言	7%
双语(汉英) 自然语言	6%

表 9 | DeepSeek - Coder - v1.5 7B 预训练的数据源

我们对DeepSeek-Coder-v1.5 7B和DeepSeek-Coder 6.7B进行了比较，并使用我们的评估管道重新运行所有基准测试以确保公平比较。我们从以下几类任务评估性能范围：

- 1. 代码生成任务；
- 2. 代码理解任务；
- 3. 代码修复任务；
- 4. 其他相关任务。

• **编程**：本类别包括使用Chen等人（2021）的HumanEval数据集在多语言环境下进行的评估，以及使用Austin等人（2021）的MBPP数据集在Python环境下进行的评估。 **数学推理** 我们使用GSM8K基准（Cobbe等，2021）和MATH基准（Hendrycks等，2021）[4]来评估在数学推理任务上的性能。这些任务涉及通过生成程序来解决数学问题。 **自然语言** 我们的自然语言任务评估包括MMLU（Hendrycks等，2020）、BBH（Suzgun等，2022）、HellaSwag（Zellers等，2019）、Winogrande（Sakaguchi等，2021）和ARC-Challenge（Clark等，2018）基准。

基模型和指令模型的结果如表10所示。观察发现，尽管DeepSeek-Coder-Base-v1.5模型在编码性能上略有下降，但在大多数任务上相较于DeepSeek-Coder-Base模型表现出了明显的改进。特别是在数学推理和自然语言类别中，DeepSeek-Coder-Base-v1.5在所有基准测试中显著优于其前身，这也展示了该模型在数学推理和自然语言处理能力上的显著提升。

型号尺寸	自然语言编程数学推理									
	HumanEval	MBPP	GSM8K	MATH	MMLU	BBH	HellaSwag	WinoG	ARC - C	
DeepSeek - Coder - Base 6.7 B			44.7%	60.6%	43.2%	19.2%	36.6%	44.3%	53.8%	57.1%
DeepSeek - Coder - Base - v1.5 6.9 B	43.2%	60.4%		62.4%	24.7%	49.1%	55.2%	69.9%	63.8%	47.2%
DeepSeek - Coder - Instruct 6.7 B			66.1%	65.4%	62.8%	28.6%	37.2%	46.9%	55.0%	57.6%
DeepSeek - Coder - Instruct - v1.5 6.9 B	64.1%	64.6%		72.6%	34.1%	49.5%	53.3%	72.2%	63.4%	48.1%

表10 | DeepSeek-Coder-Base与DeepSeek-Coder-Base-v1.5性能比较分析。数学任务通过编程解决。

6. Conclusion

在本技术报告中，我们介绍了名为DeepSeek-Coder的一系列专门用于编程的大型语言模型（LLMs），可供选择的规模有1.3B、6.7B和33B参数。这些模型的独特之处在于它们基于精心筛选的项目级代码语料库进行训练，并采用“填补空白”的预训练目标来增强代码填充能力。一个重要进展是将模型的上下文窗口扩展到16,384个标记，从而显著提高了其在处理大量代码生成任务方面的有效性。我们的评估结果显示，我们系列中最先进的模型DeepSeek-Coder-Base 33B在多种标准测试中超越了现有的开源代码模型。令人印象深刻的是，尽管DeepSeek-Coder-Base 6.7B模型的规模较小，但其性能与具有34B参数的CodeLlama相当，这证明了我们预训练语料库的高质量。

为了增强DeepSeek-Coder-Base模型的零样本指令能力，我们使用高质量的教学数据对它们进行了微调。这导致DeepSeek-Coder-Instruct 33B模型在一系列与编程相关任务中表现优于OpenAI的GPT-3.5 Turbo，展示了其在代码生成和理解方面的卓越能力。

为了进一步提高DeepSeek-Coder-Base模型的自然语言理解能力，我们基于DeepSeek-LLM 7B检查点进行了额外的预训练。此次额外训练处理了一个包含20亿个标记的多样数据集，其中包括自然语言、代码和数学数据。结果是创建了一个新的

并且改进了代码模型，DeepSeek-Coder-v1.5。我们的观察表明，DeepSeek-Coder-v1.5 不仅维持了其前身在高级编程方面的高水平性能，还展示了增强的人类自然语言理解能力。这一进步强化了我们相信最有效的以代码为中心的大规模语言模型（LLMs）应基于强大的通用LLMs构建的观点。原因显而易见：要有效地解析和执行编程任务，这些模型必须具备对人类指令的深刻理解，而这些指令往往以各种形式的自然语言出现。展望未来，我们的承诺是基于更大规模的通用LLMs开发并公开分享更加强大的以代码为中心的LLMs。

Acknowledgements

我们衷心感谢刘波、邓 Chengqi、阮冲、戴大迈、李嘉思、管康、张明川、黄盼盼、于疏萍、马世荣、孙耀峰、乔诗怡、邵志红和郝泽文在训练DeepSeek-Coder模型期间提供的宝贵讨论和协助。

参考文献

L. B. Allal, R. Li, D. Kocetkov, C. Mou, C. Akiki, C. M. Ferrandis, N. Muennighoff, M. Mishra, A. Gu, M. Dey, 等. Santacoder: 不要伸手摘星！arXiv 预印本 [arXiv:2301.03988](#), 2023.

J. 奥斯汀, A. 奥德纳, M. 尼尔, M. 波斯玛, H. 米卡莱夫斯基, D. 约翰逊, E. 江, C. 蔡, M. 特里, Q. 乐, 和 C. 斯图尔特. 使用大型语言模型进行程序合成, 2021.

M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen. 通过有效训练语言模型来填补中间环节. arXiv预印本 [arXiv:2207.14255](#), 2022.

F. 柯萨诺, J. 戈沃, D. 阮, S. 阮, L. 菲普斯-科斯丁, D. 彭肯尼, M.-H. 叶, Y. 子, C. J. 安德森, M. Q. 菲尔德曼, 等. Multipl-e：一种可扩展且多语言的神经代码生成基准测试方法. IEEE Transactions on Software Engineering, 2023.

陈敏, 特雷克 J., jun H., 袁晴, 德·奥利维拉 Pinto H., 卡普兰 J., 爱德华兹 H., 布尔达 Y., 约瑟夫 N., 布罗克曼 G. 等. 评估训练于代码上的大规模语言模型. arXiv 预印本 [arXiv:2107.03374](#), 2021.

宋晨, 王思远, 陈亮, 和田宇. 通过位置内插法扩展大型语言模型的上下文窗口. arXiv 预印本 [arXiv:2306.15595](#), 2023.

P. 克拉克, I. 科怀, O. 埃齐奥尼, T. 科特, A. 萨布哈瓦尔, C. 梅尼克, 和 O. 塔夫乔德. 自认为已经解决了问答问题吗？尝试 ARC (AI2 原理挑战) 吧. arXiv 预印本 [arXiv:1803.05457](#), 2018.

柯布 K., 科萨鲁贾 V., 巴维安 M., 陈 M., 基恩 H., 基亚尔 L., 普拉普特 M., 特雷克 J., 海顿 J., 钵野 R., 等. 训练验证者解决数学文字问题. arXiv 预印本 [arXiv:2110.14168](#), 2021.

T. 道. Flashattention - 2：更快的注意力，更好的并行性和工作分区，2023 年。

DeepSeek - AI。Deepseek llm：使用 longtermise 扩展开源语言模型。arXiv 预印本 [arXiv:2401.02954](#)，2024。

丁辉, 谭猛, 詹纳·辛, 拉曼纳坦·M·K, 纳拉帕蒂·瑞, 比塔亚·P, محمد丁宇, 王喆, 乌尔夫·阿, 罗思·D, 等. Crosscodeeval: 一个多样性和多语言的跨文件代码完成基准. 在第三十七届神经信息处理系统会议数据集和基准-track, 2023.

杜志强, 钱志强, 刘 X, 丁志强, 邱志强, 杨志强, 唐志强. Glm: 一般语言模型预训练与自回归空白填充. 在第 60 届年会的会议记录中
计算语言学协会 (第 1 卷: 长篇论文), 第 320 - 335 页, 2022.

D. 梅德, A. 阿加汉yan, J. 林, S. 王, E. 威廉斯, F. 石, R. 仲, W.-t. 伊, L. 茨特勒莫耶尔, 和 M. 李. Incoder: 一种代码填充与合成生成模型. arXiv 预印本 arXiv:2204.05999, 2022.

高亮, 马达安, 周世, 阿隆, 刘鹏, 杨宇, 卡拉丁, 和 内布伊格. Pal: 基于程序的语言模型. 在国际机器学习会议上的论文, 页码: 10764–10799. PMLR, 2023.

https:

G. 双子座团队. 双子座: 一个功能强大的多式联运模型家族, 2023 年. URL

// goo.gle / GeminiPaper

戈 Z., 邵 Z., 郭 Y., 杨 Y., 黄 M., 段 N., 陈 W., 等. Tora: 一个用于数学问题求解的工具集成推理代理. arXiv 预印本 arXiv:2309.17452, 2023.

D. 森德里克斯, C. 布恩斯, S. 巴萨特, A. 周, M. 毛伊卡, D. 宋, 和 J. 斯坦哈特. 测量大规模多任务语言理解. 在国际学习表示会议 (ICLR) 上发表, 2020.

D. 汉德里克斯, C. 布恩斯, S. 卡达瓦斯, A. 阿罗拉, S. 基萨特, E. 坂田, D. 宫松, 和 J. 斯坦哈特. 使用 Math 数据集衡量数学问题解决能力. arXiv 预印本 arXiv:2103.03874, 2021.

High - Flyer。Hai - llm: 用于训练大型模型的高效轻量级工具。2023。URL

https: // www. high - flyer. cn / en / blog / hai - llm

kaiokendev。我在训练中学习的東西超熱。
/ til # extending - context - to - 8k , 2023.

https: // kaiokendev. github. io

D. Kocetkov, R. Li, L. Jia, C. Mou, Y. Jernite, M. Mitchell, C. M. Ferrandis, S. Hughes, T. Wolf, D. Bahdanau, et al. The stack: 3 tb of permissively licensed source code. Transactions on 机器学习研究, 2022 年。

V. A. 科里希坎蒂, J. 卡斯珀, S. 刘姆, L. 麦凯, M. 安德施, M. 萨伊比和 B. 卡坦扎罗. 大型变压器模型中减少激活重新计算. 机器学习与系统会议论文集, 5, 2023.

赖宇、李超、王宇、张涛、钟瑞、Zettlemoyer 劳伦斯·泽特勒迈耶、Yih 吴文淘、Fried 弗里德·戴维、王杉、于涛. DS-1000: 数据科学代码生成的自然可靠基准. 在国际机器学习会议上的论文, 页码 18319–18345. PMLR, 2023.

K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison - Burch 和 N. Carlini。去复制 - ing 训练数据使语言模型更好. 在第 60 届年会上
计算语言学协会 (第 1 卷: 长篇论文), 第 8424 - 8445 页, 2022.

刘 R. 李, L. B. 阿拉尔, 张一帆, 穆恩尼霍夫 N., 科切克夫 D., 莫尤 C., 马龙 M., 阿基基 C., 李杰, 吉姆 J. 等人. Starcoder: 让源代码与您同在! arXiv预印本 arXiv:2305.06161, 2023.

I. Loshchilov 和 F. Hutter。解耦权重衰减正则化, 2019 年。

卢鹏, 邱亮, 张克威, 吴远宁, 朱思诚, 拉贾普罗希特·塔库尔, 克拉克·彼得, 卡利扬·阿肖卡。基于策略梯度的动态提示学习在半结构化数学推理中的应用. 第十一届国际学习表示会议, 2022年。

苗思远, 梁楚楚, 和 孙凯英. 用于评估和开发英语数学应用题求解器的多样化语料库. 在第58届计算语言学协会年度会议记录, 页码: 975–984, 2020.

戴利·纳拉扬、阿米尔·哈拉普、阿尼尔·费尼亚西、维杰·塞沙德里、尼尔·R·达万、格雷戈里·R·甘杰、佩里·B·吉布斯和迈克尔·扎利亚. Pipedream: 深度神经网络训练的一般化流水线并行性. 在第27届ACM操作系统原理会议论文集, 页码1-15, 2019年。

E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, 和 C. Xiong. Codegen: 一种用于代码的开放大型语言模型及多轮程序合成. arXiv 预印本 arXiv:2203.13474, 2022.

E. Nijkamp, H. Hayashi, C. Xiong, S. Savarese, and Y. Zhou. Codegen2: 关于在编程和自然语言上训练大规模语言模型的经验教训, 2023。

OpenAI. Gpt - 4 技术报告, 2023 年。

A. 法尔特, S. 萨蒂姆拉和 N. 吉瓦尔. 真正能够解决简单的数学文字题吗? In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2080–2094, 2021.

拉弗尔 C, 沙希尔 N, 罗伯茨 A, 李 K, 纳朗 S, 马特纳 M, 周 Y, 李 W, 以及刘 P. J. 探索统一的文本到文本变换器在转移学习极限上的应用, 2023。

萨利布·拉詹巴德里、杰伊·拉斯基、奥卢塞贡·鲁瓦塞和何一。零：面向训练万亿参数模型的内存优化。在SC20：国际高性能计算、网络、存储与分析大会, 页码1-16。IEEE, 2020。

罗西耶 B, 格林希 J, 格洛克勒 F, 苏特拉 S, 加特 I, 谭 X.E, 阿迪 Y, 刘 J, 雷梅兹 T, 拉宾 J 等. 代码羊驼：开放的基础模型代码. arXiv 预印本 arXiv:2308.12950, 2023.

sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: 大规模 adversarial winograd schema挑战. 《通信》(通讯), 64(9):99–106, 2021.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword unit. arXiv preprint arXiv: 1508.07909, 2015.

J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen 和 Y. Liu。Roformer: 具有旋转位置嵌入的增强型变压器, 2023 年。

苏朱恩, 萨卡尔斯, 施尔利, 吉尔曼, 泰耀, 李Quant, 希尼·奇, 查乌德里, 雷量子, 奇, 和 魏. 具有挑战性的基准任务及链式思考能否解决它们. arXiv预印本arXiv:2210.09261, 2022.

R. Taori , I. Gulrajani , T. Zhang , Y. Dubois , X. Li , C. Guestrin , P. Liang 和 T. B. 桥本。

<https://github.com/tatsu-lab>

斯坦福羊驼：一种遵循指令的美洲驼模型。

[/stanford_羊驼](#)

, 2023. H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. B. atra, P. Bhargava, S. Bhosale, 等人. Llama 2：开放基础模型及微调聊天模型. arXiv预印本 arXiv:2307.09288, 2023. Y. Wang, W. Wang, S. Joty, 和 S. C. Hoi. Codet5：识别意识统一的预训练编码器-解码器模型，用于代码理解和生成. arXiv预印本 arXiv:2109.00859, 2021. R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, 和 Y. Choi. Hellaswag：机器真的能完成你的句子吗？在第57届计算语言学协会年度会议记录中，页码：4791–4800, 2019. Q. Zheng, X. Xia, X. Zou, Y. Dong, S. Wang, Y. Xue, L. Shen, Z. Wang, A. Wang, Y. Li, 等人. Codegeex：多语言基准测试下的预训练代码生成模型，用于humaneval-x. 在第29届ACM SIGKDD知识发现与数据挖掘会议记录中，页码：5673–5684, 2023.

A. 用 DeepSeek - Coder - Instruct 聊天的案例

我们将呈现两个与DeepSeek-Coder-Instruct交互的案例，其中一个涉及关于创建数据库和进行数据分析的多轮对话，另一个则集中在使用模型解决来自LeetCode的示例问题上。

在第一种情景中，如图5所示，我们指示模型使用Python构建一个学生数据库，并随机插入10条信息。随后，在第二轮对话中，我们继续通过分析学生年龄分布来进一步询问模型。从图5可以看出，模型能够生成无错误且全面的代码，并附有解释性说明。在第二种情景中，如图6所示，我们进一步评估模型的能力，通过在一个领域外的LeetCode竞赛问题上进行测试。该特定问题于2023年11月发布，不在我们的数据收集范围内，因此也不属于模型的训练数据。结果显示，我们的模型在解决超出其训练分布的问题时表现出色。

Q1：用 Python 构建数据库，包括关于学生的各种信息，插入十条随机记录并显示。

CODE

当然，这里是一个简单的例子，你可以使用 Python 中的 SQLite。

```
"""python
导入 sqlite3
随机导入
# 连接到 SQLite 数据库
conn = sqlite3.connect('students.db')
# 创建一个光标对象
cur = conn.cursor()
...
# 创建表
cur.execute("""
    如果不存在学生，请创建表格 (
    ID INTEGER PRIMARY KEY,
    名称 TEXT,
    年龄 INTEGER,
    等级文本) """)
...
# 显示所有记录
cur.execute(选择 * 来自学生)
...
"""
```

此脚本将创建一个名为 "students.db" 的 SQLite 数据库和一个名为 "学生" 的表，有四列："id"，"name"，"age"，和 "grade"。然后，它将在表中插入 10 条随机记录并显示所有记录。...

运行代码

```
(1, 'Student0', 24, 'B')
(2, 'Student1', 18, 'F')
(3, 'Student2', 21, 'A')
(4, 'Student3', 22, 'F')
(5, 'Student4', 21, 'C')
(6, 'Student5', 19, 'A')
(7, 'Student6', 20, 'B')
(8, 'Student7', 24, 'A')
(9, 'Student8', 21, 'A')
(10, 'Student9', 20, 'B')
```

Q2: 创建描绘年龄分布的图。

CODE

要创建描述学生年龄分布的图表，您可以在 Python 中使用 matplotlib 库。下面是您可以做到的：

```
"""python
导入 sqlite3
进口 pandas 作为 pd
导入 matplotlib.pyplot 作为 plt

# 连接到 SQLite 数据库
conn = sqlite3.connect('students.db')

# 将数据库中的数据读入 pandas DataFrame
df = pd.read_sql_query(选择 * 来自学生, conn)

# 创建年龄分布的直方图
plt.hist(df['age'], bins = range(10, 30, 2), edgecolor = 'black')
plt.title(学生年龄分布)
plt.xlabel(年龄)
plt.ylabel(学生人数)
plt.show()

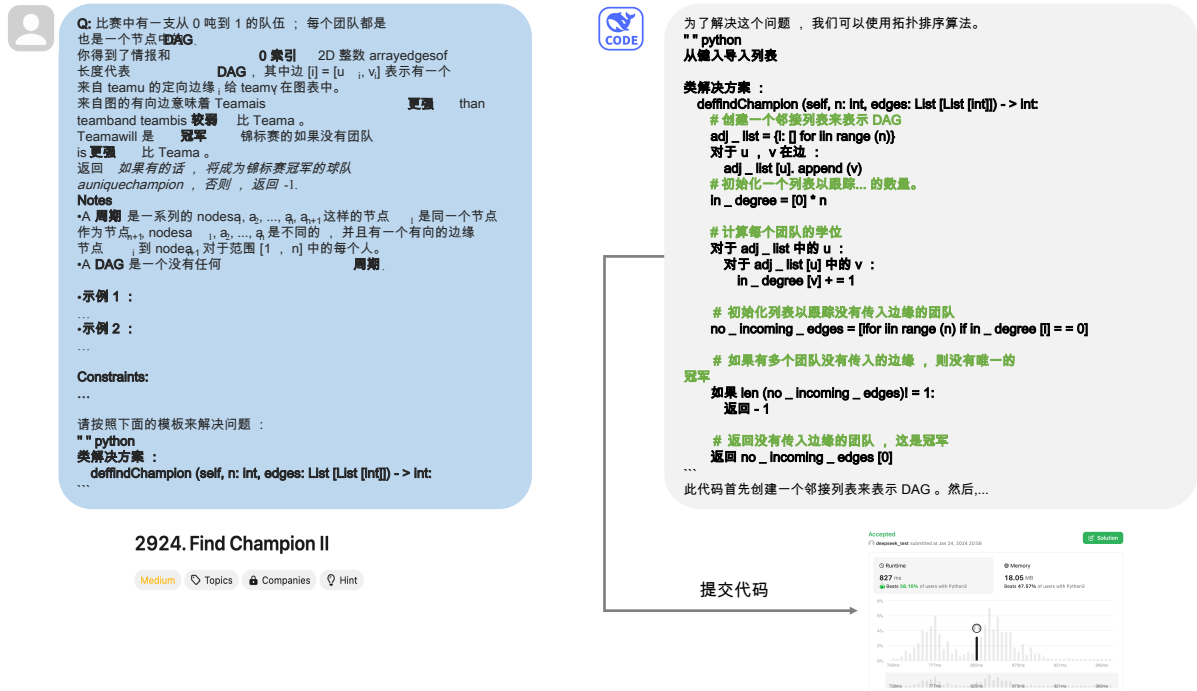
# 关闭连接
conn.close()
"""
```

此脚本将连接到 SQLite 数据库，将数据读入 pandas DataFrame，然后创建年龄分布的直方图。直方图将显示每个年龄段的学生人数。

运行代码

图 5 | 构建数据库和数据分析的示例。

21



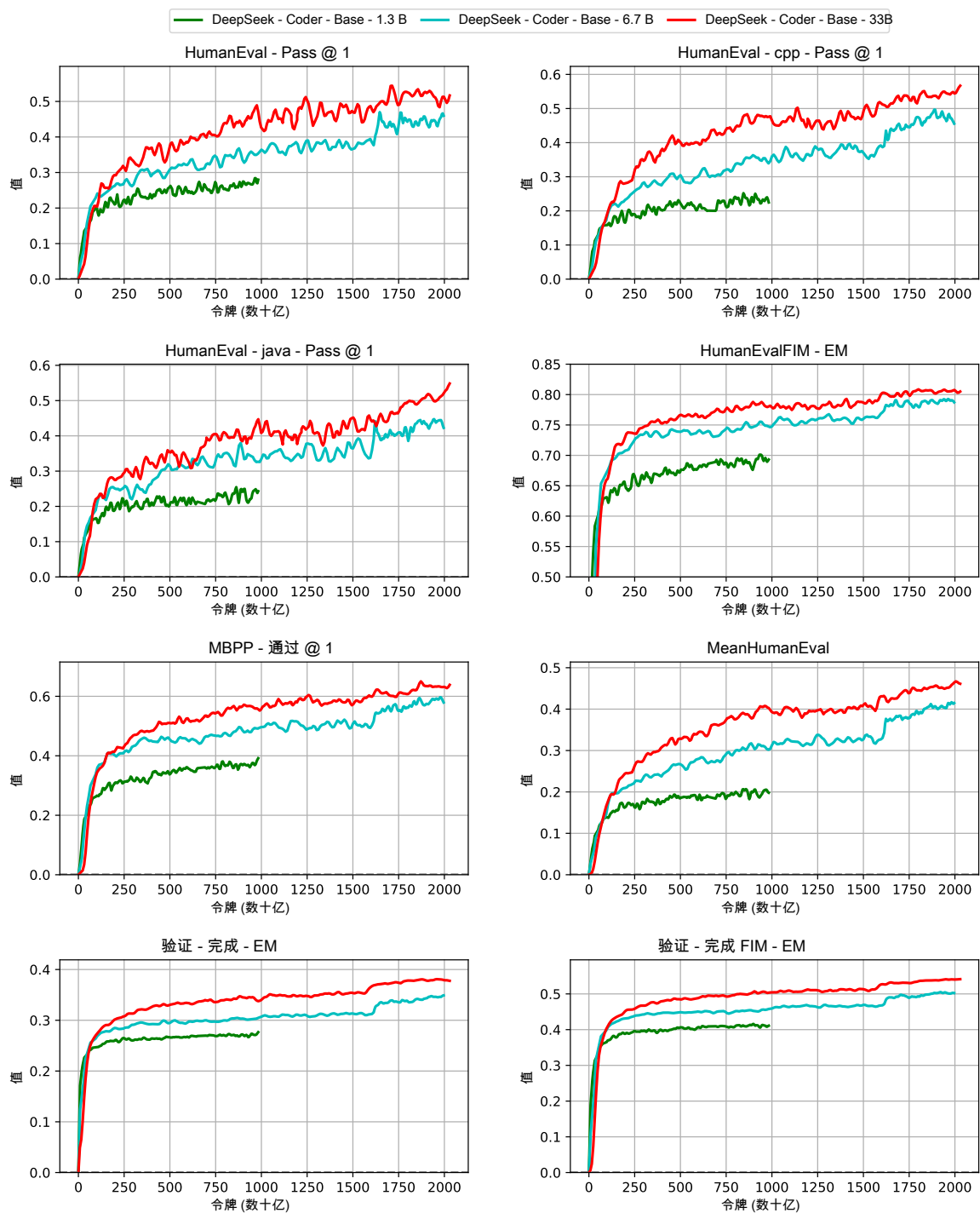


图 7 | DeepSeek - Coder - Base 训练期间的基准曲线。