Rensselaer

why not change the world?®

RPIrates: Fun with OpenAI, GPTStudio and R!

Tuesday, 14 Feb 2023

RPIrates: The RPI R Users Group
The Rensselaer IDEA
Rensselaer Polytechnic Institute

IDEA

Rensselaer Institute for Data Exploration and Applications

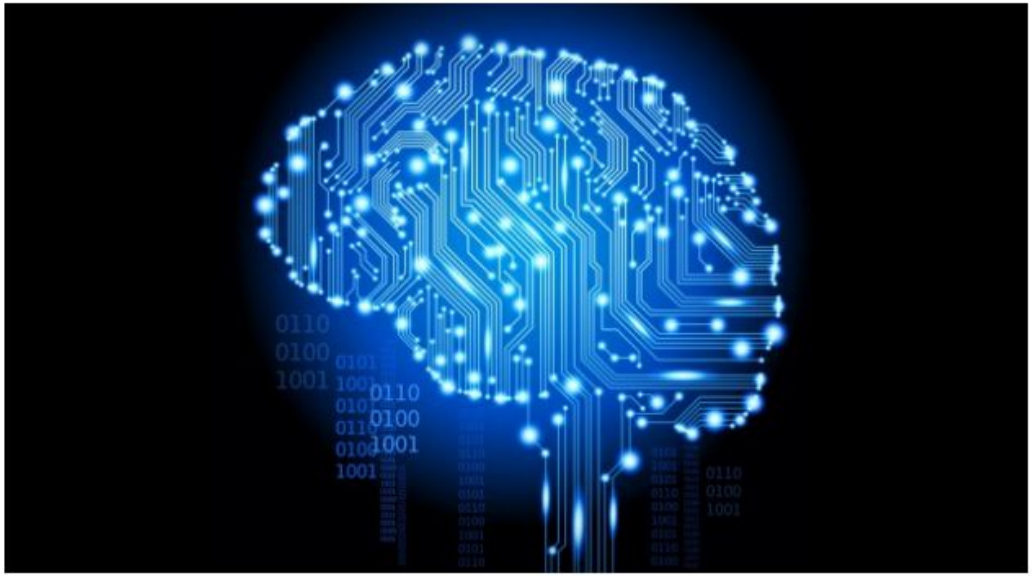
RPIrates

The RPI R Users Group

https://bit.ly/3S6Lfer

New AI Writes Its Own Computer Code

By Jessica Hall on September 14, 2021 at 10:15 am | Comments



Sometimes, when you feed an AI content from the Internet, it learns natural language. Sometimes, it reads the entire contents of GitHub and learns to produce simple snippets of code.

Sept. 9, 2021

https://nyti.ms/3Z86tLt

A.I. Can Now Write Its Own Computer Code. That's Good News for Humans.

A new technology called Codex generates programs in 12 coding languages and even translates between them. But it is not a threat to professional programmers.

The paper that started it all…

https://arxiv.org/pdf/1706.03762.pdf

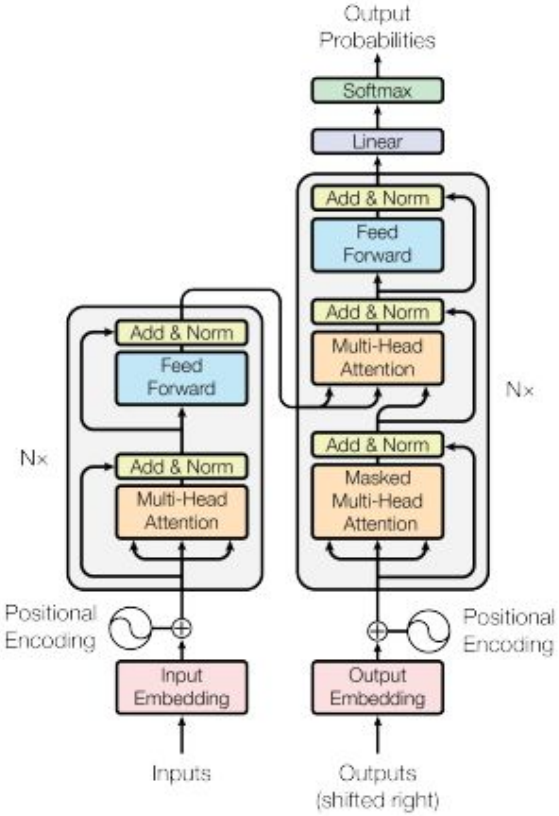


Figure 1: The Transformer - model architecture.

Dec 2017 (Google Brain)

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

The paper that started it all…

https://arxiv.org/pdf/1706.03762.pdf

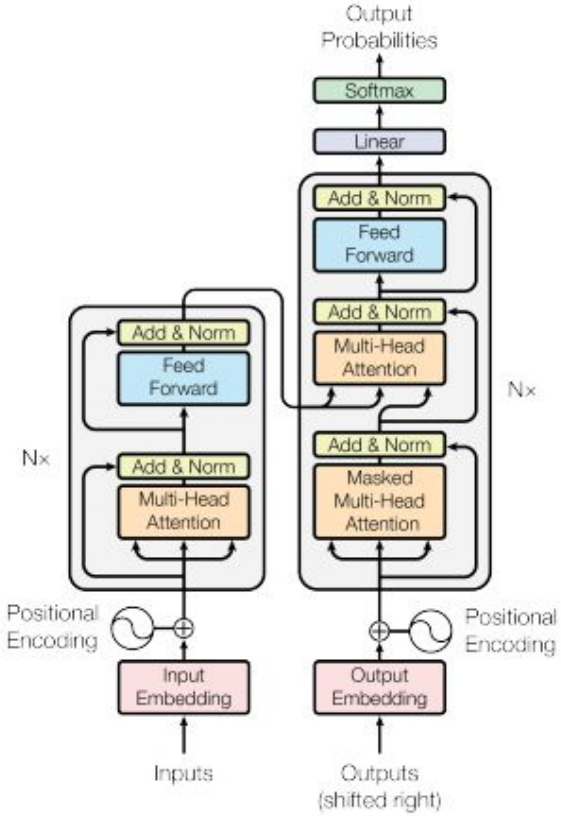


Figure 1: The Transformer - model architecture.

Dec 2017 (Google Brain)

Attention Is All You Need

Ashish Vaswa… it*
Google B…
avaswani@…

Li…

"…We are excited about the future of attention-based models and plan to apply them to other tasks…"

convolu… est
performing … rough an attention
mechanism. We … ure, the Transformer,
based solely on attention mechanisms, … currence and convolutions
entirely. Experiments on two machine translation tasks show these models to

training for 3.5 days on eight GPUs, a small fraction of the training costs of the
best models from the literature. We show that the Transformer generalizes well to
other tasks by applying it successfully to English constituency parsing both with
large and limited training data.

See https://bit.ly/3lv78YB or https://bit.ly/3YO9HDE for two great explanations!

Oct 2020 (MSFT)

https://arxiv.org/pdf/2005.08025.pdf

IntelliCode Compose: Code Generation using Transformer

Alexey Svyatkovskiy*
Microsoft
Redmond, WA, USA
alsvyatk@microsoft.com

Shao Kun Deng*
Microsoft
Redmond, WA, USA
shade@microsoft.com

Shengyu Fu
Microsoft
Redmond, WA, USA
shengyfu@microsoft.com

Neel Sundaresan
Microsoft
Redmond, WA, USA
neels@microsoft.com

ABSTRACT

In software development through integrated development environments (IDEs), code completion is one of the most widely used features. Nevertheless, majority of integrated development environments only support completion of methods and APIs, or arguments.

In this paper, we introduce IntelliCode Compose – a general-purpose multilingual code completion tool which is capable of predicting sequences of code tokens of arbitrary types, generating up to entire lines of syntactically correct code. It leverages state-of-the-art generative transformer model trained on 1.2 billion lines of source code in Python, C#, JavaScript and TypeScript programming languages. IntelliCode Compose is deployed as a cloud-based web service. It makes use of client-side tree-based caching, efficient parallel implementation of the beam search decoder, and compute graph optimizations to meet edit-time completion suggestion requirements in the Visual Studio Code IDE and Azure Notebook.

Our best model yields an average edit similarity of 86.7% and a perplexity of 1.82 for Python programming language.

1 INTRODUCTION

Machine learning has shown a great promise towards improving automated software engineering across all stages. Some of the early applications of machine learning of source code include code search [1, 2], bug detection and localization [3], program synthesis [4], code summarization [5] and code completion [6–10].

There are numerous code completion systems capable of effectively recommending method and API calls [6, 9–11], or finding the correct argument [12–14]. Majority of argument completion systems would, however, only work when the name of the method or API call is already typed in, thus leaving the task of completing the method calls to software developers.

In this paper, we introduce IntelliCode Compose – a general-purpose code completion framework, capable of generating code sequences of arbitrary token types, including local variables, methods or APIs, arguments, as well as punctuation, language keywords, and delimiters. IntelliCode Compose serves as a universal programming language modeling tool, effectively generating syntactically correct code in multiple programming languages, capable of com-

Jul 2021 (OpenAI)

https://arxiv.org/pdf/2107.03374.pdf

Evaluating Large Language Models Trained on Code

Mark Chen [*1] Jerry Tworek [*1] Heewoo Jun [*1] Qiming Yuan [*1] Henrique Ponde de Oliveira Pinto [*1]
Jared Kaplan [*2] Harri Edwards [1] Yuri Burda [1] Nicholas Joseph [2] Greg Brockman [1] Alex Ray [1] Raul Puri [1]
Gretchen Krueger [1] Michael Petrov [1] Heidy Khlaaf [3] Girish Sastry [1] Pamela Mishkin [1] Brooke Chan [1]
Scott Gray [1] Nick Ryder [1] Mikhail Pavlov [1] Alethea Power [1] Lukasz Kaiser [1] Mohammad Bavarian [1]
Clemens Winter [1] Philippe Tillet [1] Felipe Petroski Such [1] Dave Cummings [1] Matthias Plappert [1]
Fotios Chantzis [1] Elizabeth Barnes [1] Ariel Herbert-Voss [1] William Hebgen Guss [1] Alex Nichol [1] Alex Paino [1]
Nikolas Tezak [1] Jie Tang [1] Igor Babuschkin [1] Suchir Balaji [1] Shantanu Jain [1] William Saunders [1]
Christopher Hesse [1] Andrew N. Carr [1] Jan Leike [1] Josh Achiam [1] Vedant Misra [1] Evan Morikawa [1]
Alec Radford [1] Matthew Knight [1] Miles Brundage [1] Mira Murati [1] Katie Mayer [1] Peter Welinder [1]
Bob McGrew [1] Dario Amodei [2] Sam McCandlish [2] Ilya Sutskever [1] Wojciech Zaremba [1]

Abstract

We introduce Codex, a GPT language model fine-tuned on publicly available code from GitHub, and study its Python code-writing capabilities. A distinct production version of Codex powers GitHub Copilot. On HumanEval, a new evaluation set we release to measure functional correctness for synthesizing programs from docstrings, our model solves 28.8% of the problems, while GPT-3 solves 0% and GPT-J solves 11.4%. Furthermore, we find that repeated sampling from the model is a surprisingly effective strategy for producing working solutions to difficult prompts. Us-

1. Introduction

Scalable sequence prediction models (Graves, 2014; Vaswani et al., 2017; Child et al., 2019) have become a general-purpose method for generation and representation learning in many domains, including natural language processing (Mikolov et al., 2013; Sutskever et al., 2014; Dai & Le, 2015; Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018), computer vision (Van Oord et al., 2016; Menick & Kalchbrenner, 2018; Chen et al., 2020; Bao et al., 2021), audio and speech processing (Oord et al., 2016; 2018; Dhariwal et al., 2020; Baevski et al., 2020), biology (Alley et al., 2019; Rives et al., 2021), and even across multiple modalities (Das et al., 2017; Lu et al., 2019; Ramesh et al., 2021; Zellers et al., 2021). More recently, language models have

Feb 2022 (DeepMind)

DeepMind https://arxiv.org/pdf/2203.07814.pdf 2022-3-16

Competition-Level Code Generation with AlphaCode

Yujia Li*, David Choi*, Junyoung Chung*, Nate Kushman*, Julian Schrittwieser*, Rémi Leblond*, Tom Eccles*, James Keeling*, Felix Gimeno*, Agustin Dal Lago*, Thomas Hubert*, Peter Choy*, Cyprien de Masson d'Autume*, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu and Oriol Vinyals

*Joint first authors

Programming is a powerful and ubiquitous problem-solving tool. Developing systems that can assist programmers or even generate programs independently could make programming more productive and accessible, yet so far incorporating innovations in AI has proven challenging. Recent large-scale language models have demonstrated an impressive ability to generate code, and are now able to complete simple programming tasks. However, these models still perform poorly when evaluated on more complex, unseen problems that require problem-solving skills beyond simply translating instructions into code. For example, competitive programming problems which require an understanding of algorithms and complex natural language remain extremely challenging. To address this gap, we introduce AlphaCode, a system for code generation that can create novel solutions to these problems that require deeper reasoning. In simulated evaluations on recent programming competitions on the Codeforces platform, AlphaCode achieved on average a ranking of top 54.3% in competitions with more than 5,000 participants. We found that three key components were critical to achieve good and reliable performance: (1) an extensive and clean competitive programming dataset for training and evaluation, (2) large and efficient-to-sample transformer-based architectures, and (3) large-scale model sampling to explore the search space, followed by filtering based on program behavior to a small set of submissions.

Dec 2022 (DeepMind)

https://www.science.org/doi/10.1126/science.abq1158

RESEARCH ARTICLE | COMPUTER SCIENCE

Competition-level code generation with AlphaCode

YUJIA LI, DAVID CHOI, JUNYOUNG CHUNG, NATE KUSHMAN, JULIAN SCHRITTWIESER, RÉMI LEBLOND, TOM ECCLES, JAMES KEELING, FELIX GIMENO, [...], AND ORIOL VINYALS

+16 authors | Authors Info & Affiliations

SCIENCE • 8 Dec 2022 • Vol 378, Issue 6624 • pp. 1092-1097 • DOI: 10.1126/science.abq1158

Machine learning systems can program too

Computer programming competitions are popular tests among programmers that require critical thinking informed by experience and creating solutions to unforeseen problems, both of which are key aspects of human intelligence but challenging to mimic by machine learning models. Using self-supervised learning and an encoder-decoder transformer architecture, Li *et al.* developed AlphaCode, a deep-learning model that can achieve approximately human-level performance on the Codeforces platform, which regularly hosts these competitions and attracts numerous participants worldwide (see the Perspective by Kolter). The development of such coding platforms could have a huge impact on programmers' productivity. It may even change the culture of programming by shifting human work to formulating problems, with machine learning being the main one responsible for generating and executing codes. —YS

Dec 2022
(RE: AlphaCode)

Science

https://www.science.org/doi/10.1126/science.add8258

PERSPECTIVE | COMPUTER SCIENCE

AlphaCode and "data-driven" programming

Is ignoring everything that is known about code the best way to write programs?

J. ZICO KOLTER Authors Info & Affiliations

SCIENCE · 8 Dec 2022 · Vol 378, Issue 6624 · p. 1056 · DOI: 10.1126/science.add8258

RELATED RESEARCH ARTICLE

Competition-level code generation with AlphaCode

BY YUJIA LI, DAVID CHOI, JUNYOUNG CHUNG, ET AL.

Competitive programming problems represent a challenging task for even skilled programmers: Given a short natural language description of an algorithmic problem, contestants must quickly write a program that solves the task. On page 1092 of this issue, Li *et al.* (*1*) present the AlphaCode system, which represents a substantial step forward in the development of machine learning (ML) models that can synthesize computer programs to solve these types of challenging problems. But what is perhaps most surprising about the system is what AlphaCode does not do: AlphaCode contains no explicit built-in knowledge about the structure of computer code. Instead, AlphaCode relies on a purely "data-driven" approach to writing code, learning the structure of computer programs by simply observing lots of existing code.

Feb 2023 (RE: AlphaCode)

https://bit.ly/3XDYqow

Science

Current Issue First release papers Archive About Submit manuscrip

LETTER

Understanding the limits of AI coding

JAMES HENDLER Authors Info & Affiliations

SCIENCE • 9 Feb 2023 • Vol 379, Issue 6632 • p. 548

In the 9 December 2022 issue, the Research Article "Competition-level code generation with AlphaCode" (Y. Li *et al.*, p. 1092) and the accompanying Perspective, "AlphaCode and 'datadriven' programming" (J. Z. Kolter, p. 1056) describe an artificial intelligence (AI)–based system for generating code. The authors explain that the system can be used for small coding problems, such as tests for computing students, and that they are far from being useful in computing applications that include millions of lines of code, such as word processing. As we enter an era of AI where tools like AlphaCode and chat-GPT will change how tasks are performed, it is important to understand the boundaries of what they can and cannot do.

OpenAI Codex

We’ve created an improved version of OpenAI Codex, our AI system that translates natural language to code, and we are releasing it through our API in private beta starting today. Codex is the model that powers GitHub Copilot, which we built and launched in partnership with GitHub a month ago. Proficient in more than a dozen programming languages, Codex can now interpret simple commands in natural language and execute them on the user’s behalf—making it possible to build a natural language interface to existing applications. We are now inviting businesses and developers to build on top of OpenAI Codex through our API.

START USING CODEX START

▶ REWATCH LIVE DEMO

⬡ VIEW THE CODEX CHALLENGE

🗋 READ PAPER

Your AI pair programmer

GitHub Copilot uses the OpenAI Codex to suggest code and entire functions in real-time, right from your editor.

Start my free trial › Explore docs

sentiments.ts write_sql.go parse_expenses.py addresses.rb

```
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";

// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
  const response = await fetch(`http://text-processing.com/api/sentiment/`, {
    method: "POST",
    body: `text=${text}`,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  });
  const json = await response.json();
  return json.label === "pos";
}
```

Copilot

Selected tutorial videos (Attention, TNNs, Codex…)

- OpenAI Codex demo video: https://bit.ly/3xaosVt
- "ChatGPT Tutorial for Developers": https://bit.ly/40LbMC4
- "Getting Started with Codex": https://bit.ly/3RJTL2U
- "On Large Language Models for Understanding Human Language" https://bit.ly/3xaKmI5
- "Large Language Models Part 2": https://bit.ly/3JW4YeG
- "Transformers, explained": https://bit.ly/3loUIH9
- "IntelliCode Compose: Code Generation using Transformer": https://bit.ly/40FM0iF
- "Illustrated Guide to Transformers Neural Networks": https://bit.ly/3YGRmrX
- "Attention - the beating heart of ChatGPT": https://bit.ly/3JYxoVx
- "ChatGPT - How it works: Transformers & NLP 5": https://bit.ly/3DWijQx
- "Transformer Neural Networks - EXPLAINED!": https://bit.ly/3JSxqye

Selected Links & Papers

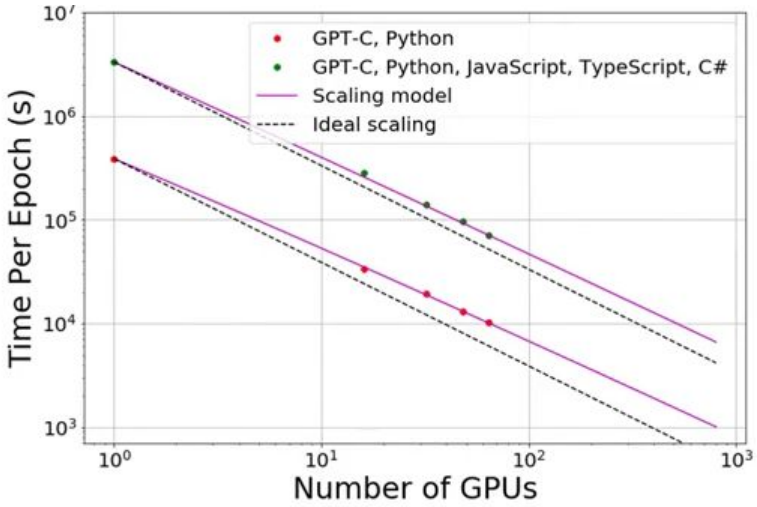
- **tl;dr:** "Transformer Neural Networks: A Step-by-Step Breakdown": https://bit.ly/3lv78YB
- "Attention Is All You Need" (2017): https://arxiv.org/pdf/1706.03762.pdf
- "IntelliCode Compose: Code Generation using Transformer" (2021): https://bit.ly/3Xtwa7W
- "Evaluating Large Language Models Trained on Code" (2021): https://bit.ly/3E7VLfK
- "Competition-Level Code Generation with AlphaCode" (2022): https://bit.ly/3lwR9cC
- GitHub Copilot documentation: https://docs.github.com/en/copilot

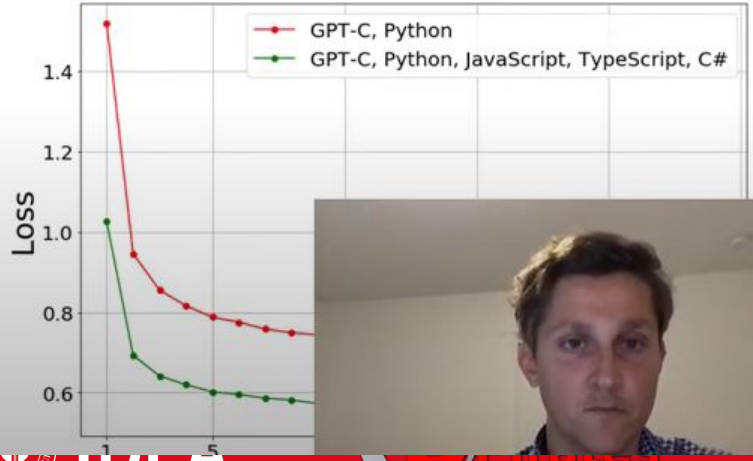
https://bit.ly/40FM0iF

Training at Scale and Distillation

- Training at scale
  - Distributed data parallel training, gradient accumulation
  - PyTorch + Horovod, ONNX runtime optimizations
  - 5 DGX-2 boxes (16 V100 GPUs, 32 GB HBM2 memory)
- ONNX runtime static graph optimizations
  - Enable constant folding, redundant operation elimination, operator fusion

	Python	C#, Python, JavaScript, TypeScript
Cumulative batch size	160	160
Time per epoch	2.8 hours	19.7 hours
Number of samples per second	163 ± 5	148 ± 5
Number of tokens per second	167000 ± 5000	152000 ± 5000





OpenAI

Understanding Codex training data and outputs

Written by Maddie Simens
Updated over a week ago

OpenAI cares deeply about developers and is committed to respecting their rights. Our hope is that Codex will lower barriers to entry and increase opportunities for beginner programmers, make expert programmers more productive, and create new code-generation tools.

The Codex model was trained on tens of millions of public repositories, which were used as training data for research purposes in the design of Codex. We believe that is an instance of transformative fair use.

The source material from those public repositories is intended to be used for these research and training purposes only; it is not intended to be included verbatim in Codex outputs. Analysis has shown that, even in this early stage of development, the vast majority of output (>99%) does not match training data. Of course, certain source material, like all computer programs, contains common, widely-used solutions that are either standard and/or functionally-mandated.

During this early, developmental stage of Codex, we continue to refine the product in numerous ways. We welcome feedback from developers, including any questions or concerns they may have about the generated output during our free beta period.

https://bit.ly/40THOvM

Code completion Limited beta

https://bit.ly/40KdrYA

Learn how to generate or manipulate code

Introduction

The Codex model series is a descendant of our GPT-3 series that's been trained on both natural language and billions of lines of code. It's most capable in Python and proficient in over a dozen languages including JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, SQL, and even Shell. During this initial limited beta period, Codex usage is free. Learn more.

You can use Codex for a variety of tasks including:

- Turn comments into code
- Complete your next line or function in context
- Bring knowledge to you, such as finding a useful library or API call for an application
- Add comments
- Rewrite code for efficiency

https://bit.ly/3RO3F3i

Codex Limited beta

The Codex models are descendants of our GPT-3 models that can understand and generate code. Their training data contains both natural language and billions of lines of public code from GitHub. Learn more.

They're most capable in Python and proficient in over a dozen languages including JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, SQL, and even Shell.

We currently offer two Codex models:

LATEST MODEL	DESCRIPTION	MAX REQUEST	TRAINING DATA
`code-davinci-002`	Most capable Codex model. Particularly good at translating natural language to code. In addition to completing code, also supports inserting completions within code.	8,000 tokens	Up to Jun 2021
`code-cushman-001`	Almost as capable as Davinci Codex, but slightly faster. This speed advantage may make it preferable for real-time applications.	Up to 2,048 tokens	

For more, visit our guide to working with Codex.

https://platform.openai.com/examples?category=code

Examples

Explore what's possible with some example applications

Search...

Code

Natural language to OpenAI API
Create code to call to the OpenAI API using a natural language instruction.

Natural language to Stripe API
Create code to call the Stripe API using natural language.

SQL translate
Translate natural language to SQL queries.

Python to natural language
Explain a piece of Python code in human understandable language.

Calculate Time Complexity
Find the time complexity of a function.

Translate programming languages
Translate from one programming language to another

Explain code
Explain a complicated piece of code.

Python bug fixer
Find and fix bugs in source code.

JavaScript helper chatbot
Message-style bot that answers JavaScript questions

JavaScript to Python
Convert simple JavaScript expressions into Python.

Write a Python docstring
An example of how to create a docstring for a given Python function. We specify the Python version, paste in the code, and then ask within a comment for a docstring, and give a characteristic beginning of a docstring (""").

JavaScript one line function
Turn a JavaScript function into a one liner.

What about R?

https://github.com/MichelNivard/gptstudio

gptstudio

lifecycle experimental | CRAN not published | codecov 59% | R-CMD-check passing

The goal of gptstudio is for R programmers to easily incorporate use of large language models (LLMs) into their project workflows. These models appear to be a step change in our use of text for knowledge work, but you should carefully consider ethical implications of using these models. Ethics of LLMs (also called Foundation Models) is an area of very active discussion.

For further addins, tailored for R developers, also see the sister package: gpttools

Install the addins from this package:

```
require(devtools)
install_github("MichelNivard/gptstudio")
```

What about R? (2)

https://jameshwade.github.io/gpttools/

gpttools 0.2.2 Reference Changelog

gpttools

The goal of gpttools is to extend gptstudio for R package developers to more easily incorporate use of large language models (LLMs) into their project workflows. These models appear to be a step change in our use of text for knowledge work, but you should carefully consider ethical implications of using these models. Ethics of LLMs (also called Foundation Models) is an area of very active discussion.

The package has four addins:

- **Comment code:** uses code-davinci-edit-001 model from OpenAI to add comments to your code with the prompt: “add comments to each line of code, explaining what the code does”
- **Add roxygen:** uses **text-davinci-003** model from OpenAI to add and fill out a roxygen skeleton to your highlight code (should be a function) with the prompt: *“insert roxygen skeleton to document this function”*
- **Convert script to function:** uses **code-davinci-edit-001** model from OpenAI to convert a highlighted script into a function with the prompt: *“convert this R code into an R function”*
- **Write a unit test for a function with testthat:** uses **text-davinci-003** model from OpenAI to suggest a unit test for a selected function with the prompt: *“Suggest a unit text for this function using the testthat package”*
- A freeform addin that let’s you specify the prompt using the “edit” functionality of ChatGPT

Cut to RStudio…

Ethical Issues…

- Copyright?
- Cheating?
- Employment?

"As we enter an era of AI where tools like AlphaCode and chatGPT will change how tasks are performed, it is important to understand the boundaries of what they can and cannot do."
-- Jim Hendler (09 Feb 2023)

Science
Current Issue First release papers Archive About
LETTER

https://bit.ly/3XDYqow

Understanding the limits of AI coding

JAMES HENDLER Authors Info & Affiliations

SCIENCE • 9 Feb 2023 • Vol 379, Issue 6632 • p. 548

In the 9 December 2022 issue, the Research Article "Competition-level code generation with AlphaCode" (Y. Li *et al.*, p. 1092) and the accompanying Perspective, "AlphaCode and 'datadriven' programming" (J. Z. Kolter, p. 1056) describe an artificial intelligence (AI)–based system for generating code. The authors explain that the system can be used for small coding problems, such as tests for computing students, and that they are far from being useful in computing applications that include millions of lines of code, such as word processing. As we enter an era of AI where tools like AlphaCode and chatGPT will change how tasks are performed, it is important to understand the boundaries of what they can and cannot do.

To make sure that code can be maintained and managed by other programmers, human developers use mnemonic variable names and embed explanatory comments. Understanding, debugging, and extending code written by other humans remains a formidable challenge—perhaps even more difficult than producing the code in the first place. In addition, many techniques are used for validation and verification, and code used in mission-critical applications, such as airline flight systems, goes through substantial quality assurance testing. AI models have yet to address the challenges of maintaining code, ensuring that users can decipher it, and subjecting programs to safety protocols.

Understanding and evaluating the limits of these techniques is crucial before they are put into real-world use. Some testing of capabilities has been applied to language generation tools (*1*, *2*), but AI coding remains a nascent field. The Technology Policy Committee of the Association for Computing Machinery recommends more investment in transparency and accountability for AI algorithms (*3*). The promise of systems like AlphaCode must be carefully balanced against the risks of their use. The interaction between AI code-generation systems and human programmers must be resolved before such systems can become an integral part of the future of computing.

What about Copyright?

https://bit.ly/3RRu0gZ

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

J. DOE 3 and J. DOE 4, individually and on behalf of all others similarly situated, Individual and Representative Plaintiffs, v. GITHUB, INC., a Delaware corporation; MICROSOFT CORPORATION, a Washington corporation; OPENAI, INC., a Delaware nonprofit corporation; OPENAI, L.P., a Delaware limited partnership; OPENAI GP, L.L.C., a Delaware limited liability company; OPENAI STARTUP FUND GP I, L.L.C., a Delaware limited liability company; OPENAI STARTUP FUND I, L.P., a Delaware limited partnership; OPENAI STARTUP FUND MANAGEMENT, LLC, a Delaware limited liability company, Defendants.	Case No. **COMPLAINT** **CLASS ACTION** **DEMAND FOR JURY TRIAL**

CLASS ACTION COMPLAINT

What about Copyright?

https://bit.ly/3RRu0gZ

Plaintiffs J. Doe 3 and J. Doe 4 ("Plaintiffs"), on behalf of themselves and all others similarly situated, bring this Class Action Complaint (the "Complaint") against Defendants GitHub, Inc.; Microsoft Corporation; OpenAI, Inc.; OpenAI, L.P.; OpenAI GP, L.L.C.; OpenAI Startup Fund GP I, L.L.C.; OpenAI Startup Fund I, L.P.; and OpenAI Startup Fund Management, LLC[1] for violation of the Digital Millennium Copyright Act, 17 U.S.C. §§ 1201–1205 (the "DMCA"); violation of the Lanham Act, 15 U.S.C. § 1125; violation of Unfair Competition law, *Cal. Bus. & Prof. Code* §§ 17200, *et seq.*; violation of the California Consumer Privacy Act, *Cal. Civ. Code* § 1798.150 (the "CCPA"); and Breach of Contract regarding the Suggested Licenses, GitHub's Privacy Statement, and GitHub's Terms of Service, *Cal. Bus. & Prof. Code* §§ 22575–22579, *Cal. Civ. Code* § 1798.150. Plaintiffs and the Class also bring this Complaint against Defendants for their Tortious Interference in Plaintiffs' Contractual Relationships; Fraud, and Negligence regarding handling of sensitive data.