Advanced NLP

NLP For Program Synthesis

Zhangir Azerbayev (zhangir.azerbayev@yale.edu)
September 16, 2021

Background

Outline

Program Synthesis

“Program Synthesis is the task of automatically finding programs from the underlying programming language that satisfy user intent expressed in some form of constraints." (Gulwani et al. 2017)

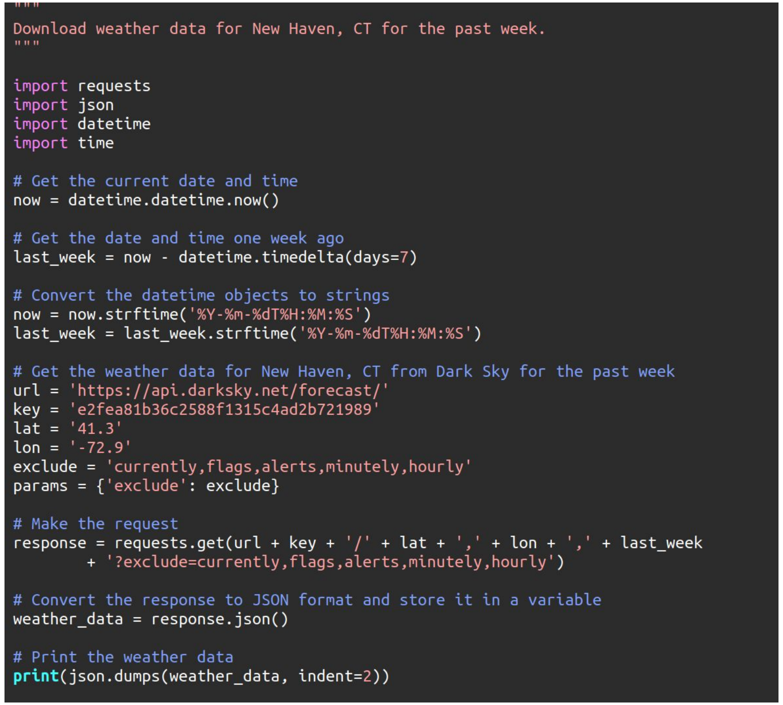
- “The holy grail of computer science” (Gulwani et al. 2017)
- Also interested in related tasks in code generation and understanding:
    - Code autocomplete, docstring generation, code search, code repair, etc.

Program Synthesis

- Deductive synthesis
  - Write a program from a specification (formal or natural language)
  - Sometimes writing a formal specification is as hard as writing a program!
- Inductive synthesis
  - Learn a program from input/output pairs
- Hybrid task is common, e.g natural language specification + test cases.

Motivation: Software Engineering

- Many programming tasks are tedious but conceptually straightforward
- Countless programmer hours are wasted looking up API calls, transforming data, etc.
- Want to free programmers to operate at higher level of abstraction.



```python
"""
Download weather data for New Haven, CT for the past week.
"""

import requests
import json
import datetime
import time

# Get the current date and time
now = datetime.datetime.now()

# Get the date and time one week ago
last_week = now - datetime.timedelta(days=7)

# Convert the datetime objects to strings
now = now.strftime('%Y-%m-%dT%H:%M:%S')
last_week = last_week.strftime('%Y-%m-%dT%H:%M:%S')

# Get the weather data for New Haven, CT from Dark Sky for the past week
url = 'https://api.darksky.net/forecast/'
key = 'e2fea81b36c2588f1315c4ad2b721989'
lat = '41.3'
lon = '-72.9'
exclude = 'currently,flags,alerts,minutely,hourly'
params = {'exclude': exclude}

# Make the request
response = requests.get(url + key + '/' + lat + ',' + lon + ',' + last_week
        + '?exclude=currently,flags,alerts,minutely,hourly')

# Convert the response to JSON format and store it in a variable
weather_data = response.json()

# Print the weather data
print(json.dumps(weather_data, indent=2))
```

Assembly -> high-level language -> NL description

Motivation: Automated Reasoning

- State-of-the-art language models struggle with basic deductive reasoning tasks
- Hypothesis: A strong code-to-text model is a strong deductive reasoning model.
- Large step towards human-level artificial intelligence.

Q: A foo and a bar are a kind of baz. John has 4 foo and 7 bar in a basket, on the way home he lost 2 foo. How many baz does he have?

A: John has 3 baz in his basket.

(GPT-3 struggles with basic reasoning)

"Everybody should learn to program a computer, because it teaches you how to think." - Steve Jobs

Pre-neural approaches

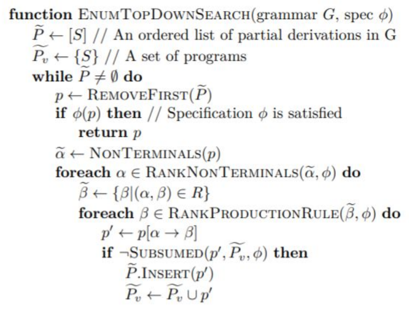
Components of a pre-neural program synthesizer:

- User Intent
  - Deductive or inductive? If deductive, how do you represent the specification? Do you tolerate ambiguity?
- Search space
  - Often have to restrict allowed operations or control structure
  - Sometimes even restrict to a model of computation (e.g regex, CFG)
- Search Technique

Main challenge: Program space grows exponentially with the length of the program. How do you efficiently search this space?

Enumerative Search

- Very simple idea: Enumerate the programs in your search space and choose the first one that works.
    - Top down tree search
    - Bottom up tree search
- Grab-bag of methods to cleverly prune the search tree
- Approach is completely unscalable



```
function EnumTopDownSearch(grammar $G$, spec $\phi$)
    $\widetilde{P} \leftarrow [S]$ // An ordered list of partial derivations in G
    $\widetilde{P_v} \leftarrow \{S\}$ // A set of programs
    while $\widetilde{P} \neq \emptyset$ do
        $p \leftarrow$ RemoveFirst($\widetilde{P}$)
        if $\phi(p)$ then // Specification $\phi$ is satisfied
            return $p$
        $\widetilde{\alpha} \leftarrow$ NonTerminals($p$)
        foreach $\alpha \in$ RankNonTerminals($\widetilde{\alpha}, \phi$) do
            $\widetilde{\beta} \leftarrow \{\beta | (\alpha, \beta) \in R\}$
            foreach $\beta \in$ RankProductionRule($\widetilde{\beta}, \phi$) do
                $p' \leftarrow p[\alpha \rightarrow \beta]$
                if $\neg$Subsumed($p', \widetilde{P_v}, \phi$) then
                    $\widetilde{P}$.Insert($p'$)
                    $\widetilde{P_v} \leftarrow \widetilde{P_v} \cup p'$
```

Figure 4.2: A simple top-down enumeration algorithm to search for a derivation p in a hypothesis space defined by a CFG G that satisfies a given specification ϕ.

Constraint Solving

- Constraint Solving: find variables that make a formula return true
  - E.g Boolean satisfiability problem
- Pipeline:
  - Create one-to-one mapping between search space and formulas
  - Use off-the-shelf SAT/SMT solver to find variables satisfying formula.

Stochastic Search: MCMC

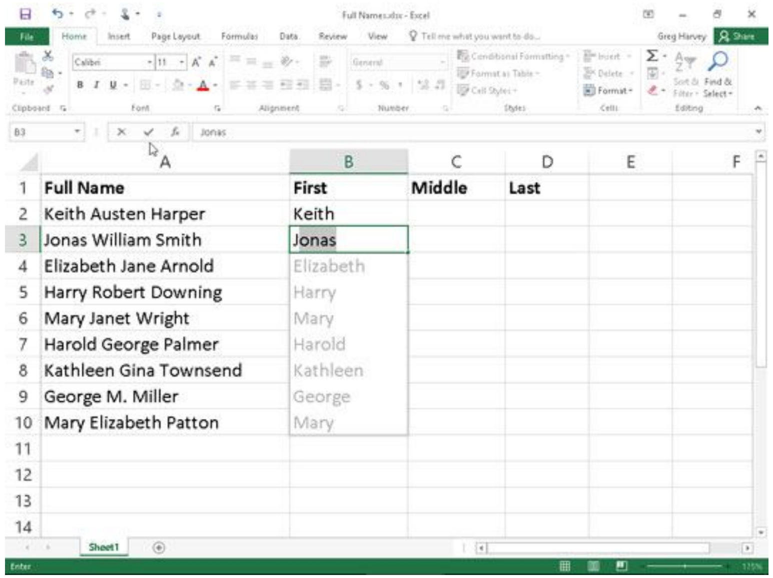
MCMC: Sample from a probability distribution by simulating a Markov Chain

1. Represent search space as a graph, where an edge connects two programs if their edit distance is 1
2. Give each program *e* a score $\text{Score}(e)$
   a. Common choice is $\exp\left(-\frac{1}{2}C(e)\right)$ where *C(e)* is the proportion of test cases passed
3. Perform probabilistic walk on graph according to $\text{Score}(e)$

Performing many iterations samples from the search space with $P(e) \propto Score(e)$

Microsoft Flashfill

- Introduced in 2012
- Fast inductive program synthesis using regular expressions
    - Limited model of computation



	A	B	C	D
1	Full Name	First	Middle	Last
2	Keith Austen Harper	Keith		
3	Jonas William Smith	Jonas		
4	Elizabeth Jane Arnold	Elizabeth		
5	Harry Robert Downing	Harry		
6	Mary Janet Wright	Mary		
7	Harold George Palmer	Harold		
8	Kathleen Gina Townsend	Kathleen		
9	George M. Miller	George		
10	Mary Elizabeth Patton	Mary		
11				
12				
13				
14				

ACM ICFP Program Synthesis Competition, 2013

- ICFP Program Synthesis Competition, 2013
    - Synthesize programs in LambdaBV, a highly simplified functional language that operates on 64 bit binary vectors
    - Winner used enumerative search with grab-bag of pruning methods
    - Enumerated ~100 million programs per question
    - Prediction takes ~5 minutes on 32 cores

Takeaway: Unscalable approach even on highly simplified problems.

Pre-neural approaches: Text-to-code

- Text to code looked impossible with pre-neural approaches
- Challenges:
  - How to parse natural language description into specification?
  - Text-to-code places no constraints on search space

Large language model revolution

General and scalable deep learning architectures have revolutionized program synthesis.

- February 2020: CodeBERT
  - SOTA code search, docstring generation, etc.
- May 2020: GPT-3 can write simple programs
- June 2021: Github Copilot
  - Code completion engine
- August 2021: OpenAI Codex (Chen et al. 2021)
  - GPT-like model trained on source code
- August 2021: Program Synthesis with Large Language Models (Austin et al. 2021)
  - Another GPT-like model trained on source code

Why so much recent progress?

- Leveraging data: previous approaches made little use of ML
- Compute
  - Well capitalized institutes such as OpenAI and Google became interested in the problem
- Replacing program search with Sequence modelling
  - Not top-down or bottom-up, but left-to-right.

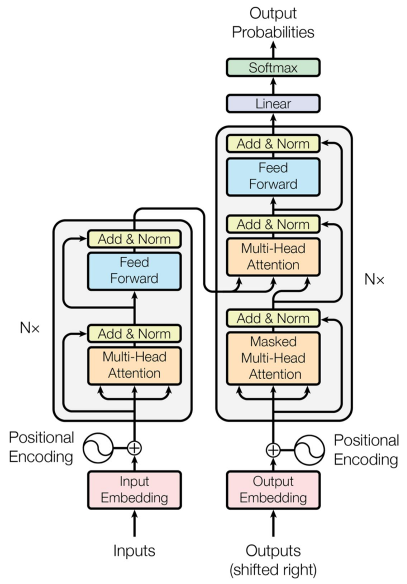


Figure 1: The Transformer - model architecture.

Papers

Main Papers:

- Program Synthesis with Large Language Models (Austin et al. 2021)
- Evaluating Large Language Models Trained on Code (Chen et al. 2021)

Additional Papers:

- Measuring Coding Challenge Competence with APPS (Hendrycks et al. 2021)
- CodeBERT: A Pre-Trained Model for Programming and Natural Languages (Feng et al. 2020)
- CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation (Lu et al. 2021)

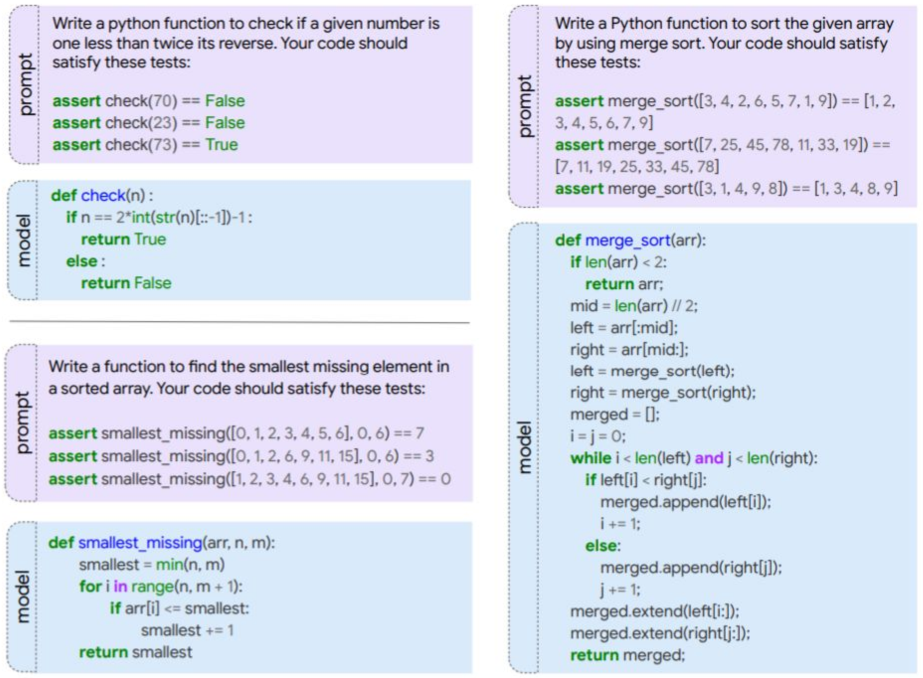
Tasks and Benchmarks

Program Synthesis

- Generate program from natural language description and/or test cases.
- Most interesting and general task.
- Benchmarks:
  - HumanEval (Chen et al.)
  - MBPP (Austin et al.)
  - MathQA-Python (Austin et al.)
  - APPS (Hendrycks et al.)

MBPP

- Entry-level programming tasks
- Provides prompt, solutions, and test cases
- ~1000 examples
- Source of difficulty is mostly control flow.


prompt
model
prompt
model
prompt
model

Write a python function to check if a given number is one less than twice its reverse. Your code should satisfy these tests:

```
assert check(70) == False
assert check(23) == False
assert check(73) == True
```

```
def check(n) :
  if n == 2*int(str(n)[::-1])-1 :
    return True
  else :
    return False
```

Write a function to find the smallest missing element in a sorted array. Your code should satisfy these tests:

```
assert smallest_missing([0, 1, 2, 3, 4, 5, 6], 0, 6) == 7
assert smallest_missing([0, 1, 2, 6, 9, 11, 15], 0, 6) == 3
assert smallest_missing([1, 2, 3, 4, 6, 9, 11, 15], 0, 7) == 0
```

```
def smallest_missing(arr, n, m):
    smallest = min(n, m)
    for i in range(n, m + 1):
        if arr[i] <= smallest:
            smallest += 1
    return smallest
```

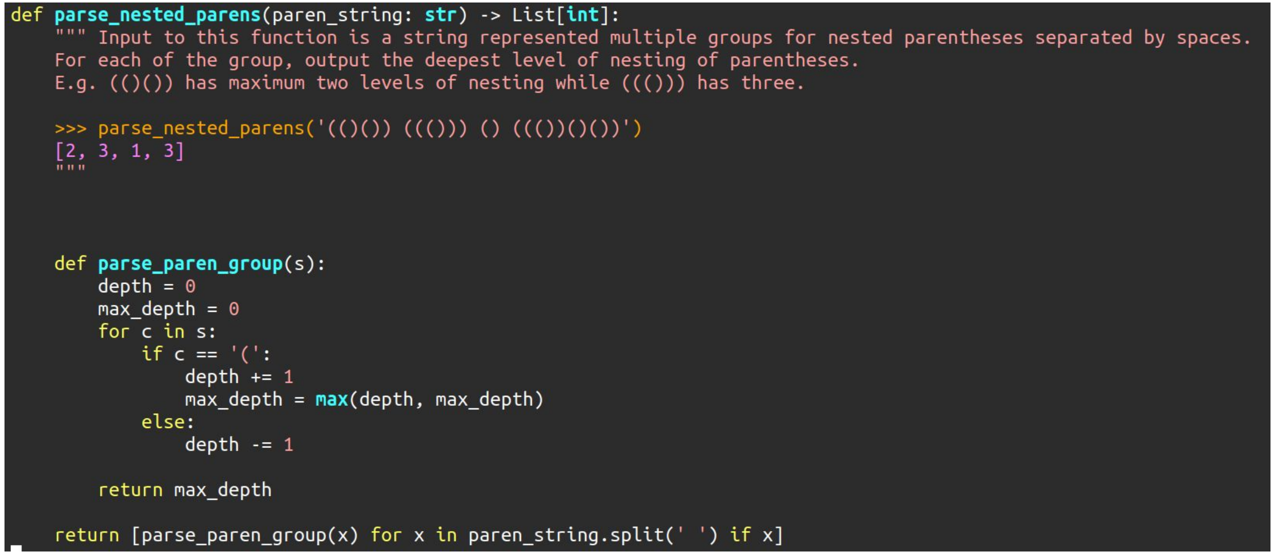
Write a Python function to sort the given array by using merge sort. Your code should satisfy these tests:

```
assert merge_sort([3, 4, 2, 6, 5, 7, 1, 9]) == [1, 2, 3, 4, 5, 6, 7, 9]
assert merge_sort([7, 25, 45, 78, 11, 33, 19]) == [7, 11, 19, 25, 33, 45, 78]
assert merge_sort([3, 1, 4, 9, 8]) == [1, 3, 4, 8, 9]
```

```
def merge_sort(arr):
  if len(arr) < 2:
    return arr;
  mid = len(arr) // 2;
  left = arr[:mid];
  right = arr[mid:];
  left = merge_sort(left);
  right = merge_sort(right);
  merged = [];
  i = j = 0;
  while i < len(left) and j < len(right):
    if left[i] < right[j]:
      merged.append(left[i]);
      i += 1;
    else:
      merged.append(right[j]);
      j += 1;
  merged.extend(left[i:]);
  merged.extend(right[j:]);
  return merged;
```

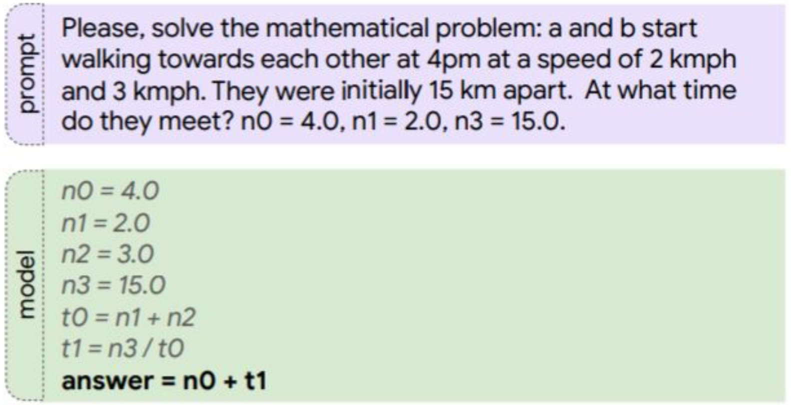
HumanEval

- Similar to MBPP
- 164 examples


```python
def parse_nested_parens(paren_string: str) -> List[int]:
    """ Input to this function is a string represented multiple groups for nested parentheses separated by spaces.
    For each of the group, output the deepest level of nesting of parentheses.
    E.g. (()()) has maximum two levels of nesting while ((())) has three.

    >>> parse_nested_parens('(()()) ((())) () ((())()())')
    [2, 3, 1, 3]
    """



    def parse_paren_group(s):
        depth = 0
        max_depth = 0
        for c in s:
            if c == '(':
                depth += 1
                max_depth = max(depth, max_depth)
            else:
                depth -= 1

        return max_depth

    return [parse_paren_group(x) for x in paren_string.split(' ') if x]
```

MathQA-Python

- Math word problems
- Prompt, solution, and test cases
- ~25,000 examples
- Source of difficulty is complex natural language prompts.

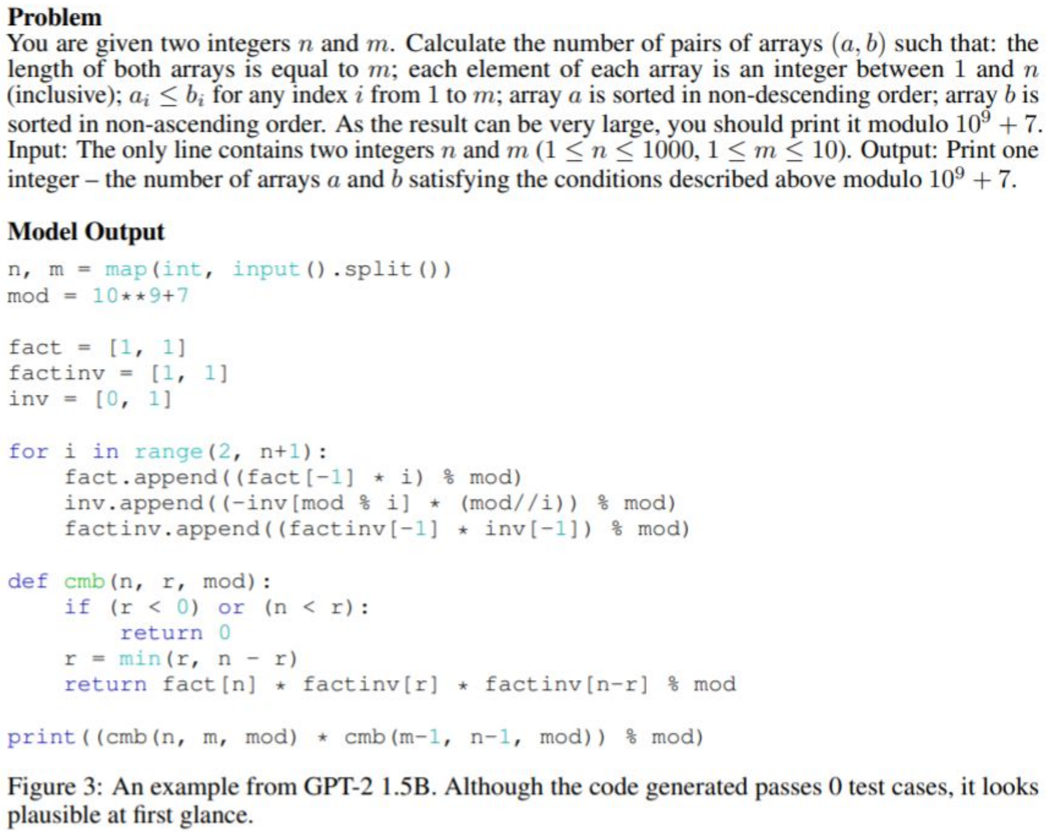

prompt
model

Please, solve the mathematical problem: a and b start walking towards each other at 4pm at a speed of 2 kmph and 3 kmph. They were initially 15 km apart. At what time do they meet? n0 = 4.0, n1 = 2.0, n3 = 15.0.

```
n0 = 4.0
n1 = 2.0
n2 = 3.0
n3 = 15.0
t0 = n1 + n2
t1 = n3 / t0
answer = n0 + t1
```

APPS

- Most difficult program synthesis benchmark
- Questions come in three levels of difficulty:
  - ~4000 Introductory
  - ~5000 Interview
  - ~1500 Competition



Problem
You are given two integers n and m. Calculate the number of pairs of arrays (a, b) such that: the length of both arrays is equal to m; each element of each array is an integer between 1 and n (inclusive); $a_i \leq b_i$ for any index i from 1 to m; array a is sorted in non-descending order; array b is sorted in non-ascending order. As the result can be very large, you should print it modulo $10^9 + 7$. Input: The only line contains two integers n and m ($1 \leq n \leq 1000$, $1 \leq m \leq 10$). Output: Print one integer – the number of arrays a and b satisfying the conditions described above modulo $10^9 + 7$.

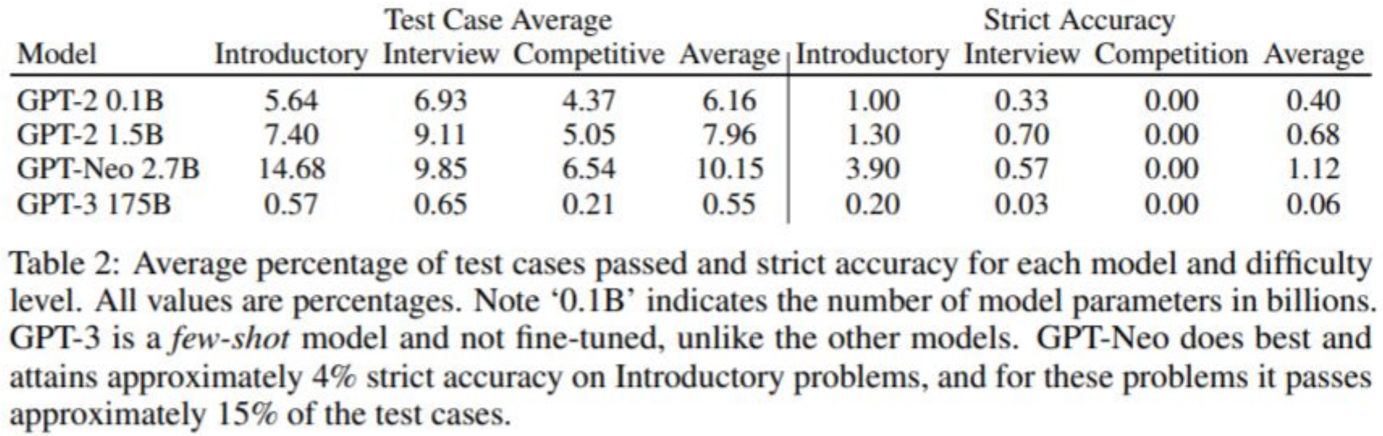
Model Output

```
n, m = map(int, input().split())
mod = 10**9+7

fact = [1, 1]
factinv = [1, 1]
inv = [0, 1]

for i in range(2, n+1):
    fact.append((fact[-1] * i) % mod)
    inv.append((-inv[mod % i] * (mod//i)) % mod)
    factinv.append((factinv[-1] * inv[-1]) % mod)

def cmb(n, r, mod):
    if (r < 0) or (n < r):
        return 0
    r = min(r, n - r)
    return fact[n] * factinv[r] * factinv[n-r] % mod

print((cmb(n, m, mod) * cmb(m-1, n-1, mod)) % mod)
```

Figure 3: An example from GPT-2 1.5B. Although the code generated passes 0 test cases, it looks plausible at first glance.

(Models finetuned on ~30GB of Github data)

APPS



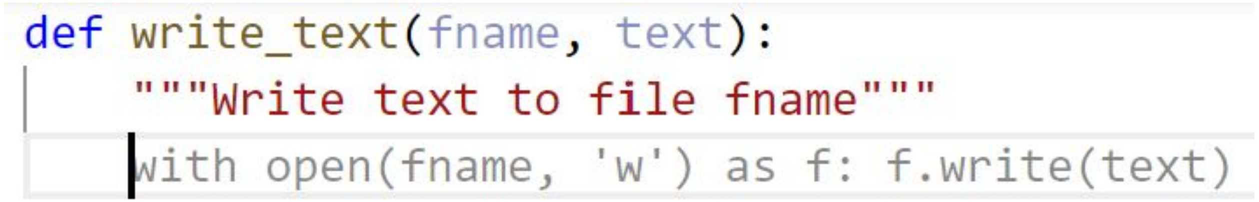
	Test Case Average				Strict Accuracy			
Model	Introductory	Interview	Competitive	Average	Introductory	Interview	Competition	Average
GPT-2 0.1B	5.64	6.93	4.37	6.16	1.00	0.33	0.00	0.40
GPT-2 1.5B	7.40	9.11	5.05	7.96	1.30	0.70	0.00	0.68
GPT-Neo 2.7B	14.68	9.85	6.54	10.15	3.90	0.57	0.00	1.12
GPT-3 175B	0.57	0.65	0.21	0.55	0.20	0.03	0.00	0.06

Table 2: Average percentage of test cases passed and strict accuracy for each model and difficulty level. All values are percentages. Note '0.1B' indicates the number of model parameters in billions. GPT-3 is a *few-shot* model and not fine-tuned, unlike the other models. GPT-Neo does best and attains approximately 4% strict accuracy on Introductory problems, and for these problems it passes approximately 15% of the test cases.

- Takeaway: APPS is really hard!
- APPS Competition is the level of human experts

Code completion

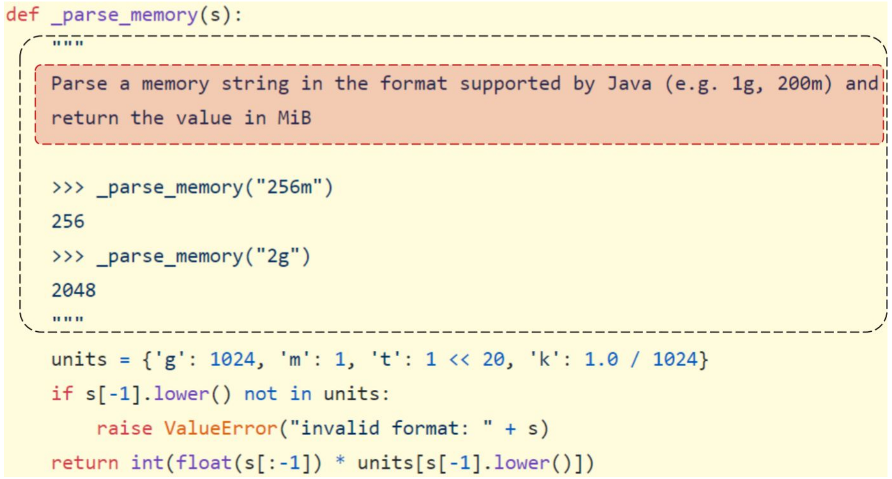
- Applications in software engineering, e.g Github Copilot.
- Self-supervised training on source code, e.g all of Github.
- PY150



```
def write_text(fname, text):
    """Write text to file fname"""
    with open(fname, 'w') as f: f.write(text)
```

Docstring generation

- Given a function body, generate a docstring.
- CodeSearchNet is a common benchmark
    - 2 million comment/body pairs
- Evaluated using BLEU score.



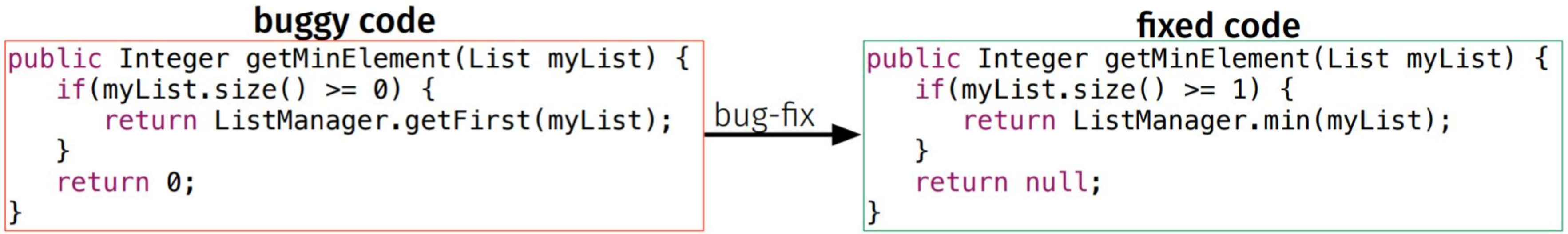
```
def _parse_memory(s):
    """
    Parse a memory string in the format supported by Java (e.g. 1g, 200m) and
    return the value in MiB

    >>> _parse_memory("256m")
    256
    >>> _parse_memory("2g")
    2048
    """
    units = {'g': 1024, 'm': 1, 't': 1 << 20, 'k': 1.0 / 1024}
    if s[-1].lower() not in units:
        raise ValueError("invalid format: " + s)
    return int(float(s[:-1]) * units[s[-1].lower()])
```

Semantic Code Search

- Given a docstring, retrieve matching code from a large database
- Common benchmark is CodeSearchNet + distractor codes

Code repair

- Fix runtime and semantic errors in code
- Bugs2Fix dataset (Java)
    - Created by mining Github commits
    - ~100k examples

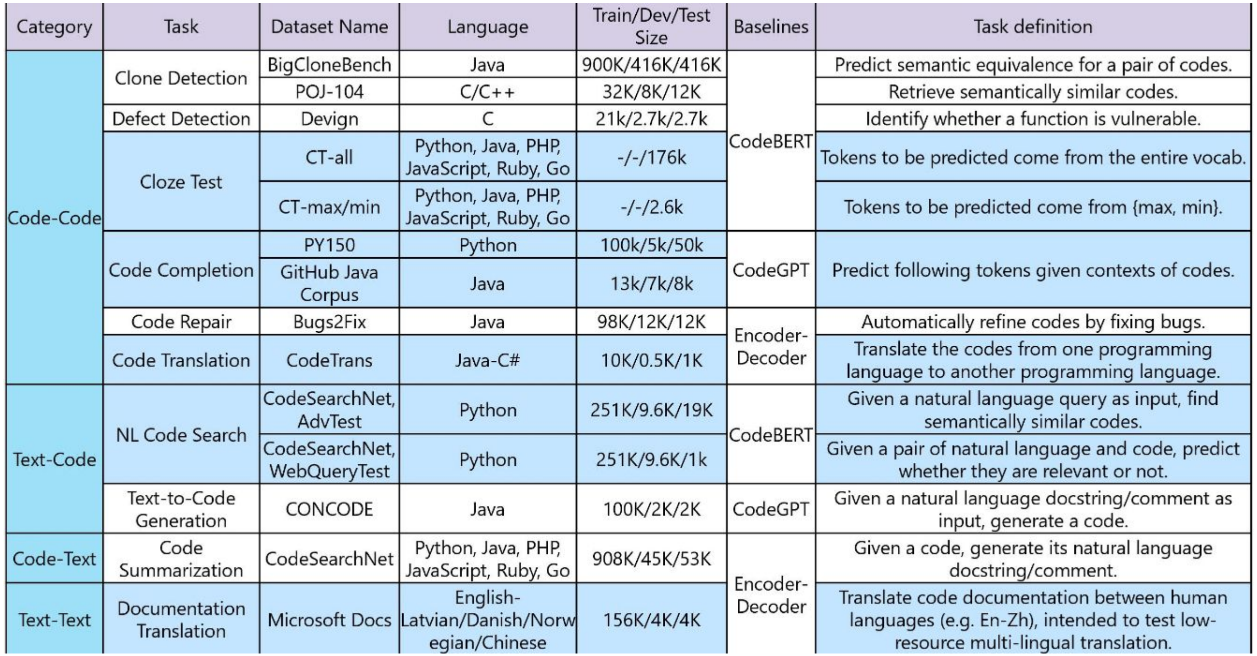


```java
public Integer getMinElement(List myList) {
   if(myList.size() >= 0) {
      return ListManager.getFirst(myList);
   }
   return 0;
}
```

```java
public Integer getMinElement(List myList) {
   if(myList.size() >= 1) {
      return ListManager.min(myList);
   }
   return null;
}
```

CodeXGlue

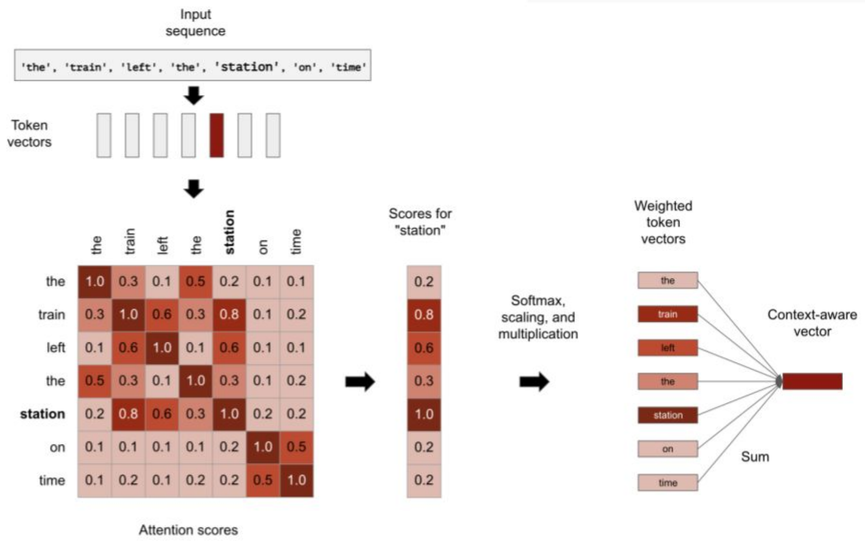
- Suite of Code generation and understanding tasks similar in spirit to GLUE benchmark for NLP.
- Repo



Category	Task	Dataset Name	Language	Train/Dev/Test Size	Baselines	Task definition
Code-Code	Clone Detection	BigCloneBench	Java	900K/416K/416K	CodeBERT	Predict semantic equivalence for a pair of codes.
		POJ-104	C/C++	32K/8K/12K		Retrieve semantically similar codes.
	Defect Detection	Devign	C	21k/2.7k/2.7k		Identify whether a function is vulnerable.
	Cloze Test	CT-all	Python, Java, PHP, JavaScript, Ruby, Go	-/-/176k		Tokens to be predicted come from the entire vocab.
		CT-max/min	Python, Java, PHP, JavaScript, Ruby, Go	-/-/2.6k		Tokens to be predicted come from {max, min}.
	Code Completion	PY150	Python	100k/5k/50k	CodeGPT	Predict following tokens given contexts of codes.
		GitHub Java Corpus	Java	13k/7k/8k		
	Code Repair	Bugs2Fix	Java	98K/12K/12K	Encoder-Decoder	Automatically refine codes by fixing bugs.
	Code Translation	CodeTrans	Java-C#	10K/0.5K/1K		Translate the codes from one programming language to another programming language.
Text-Code	NL Code Search	CodeSearchNet, AdvTest	Python	251K/9.6K/19K	CodeBERT	Given a natural language query as input, find semantically similar codes.
		CodeSearchNet, WebQueryTest	Python	251K/9.6K/1k		Given a pair of natural language and code, predict whether they are relevant or not.
	Text-to-Code Generation	CONCODE	Java	100K/2K/2K	CodeGPT	Given a natural language docstring/comment as input, generate a code.
Code-Text	Code Summarization	CodeSearchNet	Python, Java, PHP, JavaScript, Ruby, Go	908K/45K/53K	Encoder-Decoder	Given a code, generate its natural language docstring/comment.
Text-Text	Documentation Translation	Microsoft Docs	English-Latvian/Danish/Norwegian/Chinese	156K/4K/4K		Translate code documentation between human languages (e.g. En-Zh), intended to test low-resource multi-lingual translation.

Large Language Models

Transformer Architecture



	the	train	left	the	station	on	time
the	1.0	0.3	0.1	0.5	0.2	0.1	0.1
train	0.3	1.0	0.6	0.3	0.8	0.1	0.2
left	0.1	0.6	1.0	0.1	0.6	0.1	0.1
the	0.5	0.3	0.1	1.0	0.3	0.1	0.2
station	0.2	0.8	0.6	0.3	1.0	0.2	0.2
on	0.1	0.1	0.1	0.1	0.2	1.0	0.5
time	0.1	0.2	0.1	0.2	0.2	0.5	1.0

Scores for "station"
0.2
0.8
0.6
0.3
1.0
0.2
0.2

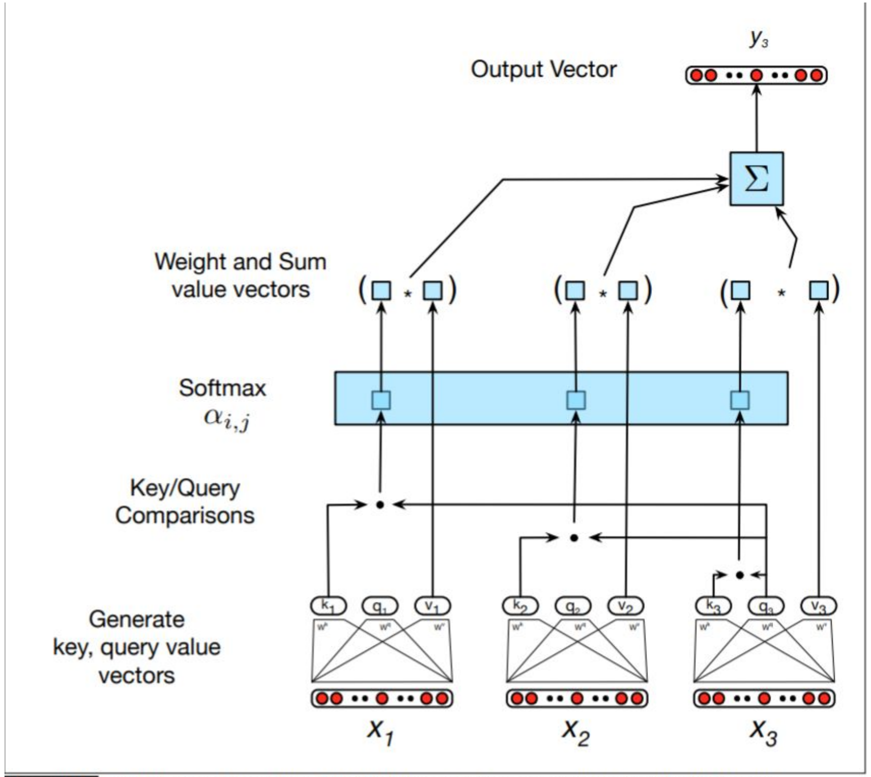


Figure 9.16 Calculation of the value of the third element of a sequence using causal self-attention.

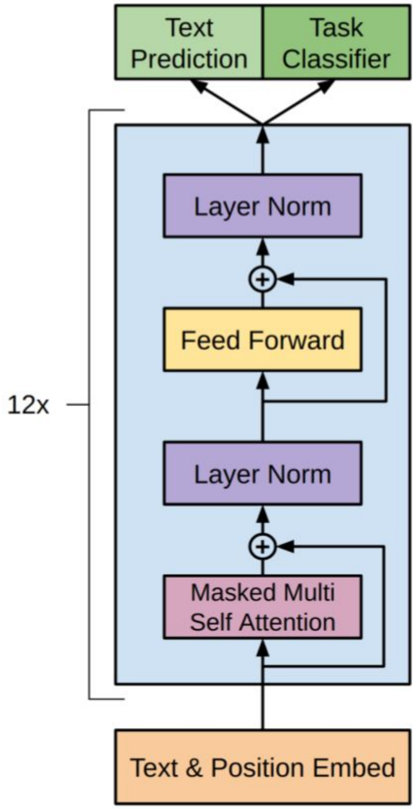
Visualizations of Transformer Block

GPT Architecture (Left-to-right)

- Left-to-right transformer
- Autoregressive next token prediction objective

GPT-3 (2020):

- 175 Billion parameters
- Few-shot learning on common NLP tasks

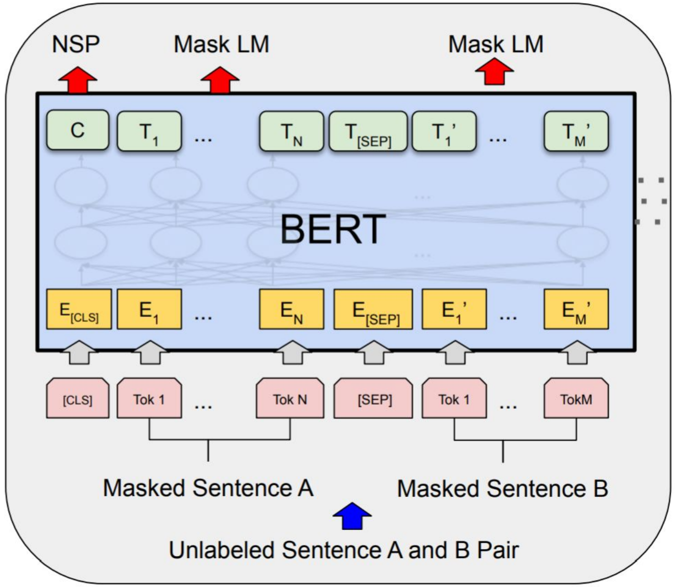


BERT Architecture (Bidirectional)

- Deep Bidirectional architecture
- Two training objectives:
  - Masked LM
  - Next sentence prediction

BERT Large (2018):

- 340 million parameters
- Achieved SOTA in wide variety of NLP tasks



GPT For Code Generation

- Left-to-right transformer directly applicable to text-to-code program synthesis
- Two significant papers:
  - Program Synthesis with Large Language Models, Austin et al. 2021
  - OpenAI Codex, Chen et al. 2021

OpenAI Codex Model

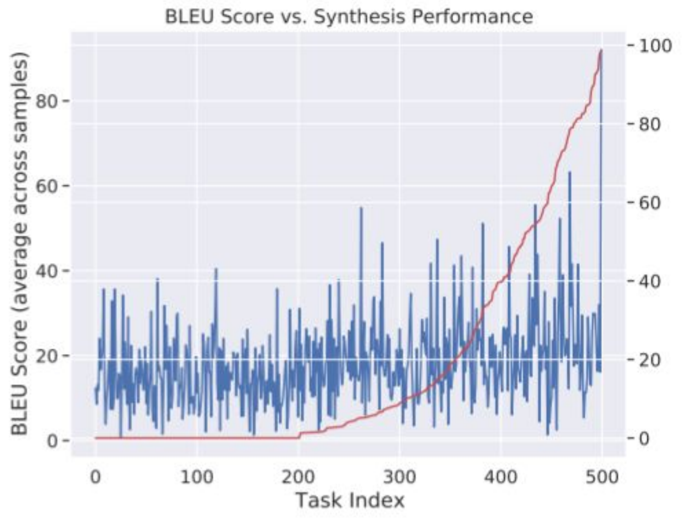
- GPT-3 Architecture with up to 12B parameters
- Trained on all of Github source code
- Usable in zero-shot, few-shot and fine-tuned settings

Austin et al. Model

- GPT-like architecture
- Train 137B parameter model
- Trained on web crawl data
    - *No source code*
- Evaluated in zero-shot, few-shot and fine-tuned setting

Evaluating Code Generation

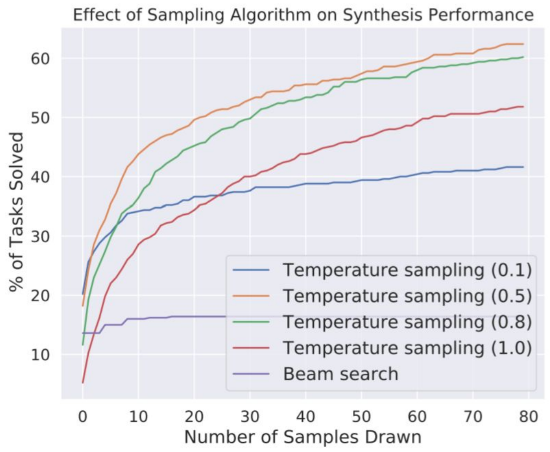
- How to evaluate program synthesis?
  - Match to reference solution (e.g BLEU)
  - Function correctness (test cases)
- Matching reference solution comes with many problems
  - Identifier renaming
  - Keywords are more important than other tokens
  - Rich structure of semantic equivalence in program space
- Use functional correctness



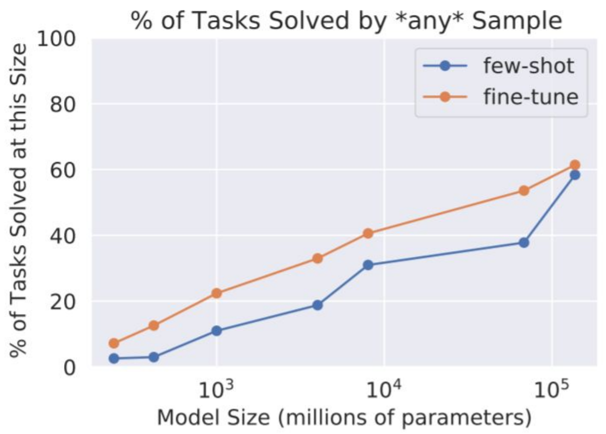
BLEU Score and synthesis performance are not strongly correlated (Austin et al.)

Sampling

- GPT-like models can generate multiple completions
- How to choose from multiple samples?
  - If you don’t have test cases: likelihood
  - If you have test cases: % of test cases passed
- pass@k metric: Generate k samples, what proportion pass all the test cases?
- Model answers question if at least one sample passes test cases



Austin et al. Results: MBPP



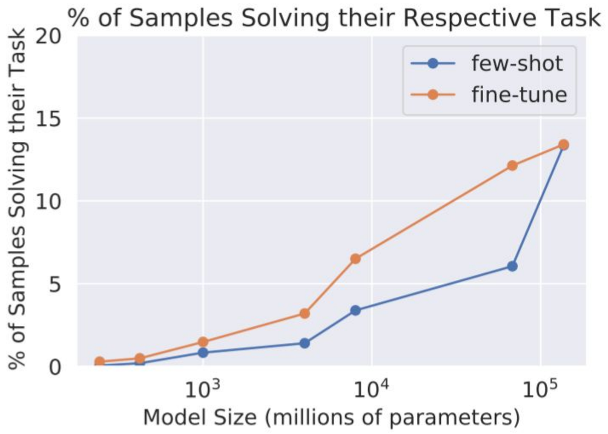


Figure 3: Performance vs model size, measured in two ways. (Left) Fraction of programs solved by *any sample* as model size is increased. This metric improves predictably as model size is increased, and fine-tuning gives a roughly constant improvement over few-shot prompting. The slope of the line shows no signs of decreasing for our largest models, which suggests that further performance gains can be had by making the model larger. (Right) Total fraction of sampled programs that solve a task, as model size is increased.

Austin et al. Results: MathQA

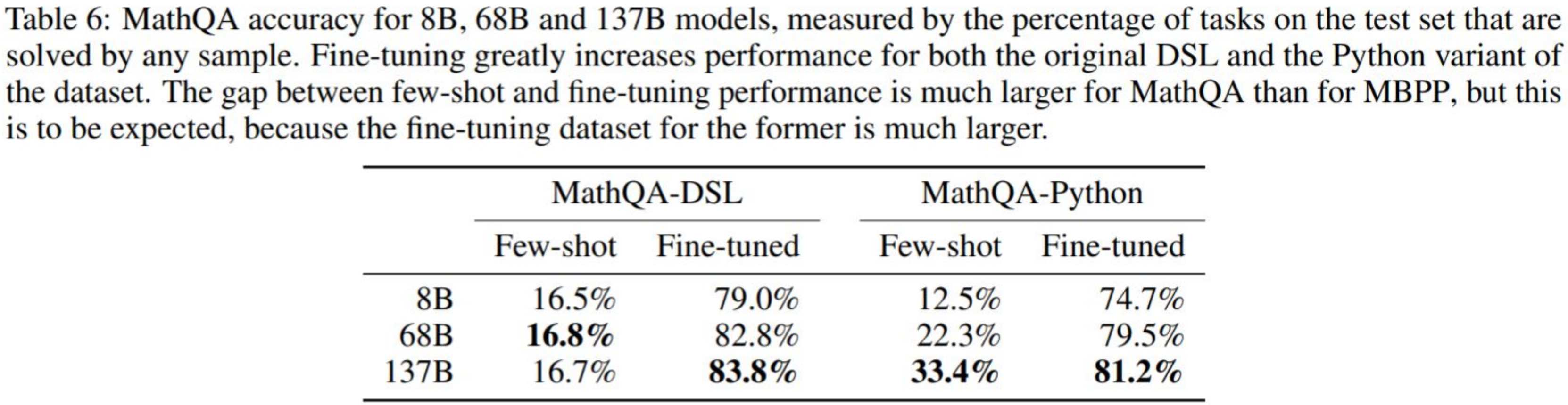
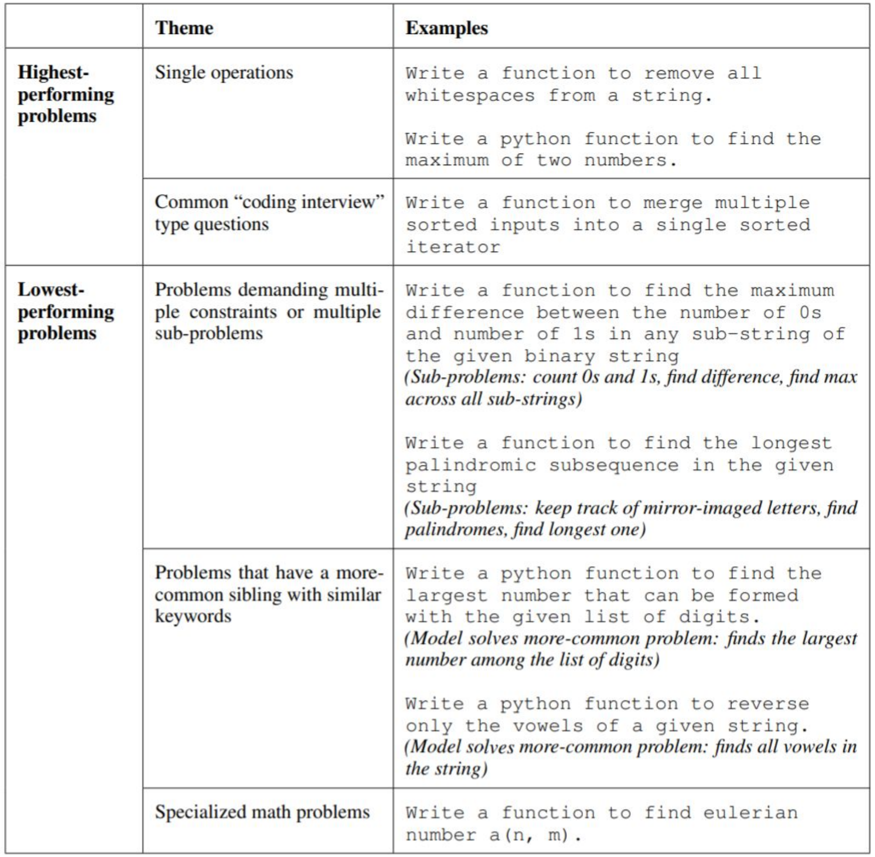


Table 6: MathQA accuracy for 8B, 68B and 137B models, measured by the percentage of tasks on the test set that are solved by any sample. Fine-tuning greatly increases performance for both the original DSL and the Python variant of the dataset. The gap between few-shot and fine-tuning performance is much larger for MathQA than for MBPP, but this is to be expected, because the fine-tuning dataset for the former is much larger.

	MathQA-DSL		MathQA-Python	
	Few-shot	Fine-tuned	Few-shot	Fine-tuned
8B	16.5%	79.0%	12.5%	74.7%
68B	**16.8%**	82.8%	22.3%	79.5%
137B	16.7%	**83.8%**	**33.4%**	**81.2%**

Austin et al. Error Analysis

Table 4: Qualitative analysis of highest- and lowest-performing problems



	Theme	Examples

Highest-performing problems	Single operations	`Write a function to remove all whitespaces from a string.`
		
		`Write a python function to find the maximum of two numbers.`

	Common "coding interview" type questions	`Write a function to merge multiple sorted inputs into a single sorted iterator`

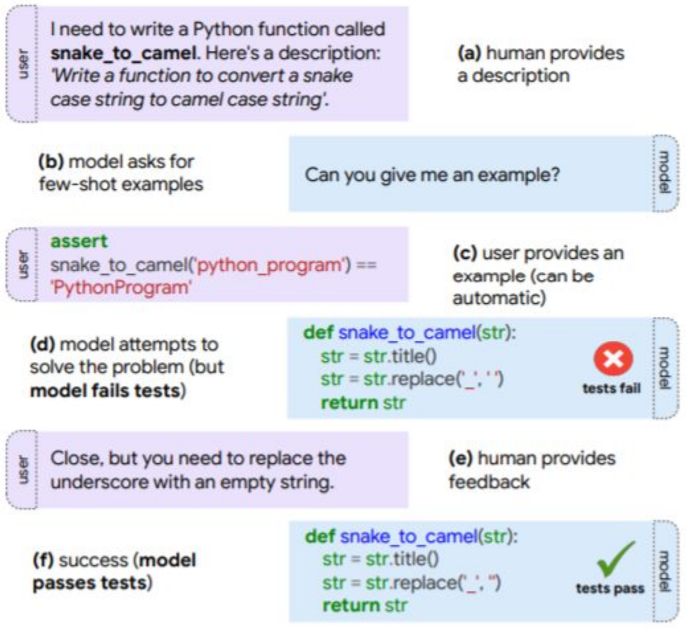
Lowest-performing problems	Problems demanding multiple constraints or multiple sub-problems	`Write a function to find the maximum difference between the number of 0s and number of 1s in any sub-string of the given binary string`
		(Sub-problems: count 0s and 1s, find difference, find max across all sub-strings)
		
		`Write a function to find the longest palindromic subsequence in the given string`
		(Sub-problems: keep track of mirror-imaged letters, find palindromes, find longest one)

	Problems that have a more-common sibling with similar keywords	`Write a python function to find the largest number that can be formed with the given list of digits.`
		(Model solves more-common problem: finds the largest number among the list of digits)
		
		`Write a python function to reverse only the vowels of a given string.`
		(Model solves more-common problem: finds all vowels in the string)

	Specialized math problems	`Write a function to find eulerian number a(n, m).`

Austin et al: Human-Computer Interaction

- Language model can function as dialogue system
  - With no extra training!
- System can ask for examples, respond to feedback


user
I need to write a Python function called **snake_to_camel**. Here's a description: *'Write a function to convert a snake case string to camel case string'.*
(a) human provides a description
(b) model asks for few-shot examples
Can you give me an example?
model
user
(c) user provides an example (can be automatic)
(d) model attempts to solve the problem (but **model fails tests**)
tests fail
model
user
Close, but you need to replace the underscore with an empty string.
(e) human provides feedback
(f) success (**model passes tests**)
tests pass
model

```
assert snake_to_camel('python_program') == 'PythonProgram'
```

```
def snake_to_camel(str):
    str = str.title()
    str = str.replace('_', ' ')
    return str
```

```
def snake_to_camel(str):
    str = str.title()
    str = str.replace('_', '')
    return str
```

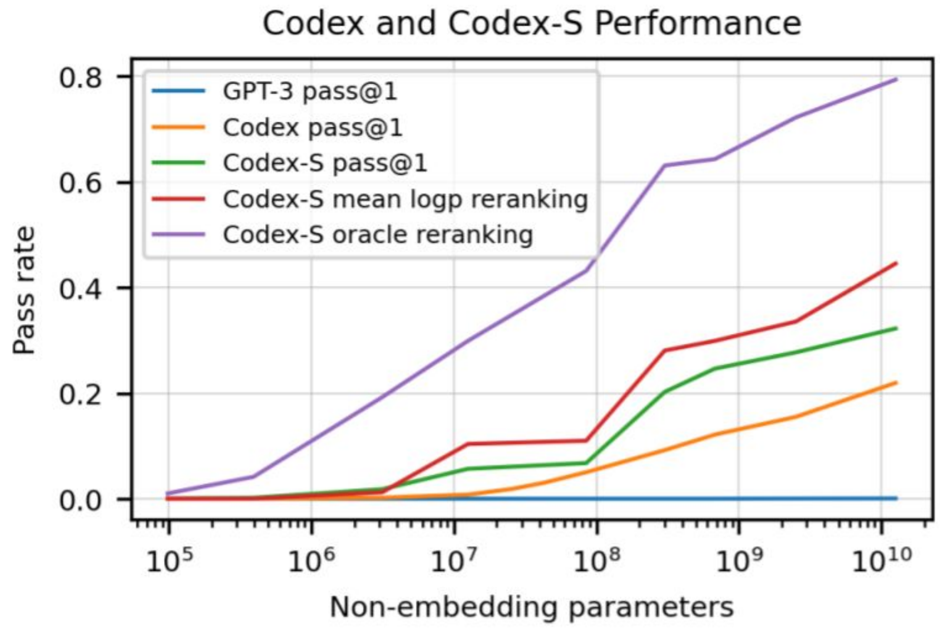
Figure 12: An overview of the "flow" of the human-model collaboration experiments. The human gives a description of the desired program and then guides the model toward the correct solution via dialog.

Austin et al: Human-Computer Interaction

- Performance:
  - Four dialogues bring MBPP pass@1 accuracy from 30% to 65%
- Clarifies ambiguous prompts
- One dialog turn corrects simple errors
  - Forgotten imports
  - Variable misuse, etc.
- Failure Mode:
  - Model can forget context
  - Attention scales quadratically: hard to process long context

Codex Results (HumanEval)

- Codex-S: fine-tuned on HumanEval

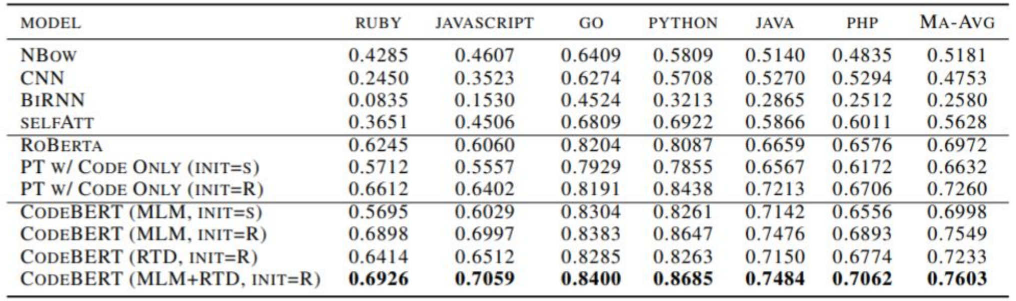


CodeBERT

- Same architecture as RoBERTA-base
  - 125M parameters
- Trained on text, source-code, and docstring-body pairs
- Two training objects
  - Masked LM Objective
  - Replaced token detection

CodeBERT Results: Code Search

- SOTA on semantic code search



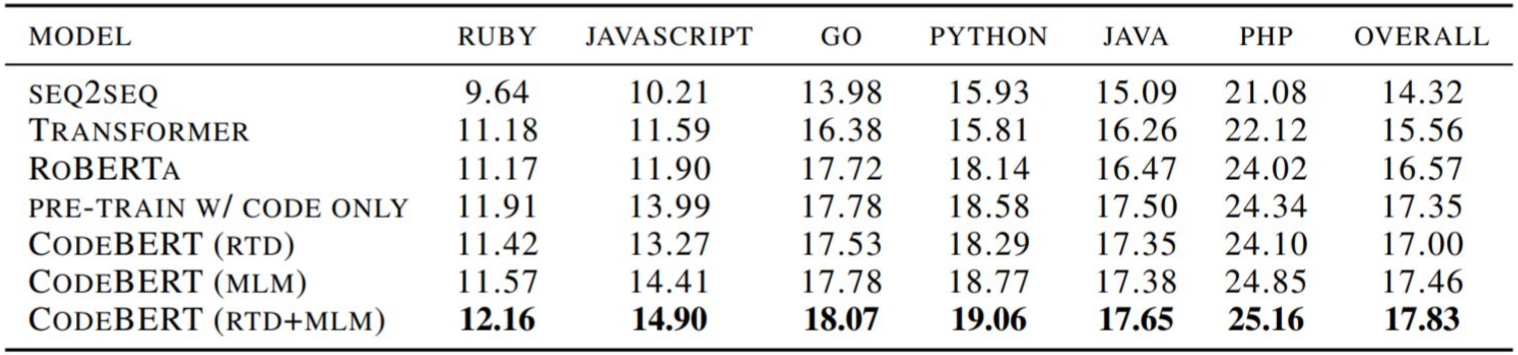
MODEL	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	MA-AVG
NBOW	0.4285	0.4607	0.6409	0.5809	0.5140	0.4835	0.5181
CNN	0.2450	0.3523	0.6274	0.5708	0.5270	0.5294	0.4753
BIRNN	0.0835	0.1530	0.4524	0.3213	0.2865	0.2512	0.2580
SELFATT	0.3651	0.4506	0.6809	0.6922	0.5866	0.6011	0.5628
ROBERTA	0.6245	0.6060	0.8204	0.8087	0.6659	0.6576	0.6972
PT W/ CODE ONLY (INIT=S)	0.5712	0.5557	0.7929	0.7855	0.6567	0.6172	0.6632
PT W/ CODE ONLY (INIT=R)	0.6612	0.6402	0.8191	0.8438	0.7213	0.6706	0.7260
CODEBERT (MLM, INIT=S)	0.5695	0.6029	0.8304	0.8261	0.7142	0.6556	0.6998
CODEBERT (MLM, INIT=R)	0.6898	0.6997	0.8383	0.8647	0.7476	0.6893	0.7549
CODEBERT (RTD, INIT=R)	0.6414	0.6512	0.8285	0.8263	0.7150	0.6774	0.7233
CODEBERT (MLM+RTD, INIT=R)	**0.6926**	**0.7059**	**0.8400**	**0.8685**	**0.7484**	**0.7062**	**0.7603**

Table 2: Results on natural language code retrieval. Baselines include four joint embeddings (first group) of NL and PL, RoBERTa, and RoBERTa which is continuously trained with masked language modeling on codes only (second group). PT stands for pre-training. We train CodeBERT (third group) with different settings, including using different initialization (from scratch (INIT=S) or initialized with the parameters of RoBERTa (INIT=R)) and using different learning objectives (MLM, RTD, or the combination of both).

(Evaluated on CodeSearchNet Corpus)

CodeBERT Results: Docstring Generation

- SOTA on docstring generation



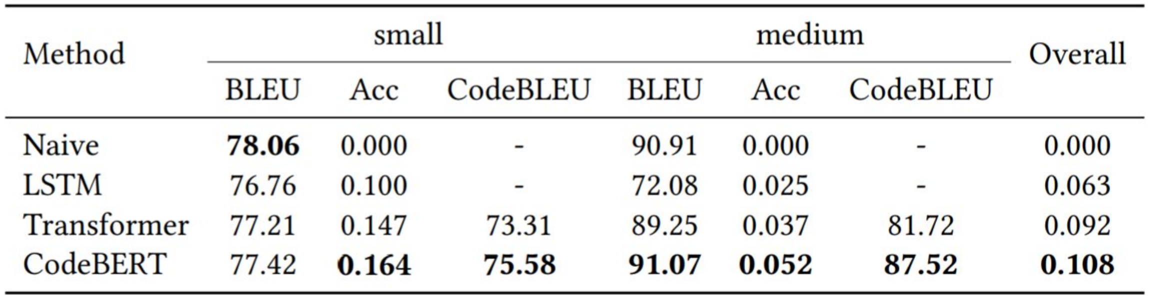
MODEL	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	OVERALL
SEQ2SEQ	9.64	10.21	13.98	15.93	15.09	21.08	14.32
TRANSFORMER	11.18	11.59	16.38	15.81	16.26	22.12	15.56
ROBERTA	11.17	11.90	17.72	18.14	16.47	24.02	16.57
PRE-TRAIN W/ CODE ONLY	11.91	13.99	17.78	18.58	17.50	24.34	17.35
CODEBERT (RTD)	11.42	13.27	17.53	18.29	17.35	24.10	17.00
CODEBERT (MLM)	11.57	14.41	17.78	18.77	17.38	24.85	17.46
CODEBERT (RTD+MLM)	**12.16**	**14.90**	**18.07**	**19.06**	**17.65**	**25.16**	**17.83**

Table 4: Results on Code-to-Documentation generation, evaluated with smoothed BLEU-4 score.

Encoder-Decoder Models

- Machine-translation-like architecture
  - CodeBERT Encoder
  - Transformer Decode
- SOTA on
  - Code repair
  - Code translation
  - Clone detection

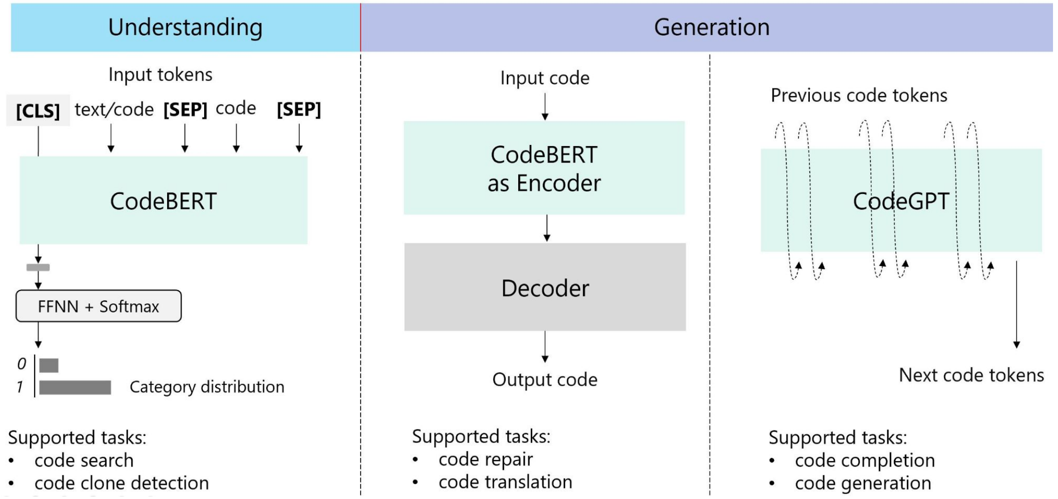
Table 11: Results on the code repair task.



Method	small			medium			Overall
	BLEU	Acc	CodeBLEU	BLEU	Acc	CodeBLEU	
Naive	**78.06**	0.000	-	90.91	0.000	-	0.000
LSTM	76.76	0.100	-	72.08	0.025	-	0.063
Transformer	77.21	0.147	73.31	89.25	0.037	81.72	0.092
CodeBERT	77.42	**0.164**	**75.58**	**91.07**	**0.052**	**87.52**	**0.108**

CodeXGlue: Pretrained Models

CodeXGlue repo includes pretrained CodeGPT, CodeBERT, and encoder-decoder models.

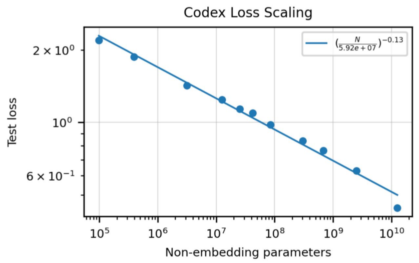


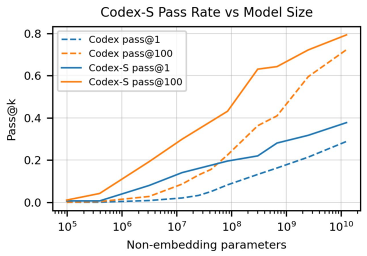
Future Directions

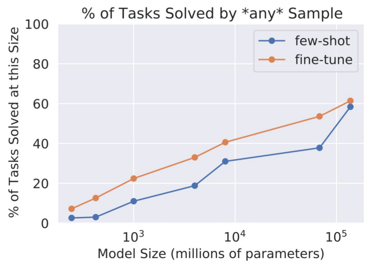
Roughly in order from least to most speculative

More Compute! More Data!

- More compute and more data will improve performance
- All these graphs are straight lines with no signs of diminishing returns







Latent Code Generation for Reasoning and QA

Question: A foo and a bar are a kind of baz. John has 4 foo and 7 bar in a basket, on the way home he lost 2 foo. How many baz does he have?

GPT-3 completion:

John has 3 baz in a basket.

Codex completion:

```
def baz_count(foo, bar, lost_foo):
    return foo + bar - lost_foo

print(baz_count(4, 7, 2))
```

Latent Code Generation for Reasoning and QA

- Codex can also do *commonsense* reasoning (sometimes)

```
"""
John has 4 oranges and 7 apples in a
basket.
On the way home he loses 2 oranges.
How many fruit does he have?
"""

def num_fruit(apples, oranges):
    return apples + oranges - 2

print(num_fruit(4, 7))
```

Latent Code Generation for Reasoning and QA

- Current systems struggle with question-answering that involves deductive and commonsense reasoning.
- Incorporate latent code generation as intermediate step

1. Treat question as code completion prompt.
2. Complete with Codex-like model
3. Surface realization

- Extra benefit: reasoning using code is interpretable

Semi-supervised curriculum learning

- Text-completion models can’t train on their outputs because we don’t know which outputs are high quality.
- But we can use held-out test cases to determine which outputs of text-to-code are high quality.

Separately,

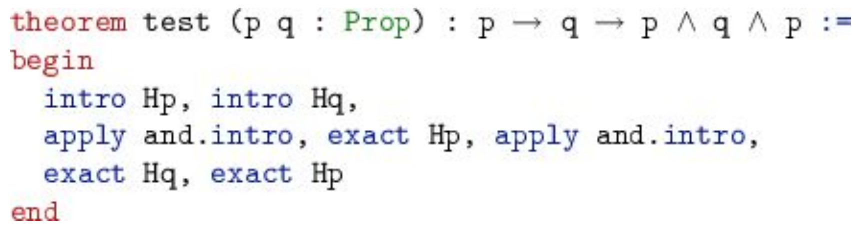
- *Curriculum learning*: Presenting data in order of increasing complexity improves learning (Bengio 2009)

Semi-supervised Curriculum Learning

1. Dataset consisting of labelled and unlabelled text-to-code problems indexed by difficulty
   a. Labelled data: Description, test cases, solution
   b. Unlabelled data: Description, test cases.
2. Train on problems of difficulty <=*N* until unlabelled examples of difficulty *N* solved.
3. Train on problems of difficulty <=*N+1*, including self-labelled examples of difficulty <=*N*.

Automating Mathematics: Interactive Theorem Provers

- Interactive theorem prover: a programming language for verifying mathematical proofs.
- Lean, Coq, Isabelle, Agda, HOL Light
- As simple theorem in Lean:



```
theorem test (p q : Prop) : p → q → p ∧ q ∧ p :=
begin
  intro Hp, intro Hq,
  apply and.intro, exact Hp, apply and.intro,
  exact Hq, exact Hp
end
```

Automating Interactive Theorem Provers

- Program synthesis for interactive theorem prover = automatic mathematician
- Challenge: Solve first-year undergraduate exercises with a language model.
- Grand challenge: Solve an open problem in mathematics using a language model.

Inductive Program Synthesis

- Traditional ML Model = Inductive Program Synthesis with very tightly constrained search space.
- Inductive Program Synthesis = ML with unconstrained search space.

Making a large inductive program synthesis dataset is not hard.

How would SOTA models perform when fine-tuned on it?

Discussion Questions

Codex API: https://beta.openai.com/playground/p/default-english-to-python

1. Would a human expert level code generation model solve human-like logical reasoning? What aspects of reasoning cannot be captured in code?

2. SOTA code-generation models use architectures designed for processing natural language. What optimizations could we make for processing code? What ideas from pre-neural methods might be useful?

Discussion Questions (cont.)

1. Do you think semi-supervised curriculum learning will work?

2. Is there any fundamental obstacle preventing a very large transformer from achieving human expert performance?

References

[1] Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models, 2021.

[2] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021.

[3] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., and Zhou, M. Codebert: A pre-trained model for programming and natural languages, 2020.

[4] Gulwani, S., Polozov, A., and Singh, R. *Program Synthesis*, vol. 4. NOW, August 2017.

[5] Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., and Steinhardt, J. Measuring coding challenge competence with apps, 2021.

[6] Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S. K., Fu, S., and Liu, S. Codexglue: A machine learning benchmark dataset for code understanding and generation, 2021.

[7] Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C. B., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S. K., Fu, S., and Liu, S. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR abs/2102.04664* (2021).