代码补全方案调研报告

时间：2022-07-08

作者：俊斌

目录：

- 参考链接
- 主流代码补全工具对比表
- TabNine
- GitHub Copilot
- Cosy--阿里巴巴
- AlphaCode--DeepMind
- **Datasets**
- **Reference**
  - Paper
  - GitHub开源项目

参考链接：

1. 知乎--史上最全智能代码补全工具汇总
2. TabNine官网
3. GitHub Copilot
4. paper--Evaluating Large Language Models Trained on Code
5. Why GitHub Copilot Will Not Change Programming

主流的代码补全工具对比表：

Kite、TabNine、GitHub Copilot、IntelliCode、Cosy、AIXcoder.

	TabNine	IntelliCode	Kite	GitHub Copilot	Cosy	AIXcoder
官网	tabnine.com	visualstudio.microsoft.com/zh-hans/services/intellicode	kite.com	copilot.github.com	developer.aliyun.com/tool/cosy	aixcoder.com
支持语言	25种主流编程语言	C#、C++、Java等	Python(专精)、JS等	Java、Python、JS、Ruby、Go等	仅支持Java	Java(优势语言)、Python、JS等
IDE/编辑器	JetBrains、VSCode、Atom等主流IDE	VS Code、Visual Studio	JetBrains、VSCode、Atom等主流IDE	JetBrains、VSCode	IntelliJ IDEA	JetBrains、VSCode
代码补全	支持行级补全	单API补全及排序优化	单API排序，行级补全较少	方法级代码生成、行级补全	行级补全	行级补全

	TabNine	IntelliCode	Kite	GitHub Copilot	Cosy	AIXcoder
IDE内代码搜索	\	\	支持Python API文档搜索	\	Java API代码示例搜索、支持通过功能描述搜索代码	需跳转到网页端
离线模式	支持	支持	支持	\	支持	支持
收费模式	提供免费版、Pro及企业版收费	企业版收费	企业版收费	收费	免费	企业版收费

其中较优秀的为：

- TabNine

  支持的语言及IDE种类最多；
- GitHub Copilot

  提供的补全效果更惊艳；
- Cosy

  代码搜索功能最丰富；

TabNine

官网：https://www.tabnine.com/

Tabnine在其网站上突出显示了一项隐私声明，声明该模型仅在您的机器上本地运行，并且您的数据不会共享。这个设置很酷的一点是，尽管您的数据没有与其他任何人共享，但模型仍在（本地）在你的代码上进行训练，因此随着你的使用，建议会变得更好。

TabNine CEO Dror Weiss 采访专题

主题：如何构建一个AI代码补全工具，将开发人员的工作效率提升一倍以上。

竞争对手，是什么让 Tabnine 与众不同？

- Tabnine 和 Copilot对比推特

人工智能代码辅助市场由两个玩家主导：微软的 Tabnine 和 Copilot。这两种产品的技术方法大不相同。微软依赖于只能由微软托管的单一巨大的单体 AI 模型。我们喜欢较小的原生代码 AI 模型所带来的灵活性和敏捷性，每个模型都是在特定语言或领域从头开始训练的。

我们目前有十几种此类模型可用于所有流行语言，还有由生态系统合作伙伴培训的社区模型。这使客户能够灵活地在我们的云或他们的网络上运行 Tabnine，并能够训练自定义 AI 模型以捕获其存储库中的特定模式。Tabnine 提供关于每次击键的建议以及完整的线路或功能建议，而 Copilot 仅限于仅在新线路上提供建议，因为推理成本和延迟要高得多。

TabNine正在解决哪些最有趣的问题？

过去，Tabnine 给出了小而频繁的建议和 Copilot 大块。Tabnine 现在两者兼而有之，而且是完全独一无二的。

有很大的架构差异。Copilot 依赖于 Codex，他们相信一种模式可以统治一切。我们将代码完成/产品与模型分离。您可以使用任何兼容的模型运行 Tabnine，而我们的模型更小、更灵活。我们从头开始构建所有这些。它不仅仅是一种模型，而是针对不同情况的多种模型。您可以使用私有代码训练自己的模型。自己连 Codex 都无法运行，所以 Copilot 无法在本地运行。但是，使用我们新的更轻的模型，您可以在您的计算机或公司服务器上本地运行它们。**您不只是使用模型，而是以各种方式控制它。**

你甚至可以训练你自己的模型，因为它已经解耦了。所以可以有社区构建模型，我们都可以共享它们。

创建这种新方法的挑战是什么？

AI 堆栈的每个组件都需要大量工作。新的训练管道、在将代码发送到模型之前处理代码的新方法以及新的推理机制，仅举几例。然后最重要的是，我们需要将产品插入人工智能后端的过程——这种解耦需要相当多的工作。我们不再使用对文本进行预训练并在代码上对其进行调整的模型。我们采用空模型并在代码上从头开始训练它们。**它利用整个学习能力并在代码上训练它，而不是只占一小部分。**例如，如果您使用一些 Python 代码，它会理解代码本身的基本原理。

团队是什么样的？

我想构建人工智能模型需要大量的脑力——你的研究团队有多大？我们的研究团队非常小。我们总体上是一个扁平化的团队，没有人专注于任何一个角色。我们有很多优秀的人，他们身兼数职。对于我们的新代码生成，我们有 7-8 人参与了我们的新代码生成工具。通过精益，我们能够快速行动。

发布中未提及的任何其他有趣的人工智能和机器学习统计数据或用途值得强调，尤其是可能在企业中有应用的？请讨论诸如用于训练各种模型的数据集、如何减轻这些模型中的潜在偏差等问题。

- 在构建可信、安全的模型时，数据的重要性怎么强调都不为过。Tabnine 很早就决定只训练完全许可的代码（例如，MIT 和 Apache 2.0 许可证），以便我们的客户可以信任 AI 的输出。
- 我们已经建立了基于客户专有代码训练和保护自定义模型的独特能力，仅供其开发人员使用。
- 我们的模型可以在 Tabnine Cloud 以及客户 VPC 上运行。
- 从技术上讲，我们的代码原生模型（在我们新发布的下一代平台中具有特色）大大优于我们之前基于 GPT 的模型，因为它们只在编程语言的代码上进行训练，因此模型学习的原语是那些适合特定数据集的原语. 此外，这些模型的全部学习能力都致力于学习代码中的规律和模式，比在代码上微调的文本模型产生更好的性能。
- 我们重新设计了平台，使其可以使用由 Tabnine 和合作伙伴构建的全新模型。我们的下一代模型是代码原生的，并且比 Tabnine 早期版本中使用的模型更强大。随着 Meta 和 SFDC 等公司的新研究出现，我们也能够在此基础上构建并在代码时间内将这些模型交付给我们的开发人员。

详细的线程对比说明

- 作者：Dror Wesis，时间：2022-06-22

通常，我建议我的团队从用户价值和体验入手，但对于这个具体的比较，必须从技术入手，因为许多产品差异源于方法、架构和技术选择的差异。

微软和 OpenAI 几乎将用于软件开发的 AI 视为 GPT-3（庞然大物语言模型）的另一个用例。代码就是文本，所以他们采用了他们的语言模型，在代码上对其进行了微调，并调用了他们得到的 Codex 的庞大的 120 亿参数 AI 模型。

Copilot 的架构是单一的：“一个模型来统治它们。”它也是完全集中的——只有微软可以训练模型，而且由于训练和推理需要大量的计算资源，也只有微软可以托管模型。

Tabnine 在对不同规模的模型进行综合评估后，更倾向于个性化的语言模型协同工作。为什么？因为代码预测实际上是一组不同的子问题，不适合单片模型方法。

例如：在 Python 中根据名称生成函数的完整代码和在 Rust 中生成一行代码的后缀是 Tabnine 很好地解决的两个问题，但最适合每个此类任务的 AI 模型是不同的。

我们发现，专业模型的组合显着提高了 100 万多用户的建议精度和长度。

Tabnine 拥有不同大小（4 亿到 30 亿）的 ML 模型，每个模型都最适合特定目的。我们的代码原生模型是针对每种流行的编程语言在数据集、模型大小和关键配置方面从头开始构建的。

Tabnine 方法的一大优势是它可以为任何代码预测任务使用正确的工具，并且对于大多数目的，我们较小的模型可以快速有效地提供出色的预测。更好的是，我们的大多数模型都可以使用廉价的硬件运行。

现在我们了解了 Microsoft 的庞大单体与 Tabnine 的众多小型模型之间的主要区别，我们可以探索产品之间的差异。

为了便于讨论，我指的是新的 Tabnine Pro 产品，但 Tabnine 的大部分功能在免费的 Starter 版本中可用。

首先，一种代码建议。 Copilot 相对不频繁地查询模型并建议一个片段或一整行代码。 Copilot 不建议在中间使用代码，因为它的 AI 模型不是最适合此目的。

同样，Tabnine Pro 也建议使用完整的代码片段或代码行，但由于 Tabnine 还使用更小且高效的 AI 模型，因此它会在输入时查询模型。作为用户，这意味着 AI 与您一起流动，即使您偏离了最初建议的代码。

结果是使用 Tabnine 时使用频率和接受的代码建议数量要高得多。数量惊人的用户每天接受超过 100 条建议。

第二，训练模型的能力。 Copilot 使用一种通用 AI 模型，这意味着每个用户都可以获得基于“GitHub 平均值”的相同通用帮助，无论他们正在从事的项目是什么。

Tabnine 可以根据客户的 GitLab/GitHub/BitBucket 存储库中的特定代码训练私有 AI 模型，从而根据项目特定代码和基础设施调整建议。

由于 Tabnine 是模块化的，因此可以对客户代码进行培训，从而可以创建私人定制副本。 Tabnine 使 AI 模型创建“民主化”，使团队可以轻松训练自己的特定 AI 模型，从而显着提高组织的价值。

第三，代码安全和隐私。这有几个方面。用户无法训练或运行 Copilot 模型。单一模型始终由 Microsoft 托管。每个 Copilot 用户都将他们的代码发送给 Microsoft；不是一些代码，也不是混淆 - 全部。

使用 Tabnine，用户可以选择在哪里运行模型：在 Tabnine 云上、在本地开发人员机器上或在自托管服务器上（使用 Tabnine Enterprise）。这是可能的，因为 Tabnine 的 AI 模型可以在硬件要求适中的情况下高效运行。

这意味着，与 Copilot 相比，开发人员可以在防火墙内使用 Tabnine，而无需向 Internet 发送任何代码。

此外，Tabnine 做出坚定而明确的承诺，即用户编写的任何代码都不会用于训练我们的模型。我们不会向我们的服务器发送有关用户编写的代码以及他们收到或接受的建议的任何信息。

第四，商业条款。 Microsoft 目前仅将 Copilot 作为面向开发人员的商业产品提供，没有免费计划（免费试用之外）或组织购买。

Tabnine 有一个很棒的免费计划和收费的高级功能，例如更长的代码完成和在客户代码上训练的私有模型。我们按用户数量收取月度/年度订阅费。 我们所有的计划都符合组织要求。

从哲学上讲，Copilot 更像是微软控制一切的围墙花园。 Copilot 用户在某种程度上是微软王国的主体。 Tabnine 的客户可以训练 AI 模型、运行它们、配置建议并控制他们的 AI。

总而言之：两种产品都很棒；欢迎您尝试（Tabnine Pro），看看您更喜欢哪一个。对于专业程序员，Tabnine 提供流内完成、使 AI 适应他们的代码的能力以及卓越的代码隐私和安全性。

结论：

- 真正的力量似乎在于将重复编写相同代码模式的乏味自动化。为什么不让 AI 助手帮忙呢？

用户事例--思科工程团队：

- Tabnine可以直接安装到 IDE中，并且可以像自动完成工具一样工作。也就是说，它根据你正在编写的代码的最佳匹配来预测你想要编写的内容。它通过根据你/你的团队之前编写的代码以及最佳开源编码实践来学习编码模式来做到这一点。Tabnine 然后创建自定义建议，这些建议应该与您正在编写的其余代码非常匹配。

  在为 DevNet Create 2021 做准备时，我们 DevNet团队的一些开发人员倡导者开始合作，为即将到来的会议和演示构建一些代码示例。我们发现通过使用 Tabnine，我们的生产力提高了 50%！ 如果您以前没有使用过 SDK 或 API，那么 协作为 API构建代码和共享 SDK 知识可能会令人费解。 Tabnine 帮助我们的开发倡导团队在我们构建演示时协同工作，以更快、更一致和更安全。Tabnine 正在学习 我们所有的团队 编码实践和模式，因为我们进步了！

个人Pro付费用户(12$/m)：

- 全行和全功能代码补全
- 自然语言到代码补全
- 在线学习用户的编程模式及风格

JupyterLab的使用方法：

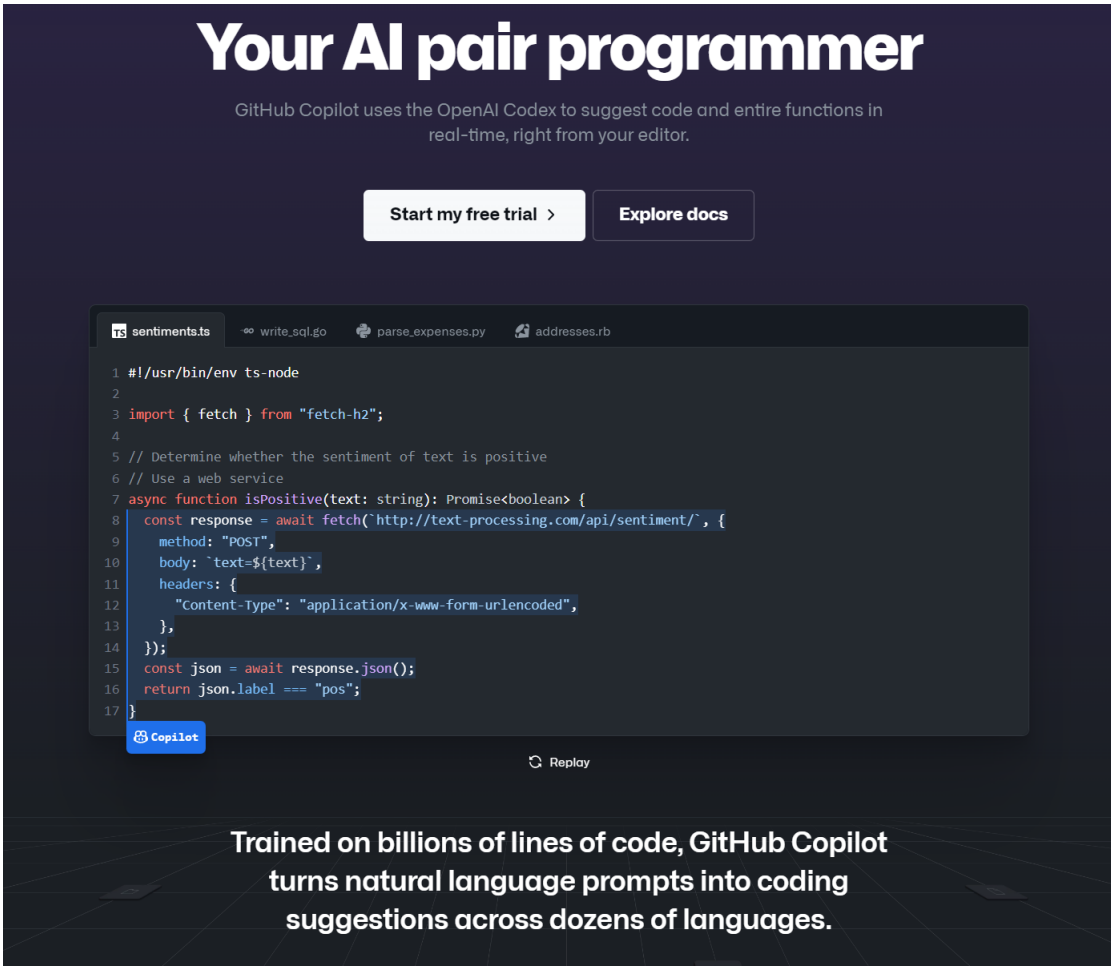
```
pip3 install jupyterlab_tabnine
```

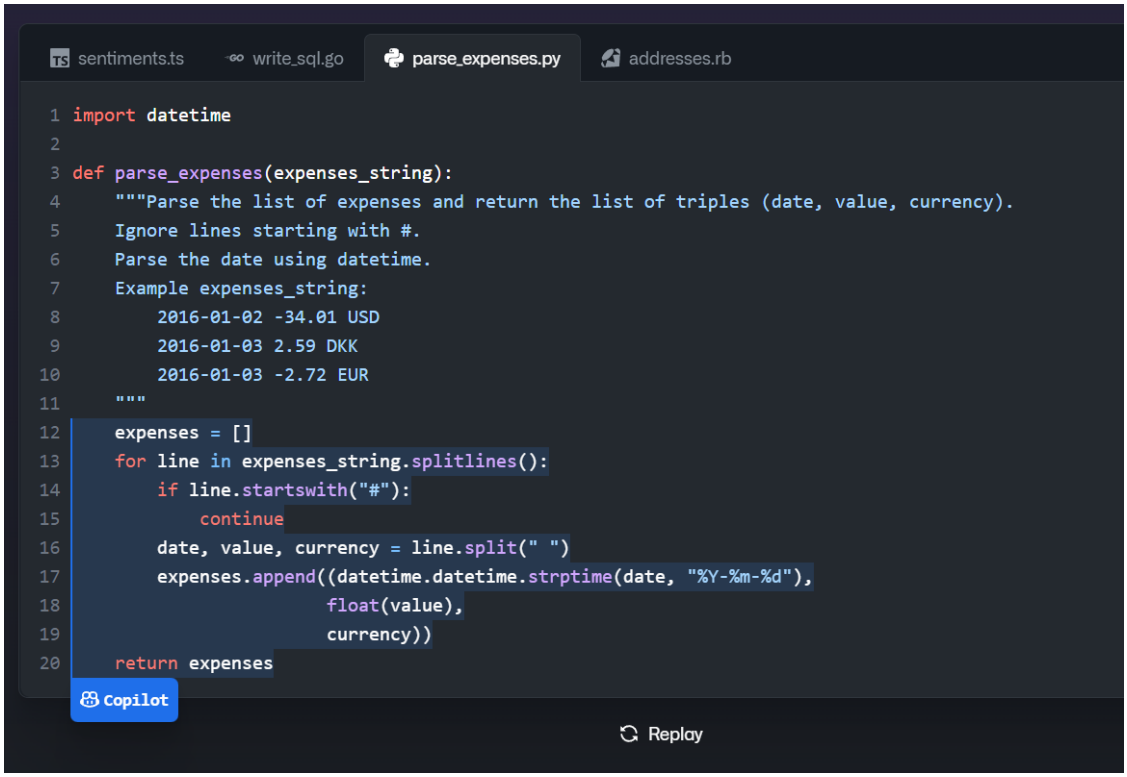
GitHub Copilot

官网：https://github.com/features/copilot

价格：10$/m

底层算法支持：Codex





1、什么是 GitHub Copilot?

GitHub Copilot 是一款 AI 结对程序员，可帮助您更快地编写代码并减少工作量。它从注释和代码中提取上下文以立即建议单个行和整个功能。GitHub Copilot 由 Codex 提供支持，Codex 是由 OpenAI 创建的生成式预训练语言模型。它可作为 Visual Studio Code、Visual Studio、Neovim 和 JetBrains 集成开发环境 (IDE) 套件的扩展。

GitHub Copilot 不适用于非编码任务，例如数据生成和自然语言生成，例如问答。

2、GitHub Copilot 是如何工作的？

OpenAI Codex 接受了公开可用的源代码和自然语言的培训，因此它适用于编程和人类语言。GitHub Copilot 扩展将您的评论和代码发送到 GitHub Copilot 服务，它依赖于上下文，如下面的隐私中所述 - 即，您正在编辑的文件中的文件内容以及项目中的相邻或相关文件。它还可以收集存储库的 URL 或文件路径以识别相关上下文。然后 OpenAI Codex 使用注释和代码以及上下文来综合和建议各个行和整个函数。

3、GitHub Copilot 训练了哪些数据？

GitHub Copilot 由 Codex 提供支持，Codex 是由 OpenAI 创建的生成式预训练 AI 模型。它已经接受了来自公开来源的自然语言文本和源代码的培训，包括 GitHub 上公共存储库中的代码。

4、GitHub Copilot 能写出完美的代码吗？

在最近的一次评估中，我们发现用户平均接受了 GitHub Copilot 显示的所有完成的 26%。我们还发现，平均超过 27% 的开发人员代码文件是由 GitHub Copilot 生成的，而在 Python 等某些语言中，这一比例高达 40%。但是，GitHub Copilot 并没有编写完美的代码。它旨在根据它可以访问的上下文生成可能的最佳代码，但它不会测试它建议的代码，因此代码可能并不总是有效，甚至没有意义。GitHub Copilot 只能保存非常有限的上下文，因此它可能无法使用项目中其他地方甚至同一文件中定义的有用函数。它可能暗示图书馆和语言的旧或不推荐使用。将用非英语编写的注释转换为代码时，与英语相比可能存在性能差异。

与任何其他代码一样，GitHub Copilot 建议的代码应经过仔细测试、审查和审查。作为开发人员，您始终负责。

5、如何充分利用 GitHub Copilot？

当您将代码划分为小函数、对函数参数使用有意义的名称以及编写好的文档字符串和注释时，GitHub Copilot 的效果最好。在帮助您浏览不熟悉的库或框架时，它似乎也做得最好。

6、GitHub Copilot 能否在其建议中引入不安全代码？

公共代码可能包含不安全的编码模式、错误或对过时 API 或惯用语的引用。当 GitHub Copilot 基于这些数据合成代码建议时，它还可以合成包含这些不良模式的代码。这是我们在 GitHub 非常关心的事情，近年来我们为开源项目提供了 GitHub Actions、Dependabot 和 CodeQL 等工具，以帮助提高代码质量。当然，您应该始终将 GitHub Copilot 与良好的测试和代码审查实践和安全工具以及您自己的判断结合使用。

7、GitHub Copilot 收集哪些数据？

GitHub Copilot 依赖文件内容和其他数据来工作。它收集数据以提供服务并保存一些数据以进行进一步分析并实现改进。有关如何使用和共享遥测数据的更多详细信息，请参阅下文。

- 用户参与数据

当您使用 GitHub Copilot 时，它将收集有关与 IDE 或编辑器交互时生成的事件的使用信息。这些事件包括用户编辑操作，例如接受和拒绝完成，以及错误和一般使用数据，以识别延迟和功能参与等指标。此信息可能包括个人数据，例如匿名标识符。

- 代码片段数据

  根据您首选的遥测设置，GitHub Copilot 还可能收集和保留以下内容，统称为“代码片段”：您正在编辑的源代码、相关文件和在同一 IDE 或编辑器中打开的其他文件、存储库的 URL 和文件路径。

底层支持算法解析--Codex

- Evaluating Large Language Models Trained on Code
- OpenAI Codex

我们介绍了 Codex，这是一种 GPT 语言模型，它在 GitHub 的公开可用代码上进行了微调，并研究了它的 Python 代码编写能力。一个独特的 Codex 生产版本为 GitHub Copilot 提供支持。在 HumanEval 上，我们发布了一个新的评估集，用于测量从文档字符串合成程序的功能正确性，我们的模型解决了 28.8% 的问题，而 GPT-3 解决了 0%，GPT-J 解决了 11.4%。此外，我们发现从模型中重复抽样是一种非常有效的策略，可以为困难的提示提供有效的解决方案。使用这种方法，我们解决了 70.2% 的问题，每个问题 100 个样本。对我们模型的仔细研究揭示了它的局限性，包括难以描述长链操作的文档字符串以及将操作绑定到变量。最后，我们讨论了部署强大的代码生成技术的潜在更广泛影响，包括安全、安保和经济。

OpenAI Codex

- OpenAI Codex Live Demo

OpenAI Codex 是 GPT-3 的后代；它的训练数据包含自然语言和来自公开来源的数十亿行源代码，包括公共 GitHub 存储库中的代码。OpenAI Codex 最擅长 Python，但它也精通十多种语言，包括 JavaScript、Go、Perl、PHP、Ruby、Swift 和 TypeScript，甚至是 Shell。它有 14KB 的 Python 代码内存，而 GPT-3 只有 4KB——因此它在执行任何任务时可以考虑超过 3 倍的上下文信息。

GPT-3 的主要技能是生成自然语言以响应自然语言提示，这意味着它影响世界的唯一方式是通过读者的思想。OpenAI Codex 具有对 GPT-3 的大部分自然语言理解，但它会生成工作代码——这意味着您可以使用 API 向任何软件发出英语命令。OpenAI Codex 使计算机能够更好地理解人们的意图，这可以使每个人都能用计算机做更多的事情。

一旦程序员知道要构建什么，编写代码的行为可以被认为是 (1) 将问题分解为更简单的问题，以及 (2) 将这些简单问题映射到已经存在的代码（库、API 或函数）存在。后一项活动可能是编程中最不有趣的部分（也是进入门槛最高的部分），也是 OpenAI Codex 最擅长的地方。

OpenAI Codex 是一种通用编程模型，这意味着它基本上可以应用于任何编程任务（尽管结果可能会有所不同）。我们已经成功地将它用于编译、解释代码和重构代码。但我们知道，我们只触及了可以做的事情的皮毛。

我们现在通过我们的 API 提供 OpenAI Codex 的私有测试版，我们的目标是尽可能安全地扩大规模。在初始阶段，OpenAI Codex 将免费提供。OpenAI 将继续建立在我们使用 GPT-3 奠定的安全基础上——审查应用程序并逐步扩展它们，同时与开发人员密切合作以了解我们的技术在世界上的影响。

该论文声称它已经解决了 10,000 个竞争性编程问题以及与持续集成相关的开源项目的问题。

论文的结论是，

> “我们发现我们的模型在难度级别可与简单面试问题相媲美的人工编写问题数据集上显示出强大的性能”

关于数据收集，论文说，

> “我们的训练数据集于 **2020** 年 **5** 月从托管在 **GitHub** 上的 **5400** 万个公共软件存储库中收集，其中包含 **179 GB** 的唯一 **Python** 文件，小于 **1 MB**。我们过滤掉了可能自动生成的文件，平均行长大于 **100**，有最大行长度大于**1000**，或者包含一小部分字母数字字符。过滤后，我们最终的数据集总计**159 GB"**

Cosy--阿里云智能编码插件

- 官网
- GitHub 官网文档地址
- 目前仅支持Java

AlphaCode--DeepMind

- DeepMind官网介绍
- arXiv--Paper

编程是一种强大且无处不在的解决问题的工具。开发可以帮助程序员甚至独立生成程序的系统可以使编程更加高效和易于访问，但迄今为止，在人工智能中融入创新已被证明具有挑战性。最近的大规模语言模型已经展示了令人印象深刻的代码生成能力，并且现在能够完成简单的编程任务。然而，当对更复杂、看不见的问题进行评估时，这些模型仍然表现不佳，这些问题需要解决问题的技能，而不仅仅是将指令翻译成代码。例如，需要了解算法和复杂自然语言的竞争性编程问题仍然极具挑战性。为了解决这一差距，我们引入了 AlphaCode，这是一种代码生成系统，可以为这些需要更深入推理的问题创建新颖的解决方案。在 Codeforces 平台最近的编程竞赛模拟评估中，AlphaCode 在超过 5000 名参与者的竞赛中平均排名前 54.3%。我们发现三个关键组件对于实现良好和可靠的性能至关重要：(1) 用于训练和评估的广泛而干净的竞争性编程数据集，(2) 大型且高效采样的基于转换器的架构，以及 (3) 大型-规模模型抽样以探索搜索空间，然后根据程序行为过滤到一小组提交。

Datasets

数据源：

- codeparrot/github-code
  - 数据集描述

    GitHub代码数据集由来自GitHub的1.15亿个代码文件组成，这些文件有32种编程语言和60个扩展名，总共有1TB的数据。该数据集是由谷歌BiqQuery上的公共GitHub数据集创建的。

  - 如何使用它

    GitHub Code 数据集是一个非常大的数据集，因此对于大多数用例，建议使用数据集的流式 API。您可以使用以下两行代码加载和迭代数据集：

```
from datasets import load_dataset

ds = load_dataset("codeparrot/github-code", streaming=True, split='train')
print(next(iter(ds)))

#OUTPUT:
{
 'code': "import mod189 from './mod189';\nvar value=mod189+1;\nexport default
value;\n",
 'repo_name': 'MirekSz/webpack-es6-ts',
 'path': 'app/mods/mod190.js',
 'language': 'JavaScript',
 'license': 'isc',
 'size': 73
}
```

    你可以看到，除了代码、 repo名称和路径之外，编程语言、许可证和文件的大小也是数据集的一部分。你也可以过滤数据集中的30种语言（见下面的完整列表）的任何子集。只要把语言列表作为一个列表传过去。例如，如果你的梦想是为Dockerfiles建立一个Codex模型，请使用以下配置。

```
ds = load_dataset("codeparrot/github-code", streaming=True, split="train",
languages=["Dockerfile"])
print(next(iter(ds))["code"])

#OUTPUT:
"""\
FROM rockyluke/ubuntu:precise

ENV DEBIAN_FRONTEND="noninteractive" \
    TZ="Europe/Amsterdam"
...
"""
```

    我们还可以访问一个文件的原始版本的许可证，所以我们可以用过滤语言的同样方法来过滤许可证。

```
ds = load_dataset("codeparrot/github-code", streaming=True, split="train",
licenses=["mit", "isc"])

licenses = []
for element in iter(ds).take(10_000):
    licenses.append(element["license"])
print(Counter(licenses))

#OUTPUT:
Counter({'mit': 9896, 'isc': 104})
```

当然，你也可以下载完整的数据集。请注意，这将下载300GB的压缩文本数据，未压缩的数据集将占用1TB的存储空间：

```
ds = load_dataset("codeparrot/github-code", split="train")
```

- Data Structure
  - Data Instances

```
{
 'code': "import mod189 from './mod189';\nvar value=mod189+1;\nexport
default value;\n",
 'repo_name': 'MirekSz/webpack-es6-ts',
 'path': 'app/mods/mod190.js',
 'language': 'JavaScript',
 'license': 'isc',
 'size': 73
}
```

  - Data Fields

Field	Type	Description
code	string	content of source file
repo_name	string	name of the GitHub repository
path	string	path of file in GitHub repository
language	string	programming language as inferred by extension
license	string	license of GitHub repository
size	int	size of source file in bytes

  - Data Splits

    该数据集只包含一个训练分割。

  - Languages

    该数据集包含 30 种编程语言和 60 多个扩展：

```
{
    "Assembly": [".asm"],
```

```
    "Batchfile": [".bat", ".cmd"],
    "C": [".c", ".h"],
    "C#": [".cs"],
    "C++": [".cpp", ".hpp", ".c++", ".h++", ".cc", ".hh", ".C", ".H"],
    "CMake": [".cmake"],
    "CSS": [".css"],
    "Dockerfile": [".dockerfile", "Dockerfile"],
    "FORTRAN": ['.f90', '.f', '.f03', '.f08', '.f77', '.f95', '.for',
'.fpp'],
    "GO": [".go"],
    "Haskell": [".hs"],
    "HTML":[".html"],
    "Java": [".java"],
    "JavaScript": [".js"],
    "Julia": [".jl"],
    "Lua": [".lua"],
    "Makefile": ["Makefile"],
    "Markdown": [".md", ".markdown"],
    "PHP": [".php", ".php3", ".php4", ".php5", ".phps", ".phpt"],
    "Perl": [".pl", ".pm", ".pod", ".perl"],
    "PowerShell": ['.ps1', '.psd1', '.psm1'],
    "Python": [".py"],
    "Ruby": [".rb"],
    "Rust": [".rs"],
    "SQL": [".sql"],
    "Scala": [".scala"],
    "Shell": [".sh", ".bash", ".command", ".zsh"],
    "TypeScript": [".ts", ".tsx"],
    "TeX": [".tex"],
    "Visual Basic": [".vb"]
}
```

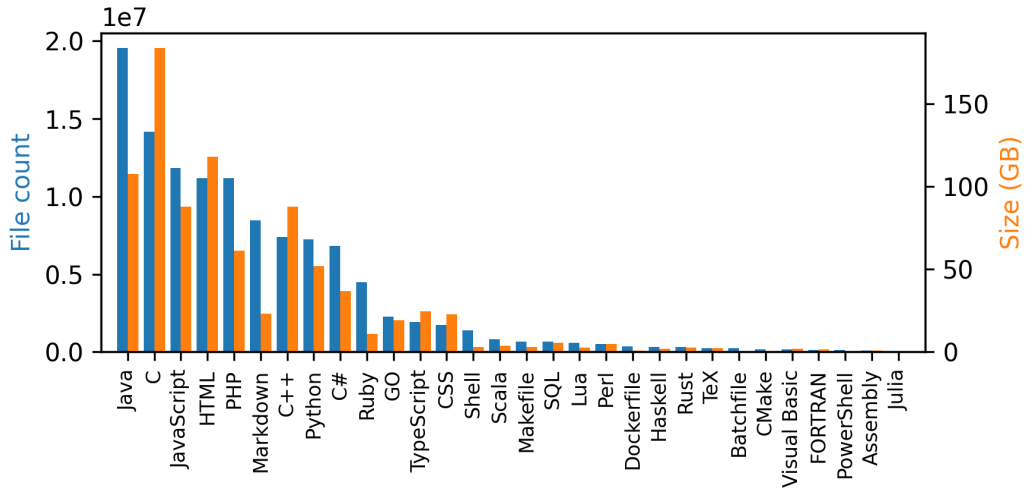
- Licenses

  每个例子都标明了相关资源库的许可证。总共有15个许可证。

```
[
  'mit',
  'apache-2.0',
  'gpl-3.0',
  'gpl-2.0',
  'bsd-3-clause',
  'agpl-3.0',
  'lgpl-3.0',
  'lgpl-2.1',
  'bsd-2-clause',
  'cc0-1.0',
  'epl-1.0',
  'mpl-2.0',
  'unlicense',
  'isc',
  'artistic-2.0'
 ]
```

- Dataset Statistics

  该数据集包含 115M 个文件，所有源代码文件大小的总和为 873 GB（请注意，由于额外的字段，数据集的大小更大）。下表中给出了每种语言的细分：



	Language	File Count	Size (GB)
0	Java	19548190	107.70
1	C	14143113	183.83
2	JavaScript	11839883	87.82
3	HTML	11178557	118.12
4	PHP	11177610	61.41
5	Markdown	8464626	23.09
6	C++	7380520	87.73
7	Python	7226626	52.03
8	C#	6811652	36.83
9	Ruby	4473331	10.95
10	GO	2265436	19.28
11	TypeScript	1940406	24.59
12	CSS	1734406	22.67
13	Shell	1385648	3.01
14	Scala	835755	3.87
15	Makefile	679430	2.92
16	SQL	656671	5.67

	Language	File Count	Size (GB)
17	Lua	578554	2.81
18	Perl	497949	4.70
19	Dockerfile	366505	0.71
20	Haskell	340623	1.85
21	Rust	322431	2.68
22	TeX	251015	2.15
23	Batchfile	236945	0.7
24	CMake	175282	0.54
25	Visual Basic	155652	1.91
26	FORTRAN	142038	1.62
27	PowerShell	136846	0.69
28	Assembly	82905	0.78
29	Julia	58317	0.29

- Dataset Creation

  数据集的创建分两步：

  1. 具有上述列表中给出的扩展名的文件是从 BigQuery 上的 GitHub 数据集检索的（此处为完整查询）。该查询于 2022 年 3 月 16 日下午 6:23:39 UTC+1 执行。
  2. 行数超过1000个字符的文件和重复的文件（完全重复，忽略空白）被删除（完整的预处理脚本在这里）。

- 使用数据的注意事项

  该数据集由来自各种存储库的源代码组成。因此，它们可能包含有害或有偏见的代码以及密码或用户名等敏感信息。

- Releases

  您可以使用 revision 参数加载任何旧版本的数据集：

  ```
  ds = load_dataset("codeparrot/github-code", revision="v1.0")
  ```

  - v1.0
    - Initial release of dataset
    - The query was executed on *Feb 14, 2022, 12:03:16 PM UTC+1*
  - v1.1
    - Fix missing Scala/TypeScript
    - Fix deduplication issue with inconsistent Python `hash`
    - The query was executed on *Mar 16, 2022, 6:23:39 PM UTC+1*

- code_contests
- The Pile

- code_search_net
- APPS (benchmarks)
- MBPP (benchmarks)
- facebook/incoder-6B
- openai_humanval
- codeparrot/codeparrot-clean

说明：

- 由于已针对 codeparrot/github-code 数据集进行了较为详细的介绍，且该数据集目前已是目前可获得的最完备的代码相关数据集，故此处不再对其他数据集进行更详细的说明，想了解详情可根据提供的链接进行查看。

Reference

Paper

- CERT: Continual Pre-Training on Sketches for Library-Oriented Code Generation
  - 14 Jun 2022 14:44:34 UTC
- Evaluating Large Language Models Trained on Code
  - 14 Jul 2021 17:16:02 UTC
- Competition-Level Code Generation with AlphaCode
  - 8 Feb 2022 23:16:31 UTC
- CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning
  - 5 Jul 2022 02:42:15 UTC
  - GitHub official code
- Natural Language Processing with Transformers Tunstall et al., 2022.
- Evaluating large language models trained on code Chen et al., 2021.
- InCoder: A Generative Model for Code Infilling and Synthesis Fried et al., 2022.
- A Conversational Paradigm for Program Synthesis Nijkamp et al. 2022.
- A systematic evaluation of large language models of code Xu et al. 2022.

GitHub 相关开源

- microsoft/PyCodeGPT
  - A pre-trained GPT model for Python code completion and generation. PyCodeGPT 是高效且有效的基于 GPT-Neo 的 Python 代码生成任务模型，类似于OpenAI Codex、Github Copliot、CodeParrot、AlphaCode。

- 训练数据

  由于公开发布的数据集规模较小，我们建议从 GitHub 从头开始收集数据。我们首先爬取了 120 万个由 GitHub 托管的与 python 相关的存储库。然后，我们使用这些存储库 URL 从 GitHub 下载每个存储库的所有内容。之后，我们得到了 1MB 以下的 60M 原始 python 文件，总大小为 330GB。最后，我们精心设计了各种数据清洗策略，得到了大约 96GB 的数据进行训练。详情请参阅下表。

模型--Model	存储库--Repositories	Size and file after filtering
CodeParrot	0.56M	12GB(compressed), 5.4M
Codex	54M	159GB
PyCodeGPT	1.2M	96GB, 13M

- 模型精度对比表

Model	Pass@1	Pass@10	Pass@100
PyCodeGPT-110M	**8.32%**	**13.53%**	**18.3%**
GPT-Neo 125M	0.75%	1.88%	2.97%
GPT-Neo 1.3B	4.97%	7.47%	16.3%
GPT-Neo 2.7B	6.41%	11.27%	21.37%
GPT-J 6B	11.62%	15.74%	27.74%
TabNine	2.58%	4.35%	7.59%
CodeParrot 110M	3.80%	6.57%	12.78%
CodeParrot 1.5B	3.58%	8.03%	14.96%
Codex 12M	2.00%	3.62%	8.58%
Codex 25M	3.21%	7.1%	12.89%
Codex 42M	5.06%	8.8%	15.55%
Codex 85M	8.22%	12.81%	22.4%
Codex 300M	13.17%	20.37%	36.27%
Codex 679M	16.22%	25.7%	40.95%
Codex 2.5B	21.36%	35.42%	59.5%
Codex 12B	28.81%	46.81%	72.31%
Pretrained Decoder-only 13M (AlphaCode)	1.5%	3.6%	8.6%
Pretrained Decoder-only 29M (AlphaCode)	3.4%	5.8%	11.2%
Pretrained Decoder-only 55M (AlphaCode)	4.2%	8.2%	16.9%
Pretrained Decoder-only 89M (AlphaCode)	4.3%	12.2%	20.0%
Pretrained Decoder-only 302M (AlphaCode)	11.6%	18.8%	31.8%

Model	Pass@1	Pass@10	Pass@100
Pretrained Decoder-only 685M (AlphaCode)	14.2%	24.4%	38.8%
Pretrained Decoder-only 1.1B (AlphaCode)	17.1%	28.2%	45.3%
PolyCoder 160M	2.13%	3.35%	4.88%
PolyCoder 400M	2.96%	5.29%	11.59%
PolyCoder 2.7B	5.59%	9.84%	17.68%

- CodeRL
  - 这是论文 CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning 的官方代码。

    程序综合或代码生成旨在生成满足问题规范的程序。最近使用大规模预训练语言模型 (LM) 的方法已显示出可喜的结果，但它们也存在一些严重的局限性。特别是，他们经常遵循标准的监督微调程序，仅从自然语言问题描述和真实程序对训练代码生成模型。这种范式在很大程度上忽略了问题规范中的一些重要但可能有用的信号，例如单元测试，因此在解决复杂的看不见的编码任务时通常会导致性能下降。为了解决这些限制，我们提出了“CodeRL”，这是一个通过预训练 LM 和深度强化学习 (RL) 进行程序合成任务的新框架。具体来说，在训练过程中，我们将生成代码的 LM 视为一个演员网络，并引入一个经过训练的评论家网络，以预测生成程序的功能正确性并为演员提供密集的反馈信号。在推理过程中，我们引入了具有关键采样策略的新一代过程，该策略允许模型根据来自示例单元测试和批评分数的反馈自动重新生成程序。对于模型主干，我们扩展了 CodeT5 的编码器-解码器架构，具有增强的学习目标、更大的模型尺寸和更好的预训练数据。我们的方法不仅在具有挑战性的 APPS 基准上取得了新的 SOTA 结果，而且在更简单的 MBPP 基准上显示出强大的零样本传输能力和新的 SOTA 结果。

  - Benchmarks -- 主要方案综合基准
    - APPS
      - 请按照此处提供的下载和预处理说明进行操作。
    - MBPP
      - 数据集可在此处获得。
  - Models

我们将发布以下预训练/微调模型检查点：

- CodeT5：使用 Next Token Prediction 学习目标和 GCPY 数据集预训练的 CodeT5-770M 模型；
- CodeRL+CodeT5：上面预训练的 CodeT5 模型，它按照我们的 CodeRL 训练框架在 APPS 上进行了微调；
- Critic：一个 CodeT5 模型，它从 CodeT5 基础初始化并训练为分类器以预测单元测试结果。Critic 用于估计回报并促进 RL 微调。

- Code generation with HuggingFace
  - 这是一个交互式博客，概述了用于代码生成的开源语言模型。这篇文章介绍：
    - code datasets
    - model architecture
    - model evaluation
  - Introduction

    语言模型在代码生成中的应用最近引起了极大的兴趣。您可能听说过 Codex，Github Copilot 背后的模型，或用于竞赛级编程的 AlphaCode。这些模型不是开源的，在有限的预算和不完整的培训信息的情况下很难复制它们。幸运的是，ML 社区贡献了一些代码模型，以供进一步研究。

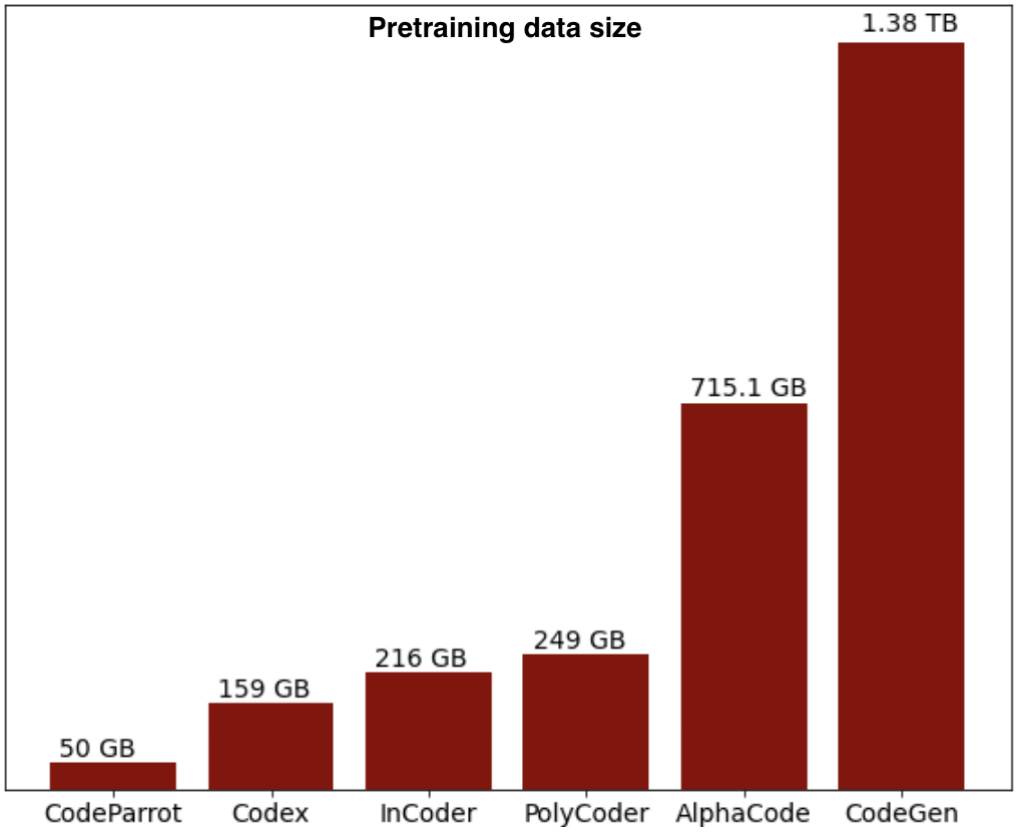
    然而，在各种模型之间迷失方向是很容易的。在Hugging Face，我们的目标是使ML民主化，并集中生态系统中的所有信息，使开源工具的使用更容易、更有效。代码模型也不例外，你可以在Hub上找到所有的开源模型，还有一些代码数据集和评估指标。在这篇博客中，我们将对这些工具以及如何使用它们进行概述。

  - 1 - Code datasets

    大多数代码模型都是根据托管在 GitHub 上的公共软件存储库中的数据进行训练的。有些还包括代码以及来自 Stack Overflow 等平台的自然文本。可以根据模型的目标任务制作其他数据集。例如，Alphacode 在 CodeContests 上进行了微调，这是一个用于机器学习的竞争性编程数据集。另一个流行的数据集是 The Pile，它是一个大型语料库，包含自然语言文本和来自不同来源的代码，例如 StackExchange 转储和流行（>100 星）GitHub 存储库。它对于旨在将自然文本转换为代码或相反的模型非常有效，例如在 CodeGen 中使用。

    下面是一些代码模型的预训练数据大小分布，我们在本节后面提供开源模型的模型特定信息：



在Hub上还有其他一些有用的数据集，如CodeSearchNet，这是一个由GitHub上的开源库中的200万个（注释，代码）对组成的语料库，适用于几种编程语言；Mostly Basic Python Problems（mbpp），这是一个由大约1000个众包的Python编程问题组成的基准，适用于初级程序员，每个问题包括任务描述、代码解决方案和3个自动测试案例，这个数据集被用于InCoder评估，以及我们后面要介绍的HumanEval。你还可以找到APPS，这是一个包含10000个英文编程问题和Python代码解决方案的基准，这个数据集也和HumanEval一起被用于Codex评估。

我们还发布了Github code dataset，这是一个1TB的代码数据，来自32种编程语言的Github存储库。它是由Google BigQuery上的公共GitHub数据集创建的。如果你因为内存限制而不想下载该数据集，可以用流模式加载。(上面已详细介绍！)

预训练模型选择：

- CodeParrot

  CodeParrot 是一个代码生成模型，它使用来自 Github 存储库的 50GB 预处理 Python 数据进行训练：CodeParrot 数据集。原始数据集包含大量重复和嘈杂的数据。因此，通过以下步骤清理数据集：

  - 完全匹配的重复数据删除功能
  - 过滤
    - Average line length < 100 tokens
    - Maximum line length < 1000 MB
    - Alphanumeric characters fraction > 0.25
    - Remove auto-generated files (keyword search)

    有关更多详细信息，请参阅此处的转换器存储库中的预处理脚本。

- InCoder

InCoder 是一个代码生成模型，它还允许通过 infilling--填充进行代码编辑。它使用来自 GitHub 的 216 GB 预处理数据和来自 28 种编程语言的 Stack Overflow 进行训练。 52 GB 是 Python，107 GB 是其他编程语言，57 GB 是来自 Stackoverflow 的非代码内容。

通过以下步骤清理了 GitHub 数据：

- Average line length < 100 tokens
- Maximum line length < 3000 MB
- Alphanumeric characters fraction > 0.4
- Remove auto-generated files (keyword search)

数据的第二部分包括来自 Stack Overflow 的问题、答案和评论。这包括：

- all questions that have at least one answer
- up to ten answers with a non-negative score (sorted by score) per question
- up to five comments per question/answer

对代码文件进行了精确匹配的重复数据删除。更多细节请参考本文。

- CodeGen

  Codegen 是一种对话式程序合成模型，其中每个问题都通过多个步骤交互式解决，每个步骤由用户的自然语言规范和系统的合成子程序组成。

  它在三个数据集上依次训练：

  - The Pile
  - 来自多种编程语言的 Google BigQuery data 文件数据集的 341GB 子集，仅保留 6 个：C、C++、Go、Java、JavaScript 和 Python
  - 来自 GitHub 存储库的 217GB Python 数据

  第二个和第三个数据集使用了以下预处理：

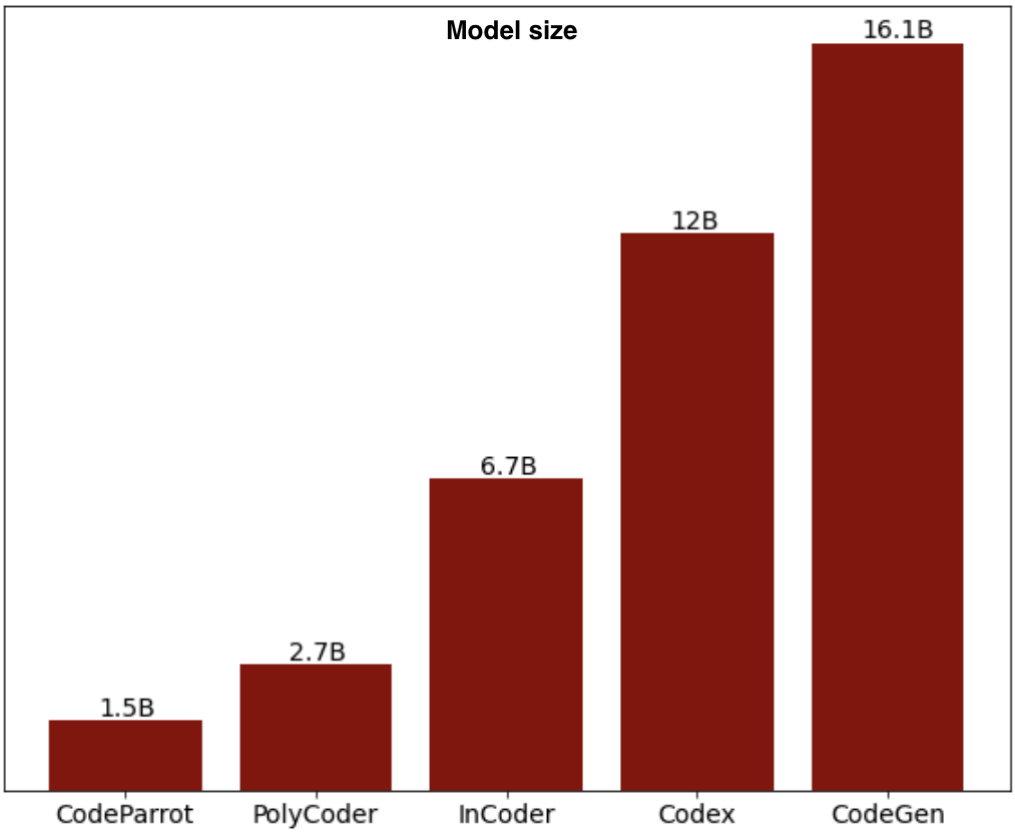
  - Exact match deduplication
  - Filtering:
    - Exact match deduplication
    - Average line length < 100 tokens
    - Maximum line length < 1000 MB
    - Characters being decimal or hexadecimal digits >90%
- PolyCoder

  PolyCoder 论文对现有代码模型进行了很好的比较。作者还在 249GB 数据上训练了一个代码生成模型，在预处理后，该模型由 2021 年 10 月来自 GitHub 的 12 种流行编程语言的流行存储库组成，至少 50 颗星。数据使用了以下预处理：

  - Exact match deduplication
  - Filtering:
    - Average line length < 100 tokens
    - Maximum line length < 1000 MB

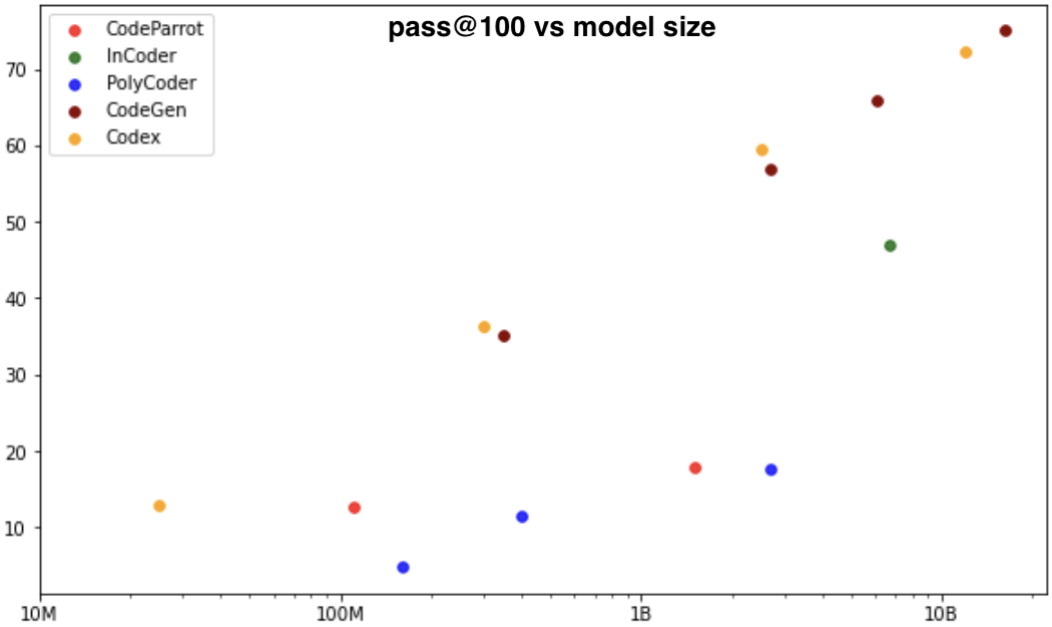
- 2 - Model architecture

在代码生成模型中使用了各种架构，但大多数都使用自动回归的从左到右设置，如GPT。然而InCoder使用了一个只有解码器的Transformer与Causal Masking目标，它通过掩码将下一个标记的预测和双向的上下文结合起来。AlphaCode使用了一个编码器-解码器架构。



- 3 - Code model evaluation

  评估代码程序的一种自然方法是查看它们是否通过单元测试，这是在 Codex 论文中介绍的 HumanEval 数据集上流行的代码生成模型评估框架 pass@k 指标背后的想法。该数据集包括 164 个手写编程问题。在 pass@k 指标中，每个问题生成 k 个代码样本，如果任何样本通过单元测试并报告已解决问题的总比例，则认为问题已解决。在大多数论文中，对 200 个候选程序完成进行采样，并使用无偏采样估计器计算 pass@1、pass@10 和 pass@100。



结论

从目前的调研信息来看，代码补全所需要的数据集问题已得到解决，且目前的开源可借鉴的代码案例也具备，故从技术上来说，已具备进行项目推进的客观条件。可以进行前期的代码及模型算法的复现相关工作，并根据评估情况进行下一步的项目规划，并对其他可能存在的问题进行排查。

需要对目前的项目需求进行细化，即指定较为全面细致的项目需求及后续的项目验收标准等相关指标。