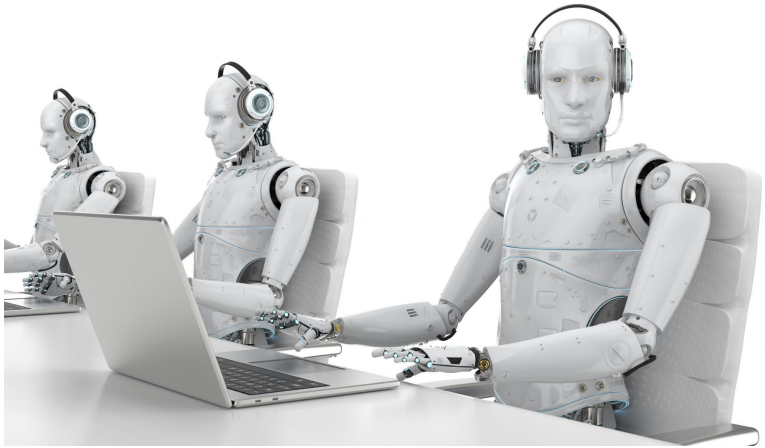
Usable Code Prediction

Overview of State-of-the-Art Models, Software Offerings and Construction of User Study

My Motivation

This Course has made me think a lot about the future of engineering, development, and deployment cycles.



Deterministic issues

Bug Detection in Syntax

Bug Detection in Purpose

Code Deployment

AI driven development?

Software Engineering as it Stands

Machine Learning, Data Science, Product Development… All disciplines have a fundamental business/research purpose that they are trying to pursue

Not sure we have a great way to encode these goals of Purpose (human discussion is messy, but it's our best solution!)

Higher Level Abstraction & Reasoning Needed

Step 1. Imagine The End Product

Step 2. Sketch out your plans

Step 3. Attempt to Implement your plans

Step 4. Adjust plans after seeing what works and what doesn't

Step 5: Repeat 3&4 Until Finished (Or Quit)

Truly Usable AI4SE

We’ve covered a lot of academic methods in this course so far, lot’s of them promising a really interesting future

I wanted to know... What is the level of impact these methods have on the average engineer?

Main idea: How much do current AI models actually help with the writing of code?

Next Token Prediction

This is directly analogous to generative language models (OpenAI GPT-2, GPT-3)

Much of what we've seen in this class is advances in long-distance dependencies and huge models pushing forward the ability to predict the next element in the sequence

These models are pre-trained on gigantic corpuses of natural language or source code

GPT-2 : "Language Models are Unsupervised Multitask Learners"

GPT-3: "Language Models are Few Shot Learners"

Deep3: "Probabilistic Models for Code with Decision Trees"

Quick Usage Explanation

I think everyone is familiar with IDE's and autocompletion…

Simply, next token prediction gives suggestions for the next written segment given the previous tokens in the sequence.

"My pen is out of _____"

Tokens = Words, Punctuation, Code Segments, Variable Names, Method Calls, etc…

Matrix = np.zeros()

For i,j in zip(data,value):

    Matrix[i,j] = … (next token)

Currently Available Tools

Anyone have Experience with these?

Early Contributions from Literature

“Probabilistic Models for Code with Decision Trees”

Introduces Abstract Syntax Trees for Domain Specific Language TGen

Trains the Deep3 Model

Introduces the Py150 dataset to be used as a benchmark

Doesn’t assess the accuracy of predicted next token, but assesses accuracy of various parts of the next-token prediction (such as: type, value, name..)

Py150 Dataset

150,000 Python Files compiled from GitHub

Training Set: 100k Files with 76M Tokens

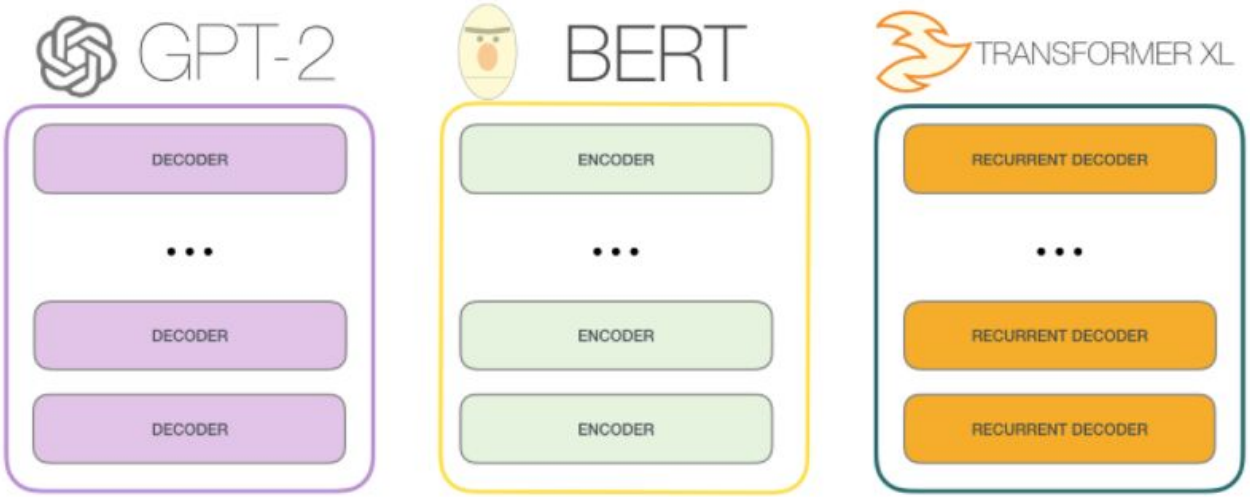
Test Set: 50k Files with same ratio

Roughly 760:1 ratio - clearly different than the pre-training set

Makes Sense bc Files larger than single functions

Plan to sample this data and understand the composition of functions, external libraries used, algo types, etc..

Transformers



We’ve hit this before…

Recap:

Encoder-Decoder Framework

Self-Attention is a major advancement that drives Transformers

Transformers have proven extremely successful in language modeling of long-distance relationships and semantic understanding

GPT-2 is a Decoder Stack

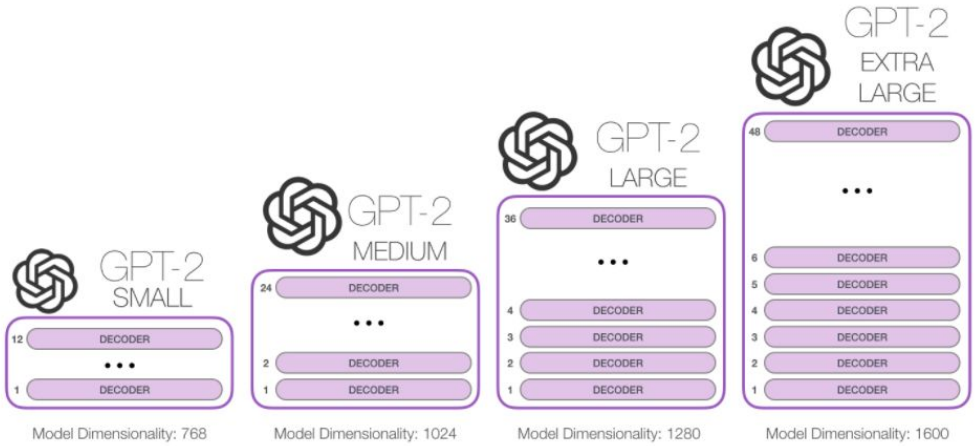
CodeGPT and GPT-2

Leading Performers on Next-Token Prediction

According to the CodeXGLUE Benchmark paper

Both of these models utilized the exact same architecture…

Essentially, GPT-2 by OpenAI but trained on CodeSearchNet dataset



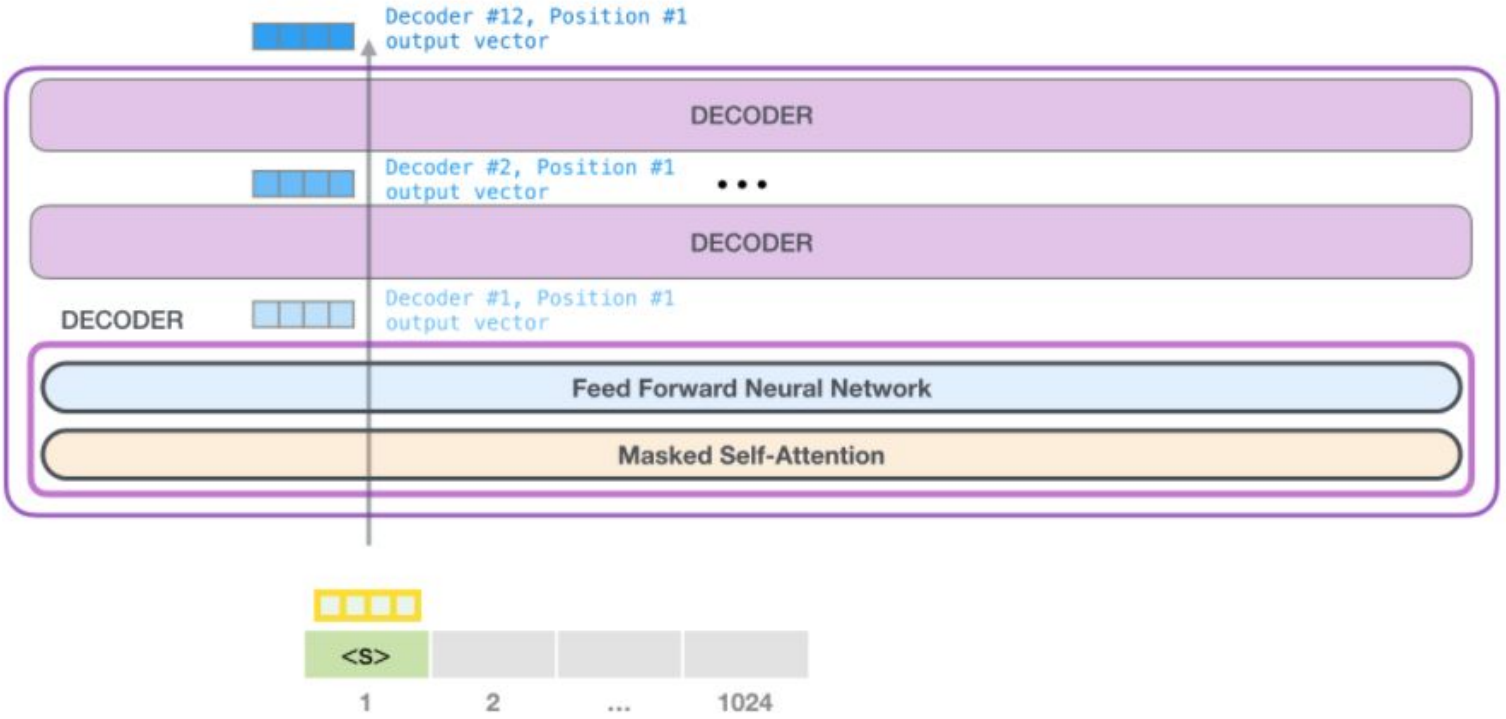
Python Focus:

1.1M python functions

119M Tokens

Roughly 110:1 ratio (seems about right?)

Decoder Stack



Evaluating Performance

GPT-2 and CodeGPT claim ~75% Accuracy

This is on the Py150 test dataset after training specifically on the 100k training set

What is this Accuracy number?

Going through the code, looks to be standard accuracy computation. I am, however, a bit skeptical of this high number?

For my study, I am going to have to dig deeper into what this academic accuracy number actually comes from.

My best guess is binary accuracy of the true token within the top N predicted tokens

I will likely move forward with a Top N Tokens approach, as a singular prediction seems like it is prone to issues

Code For Evaluation

I want to make sure I’m not fitting too strongly to my own coding style, but I am sticking with things that I am most familiar with and have developed before

I intend to utilize at least 1 example of my own code and 1 example of publicly available code on GitHub (primarily published academic work) for every category

Categories to test:

1. TensorFlow Deep Learning
2. SciKit Learn ML Pipelines
3. Pandas Data Analysis
4. Flask/Django Applications
5. Plotly Visualization

Typical Evaluation (Correctness)

A more philosophical point: Correctness itself might not be the game changer for AI Code Completion

In my characterization of the code I'm going to evaluate on, most of the tokens are set up for data (initializing) and control flow (loops)

Hypothesis: the difficulty/importance of code tokens is not uniformly distributed

I believe there are high leverage points in which we engineers have to implement the core logic that really drives the value of the business/research purpose

New Angle on Usefulness

My main goal of this project is to evaluate usefulness of these tools to the general Python engineer (approximated by me - accuracy of this is up for debate)

My goal is to select high leverage points in the code I am evaluating to see if the suggestions could help me form and think through the logic

Example: When doing Pandas Data Analysis, I often don't know if there is a pre-built function for rolling window computation. Do these tools suggest appropriate/adjacent functions that might point me in the correct direction?

Proposal: Evaluate this qualitatively on a 1-10 scale. Thoughts on Improving this?

To be continued in Project Demonstration