Large Pre-Trained Models for Code

Patrik Matosevic

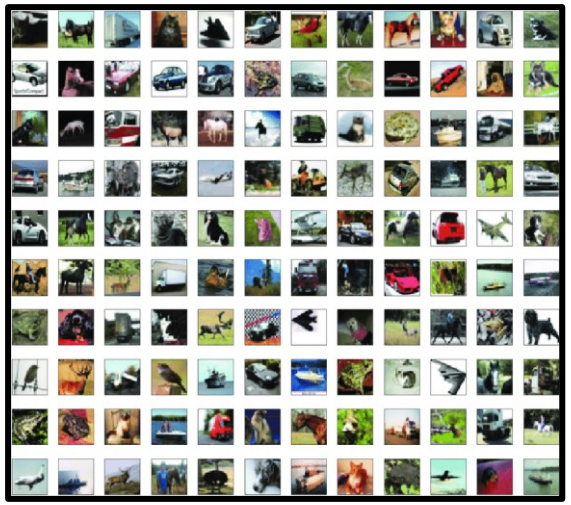
Mentor: Peter Belcak

ETH Zurich - Seminar in Deep Neural Networks, 10.05.2022

NL TEXT

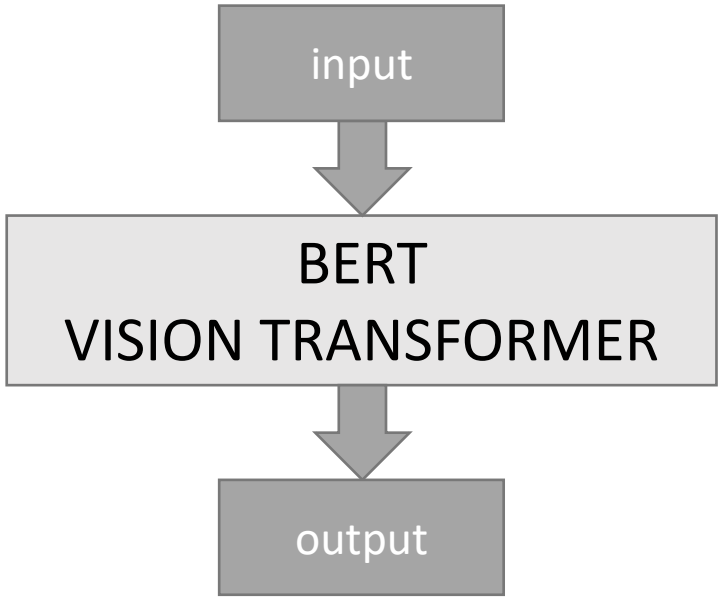
Happy families are all alike; every unhappy family is unhappy in its own way.
Everything was in confusion in the Oblonskys' house. The wife had discovered that the husband was carrying on an intrigue with a French girl…

IMAGE



CODE

```
def go_shopping():
    if store.has_eggs():
        store.buy_milk()
    for i in range(5):
        store.buy_bread()
```

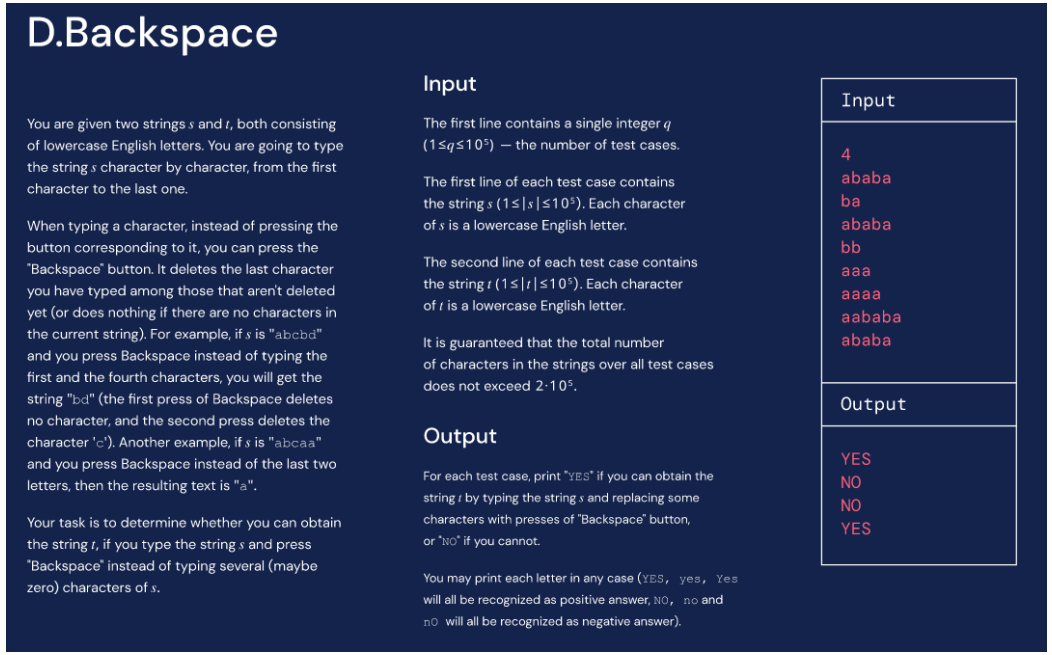


GitHub Copilot

```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2016-01-02 -34.01 USD
        2016-01-03 2.59 DKK
        2016-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                         float(value),
                         currency))
    return expenses
```


Illustrations taken from https://copilot.github.com/



D.Backspace

You are given two strings s and t, both consisting of lowercase English letters. You are going to type the string s character by character, from the first character to the last one.

When typing a character, instead of pressing the button corresponding to it, you can press the "Backspace" button. It deletes the last character you have typed among those that aren't deleted yet (or does nothing if there are no characters in the current string). For example, if s is "abcbd" and you press Backspace instead of typing the first and the fourth characters, you will get the string "bd" (the first press of Backspace deletes no character, and the second press deletes the character 'c'). Another example, if s is "abcaa" and you press Backspace instead of the last two letters, then the resulting text is "a".

Your task is to determine whether you can obtain the string t, if you type the string s and press "Backspace" instead of typing several (maybe zero) characters of s.

Input

The first line contains a single integer q ($1 \le q \le 10^5$) — the number of test cases.

The first line of each test case contains the string s ($1 \le |s| \le 10^5$). Each character of s is a lowercase English letter.

The second line of each test case contains the string t ($1 \le |t| \le 10^5$). Each character of t is a lowercase English letter.

It is guaranteed that the total number of characters in the strings over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print "YES" if you can obtain the string t by typing the string s and replacing some characters with presses of "Backspace" button, or "NO" if you cannot.

You may print each letter in any case (YES, yes, Yes will all be recognized as positive answer, NO, no and nO will all be recognized as negative answer).

Input
4 ababa ba ababa bb aaa aaaa aababa ababa
Output
YES NO NO YES

Problem statement

AlphaCode

```python
t=int(input())
for i in range(t):
    s=input()
    t=input()
    a=[]
    b=[]
    for j in s:
        a.append(j)
    for j in t:
        b.append(j)
    a.reverse()
    b.reverse()
    c=[]
    while len(b)!=0 and len(a)!=0:
        if a[0]==b[0]:
            c.append(b.pop(0))
            a.pop(0)
        elif a[0]!=b[0] and len(a)!=1:
            a.pop(0)
            a.pop(0)
        elif a[0]!=b[0] and len(a)==1:
            a.pop(0)
    if len(b)==0:
        print("YES")
    else:
        print("NO")
```

Code solution

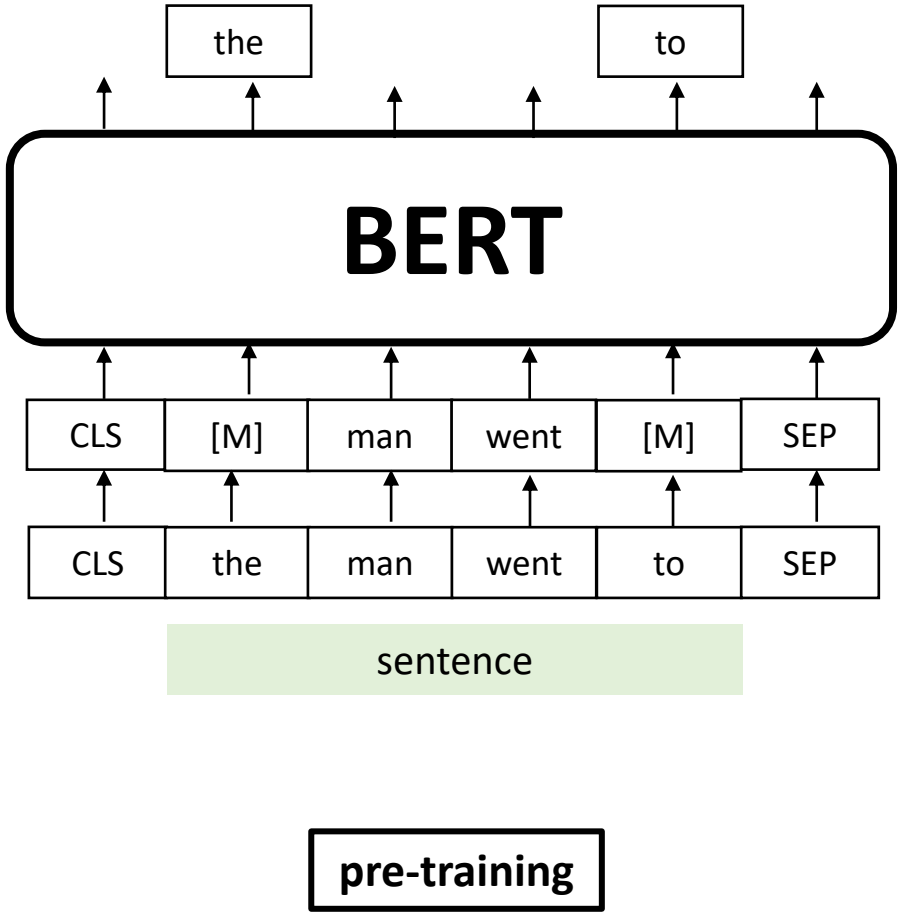
Motivation - Tasks

- Code generation
- Code search
- Code summarization
- Duplication detection
- Bug & vulnerability detection
- Programming language translation

BERT Architecture

- Training
  - 1) pre-training (masked LM + next sentence prediction)
  - 2) fine-tuning – task specific

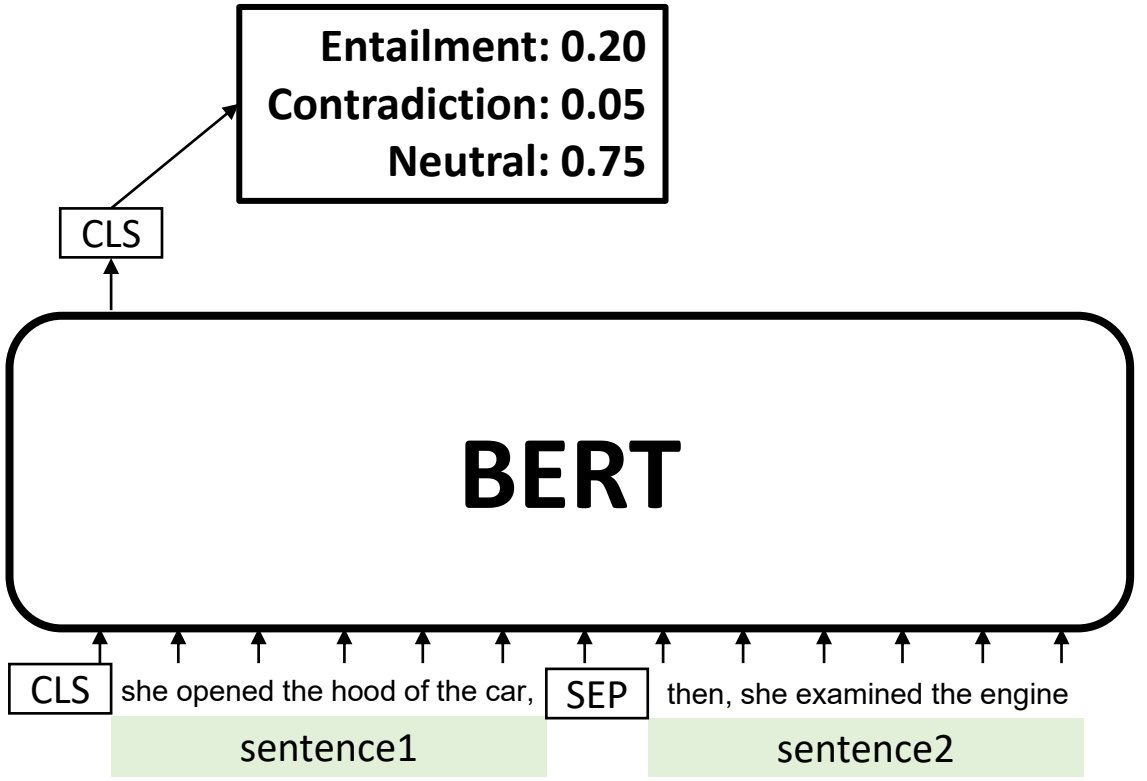
- Advancement: RoBERTa
  - more data
  - better train strategy (bigger batches, longer)
  - pre-training: only MLM



BERT Architecture

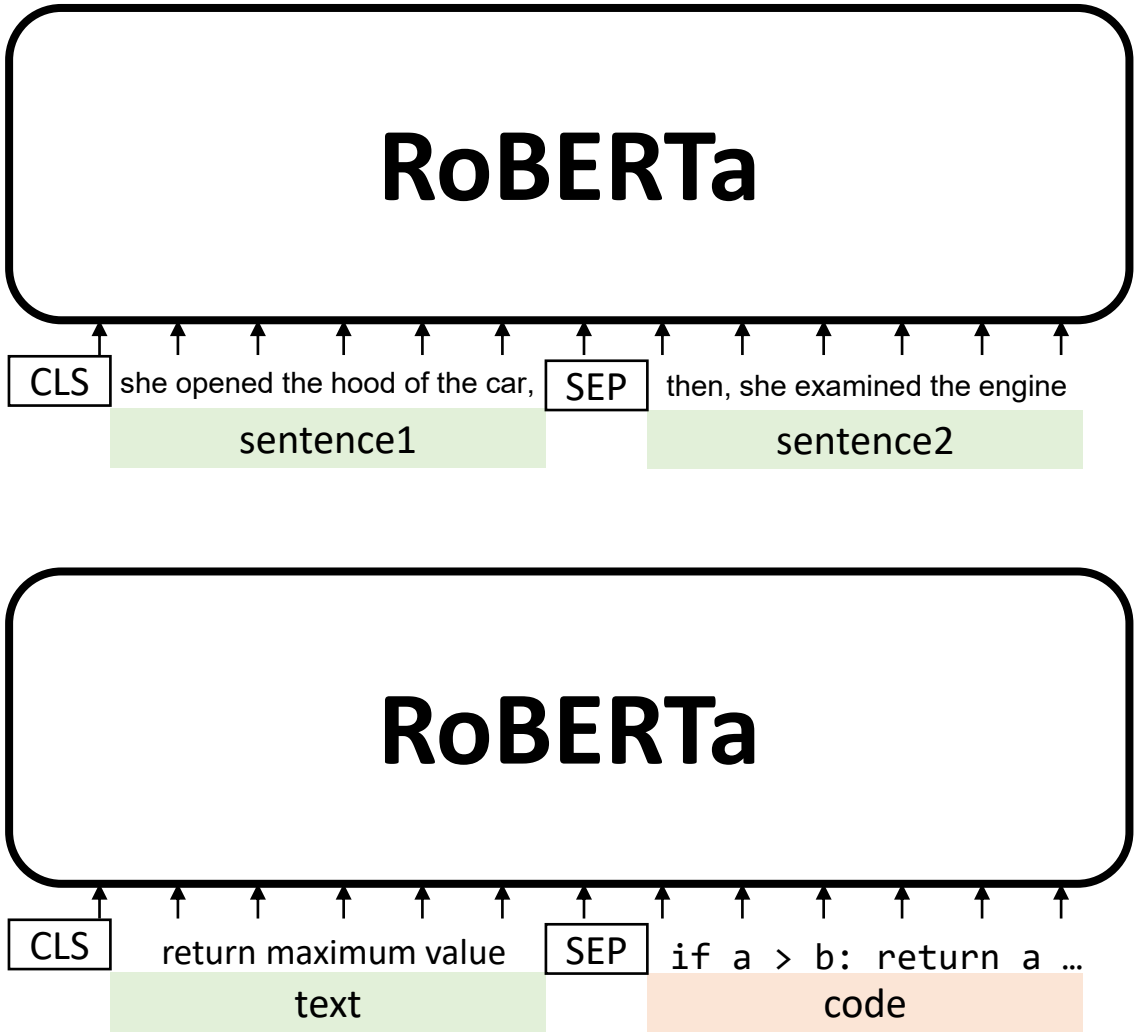
- Training
  - 1) pre-training (masked LM + next sentence prediction)
  - 2) fine-tuning – task specific

- Advancement: RoBERTa
  - more data
  - better train strategy (bigger batches, longer)
  - pre-training: only MLM



fine-tuning example: sentence-pair classification

RoBERTa on Code?



RoBERTa on Code?

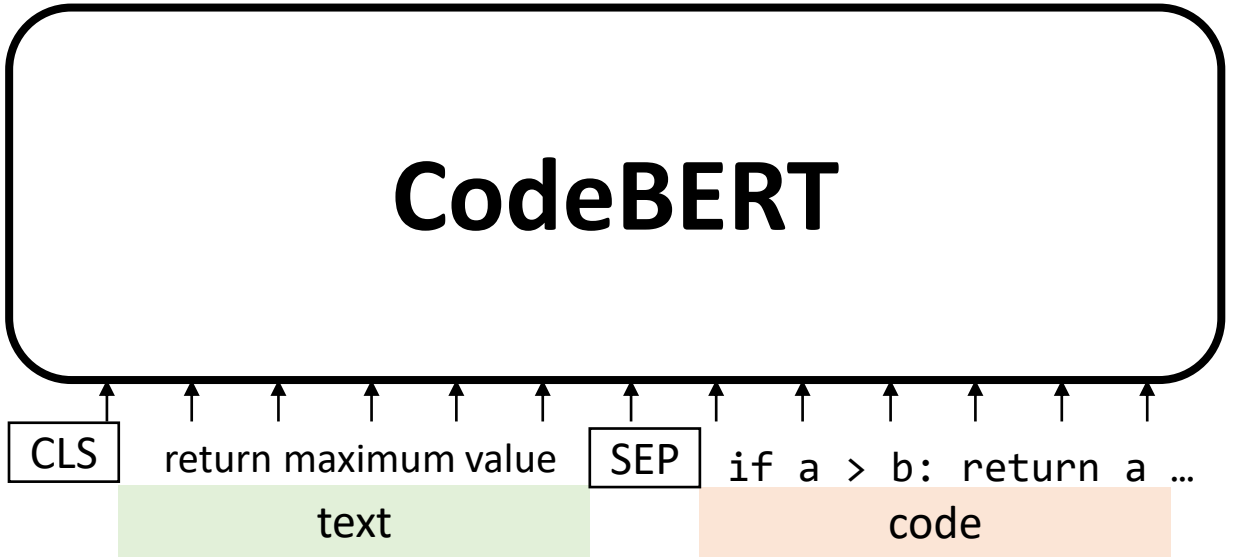
```
def max(a, b):
  """
  Returns the maximum of the
two numbers

  :param a: the first number
  :param b: the second number
  :returns: the maximum of
the two numbers
  """
  if a > b:
    return a
  else:
    return b
```

documentation

code fragment

CodeBERT (Feng et al., 2020)



- Architecture: RoBERTa
- Data: Code from public GitHub repos
  - 6 programming languages (CodeSearchNet dataset)
- Pre-training data:
  - NL (documentation) + PL (function code) pairs
- Pre-training tasks:
  - 1) Masked Language Modelling
  - 2) Replaced Token Detection

CodeBERT – Replaced Token Detection

NL and code generator – simple n-gram language model

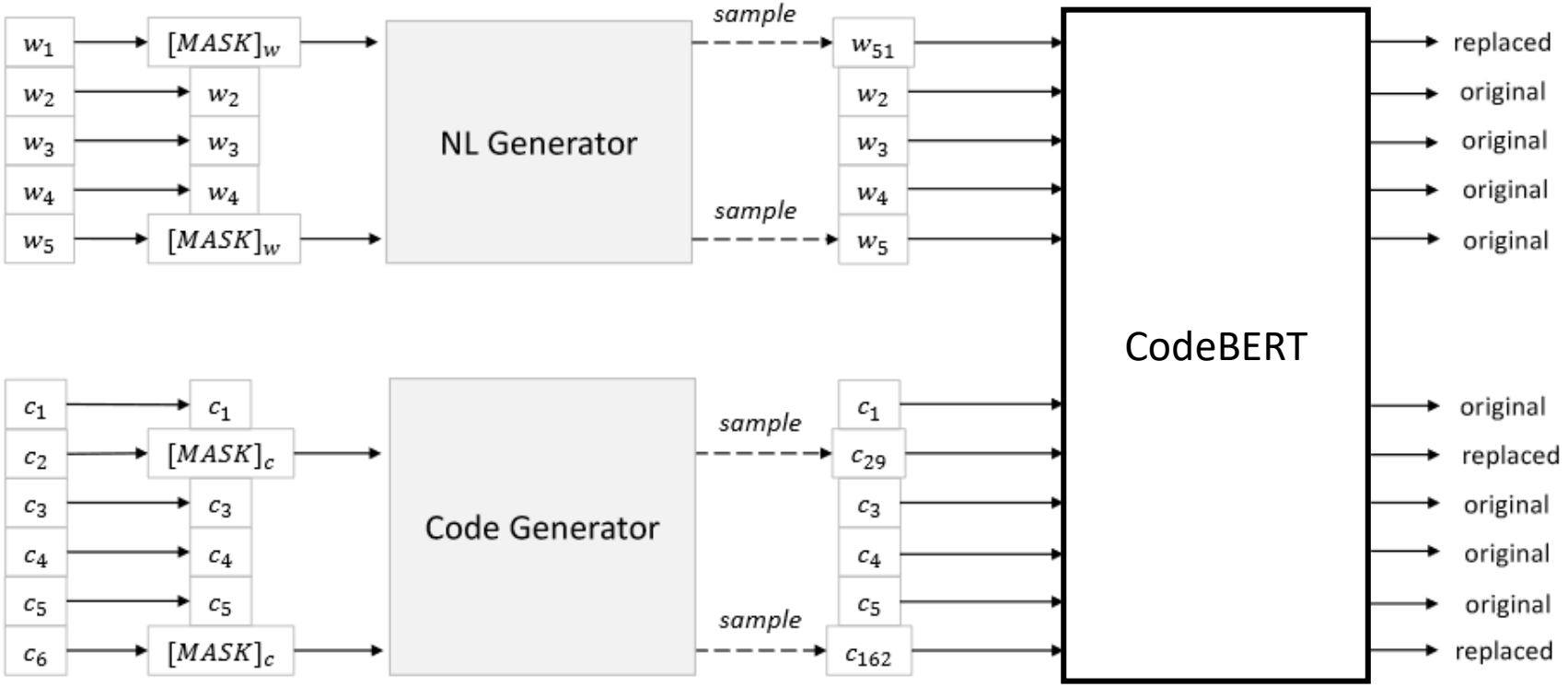
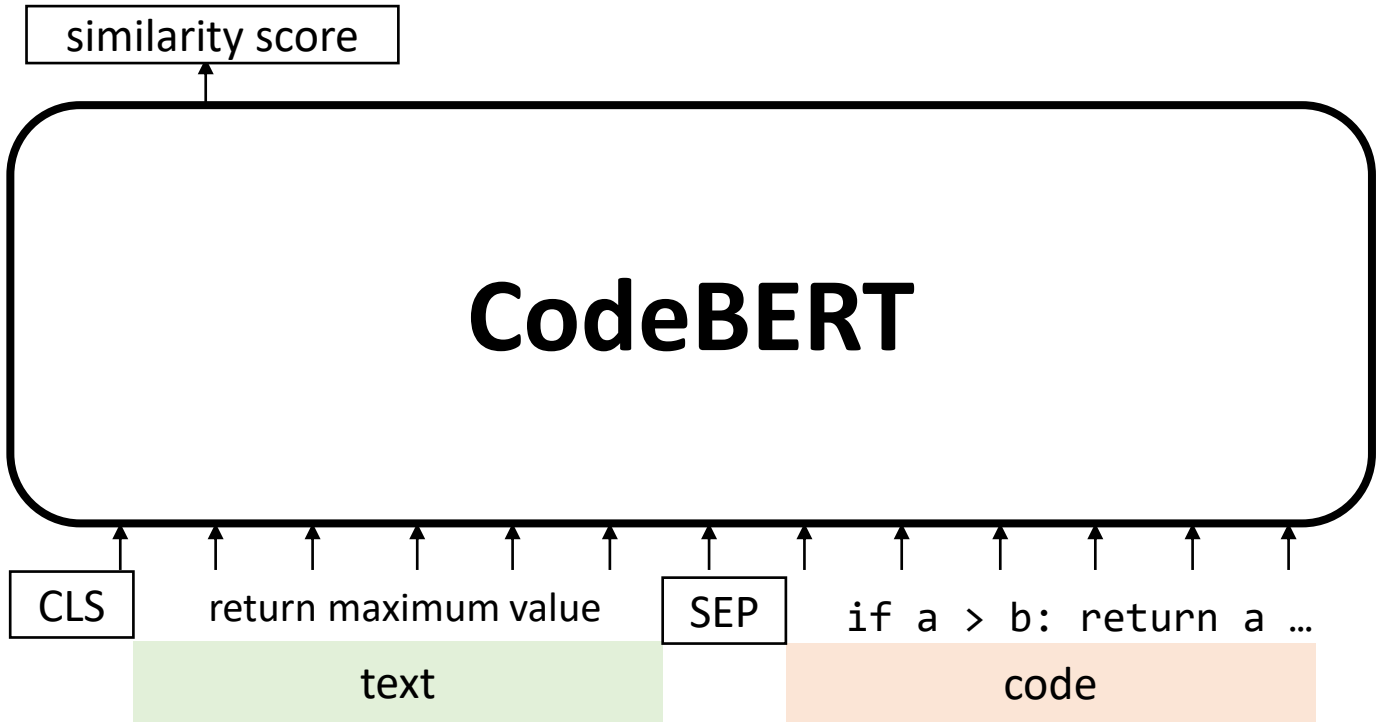


Figure taken from Feng et al., 2020

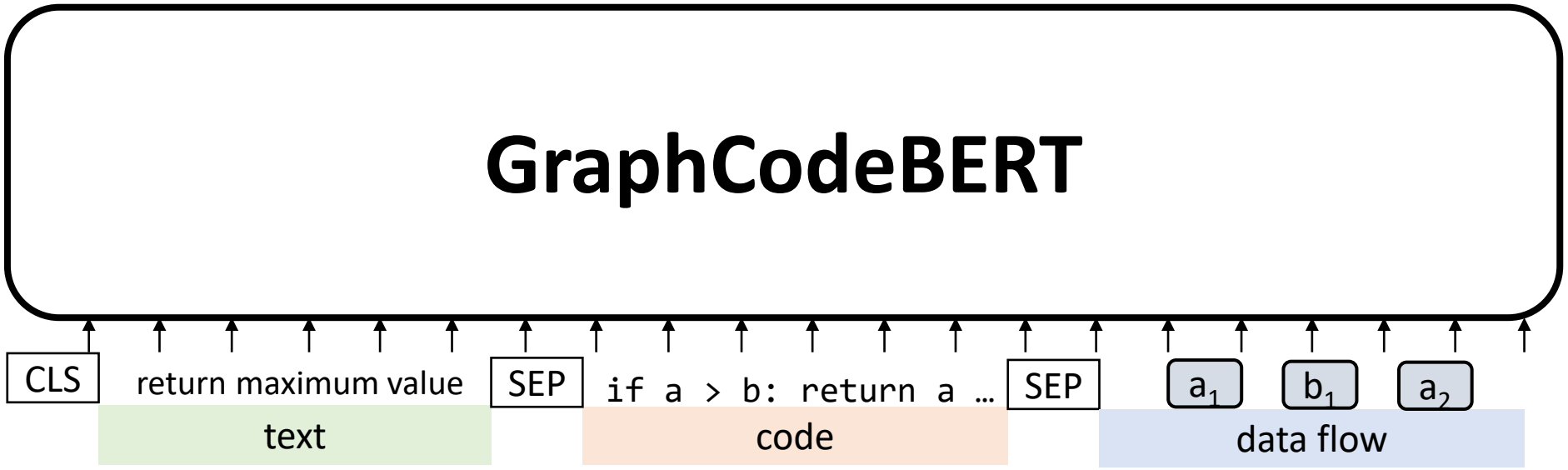
CodeBERT – Fine-Tuning (Code Search)



Code search task

- Output: similarity score [0, 1]
- Data: CodeSearchNet corpus
  - Code + docstring pairs
- Balanced positive and negative samples
  - Negative sample – replace docstring or code

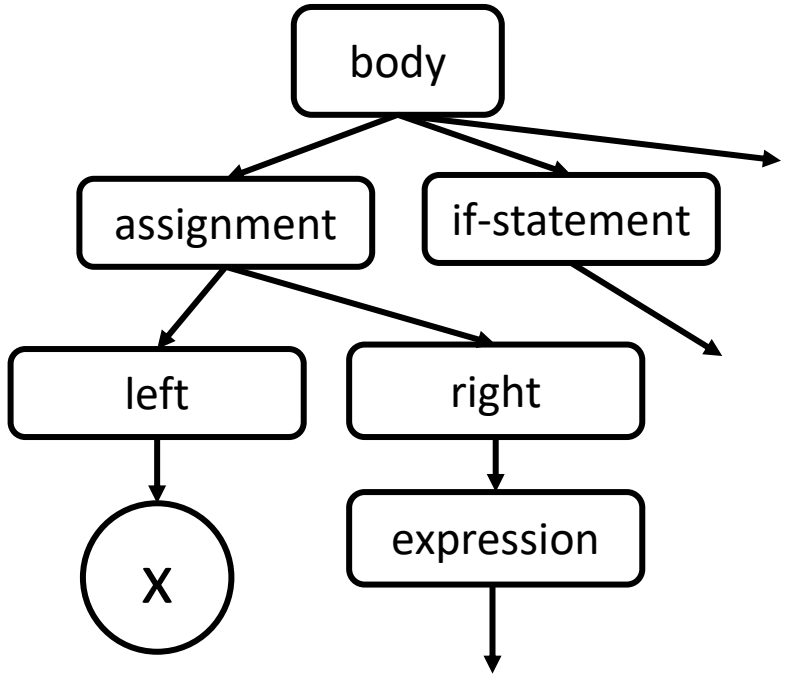
GraphCodeBERT (Guo et al., 2021)



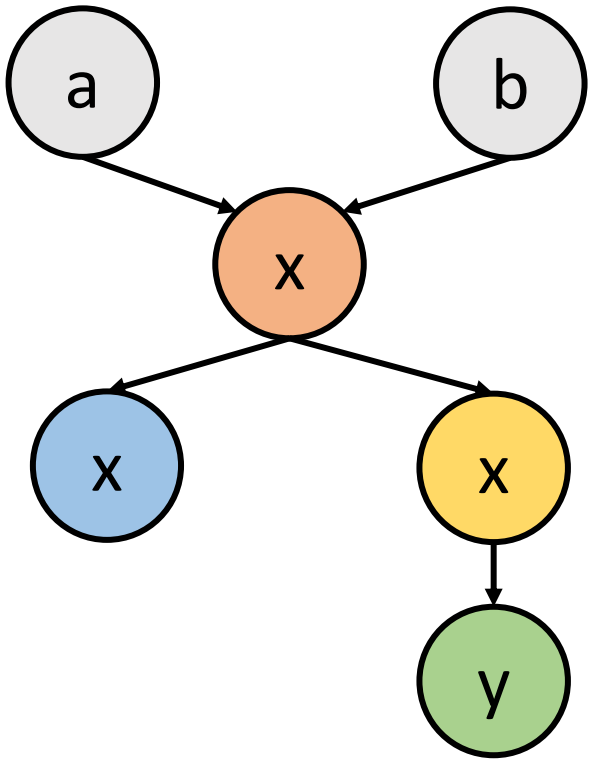
GraphCodeBERT - Data Flow

```
a = 2
b = 3
x = a + b
if x < 0:
    y = x
```

AST



variable relation graph

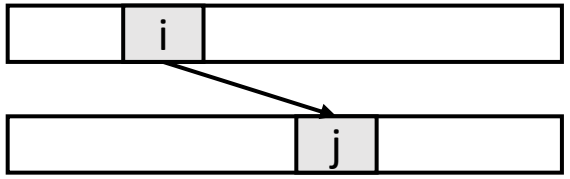


→ = value comes from

$$V = \{a, b, x_1, x_2, x_3, y\}$$
$$E = \{(a, x_1), (b, x_1), (x_1, x_2), (x_1, x_3), (x_3, y)\}$$

GraphCodeBERT - Attention

Attention (dot-product): $q_i \cdot k_j$



Attention scores:

$$\alpha = \text{softmax}(\frac{Q \cdot K}{\sqrt{d}})$$

$$\alpha = \text{softmax}(\frac{Q \cdot K}{\sqrt{d}} + M)$$

$$M_{ij} = 0 \text{ or } -\infty$$

M_{ij}	text			code			data flow		
text	0	0					-inf	-inf	
	0						-inf		
code									
								0	
						0			
data flow									
					0		0		
									-inf

Allow data flow token attention only with:

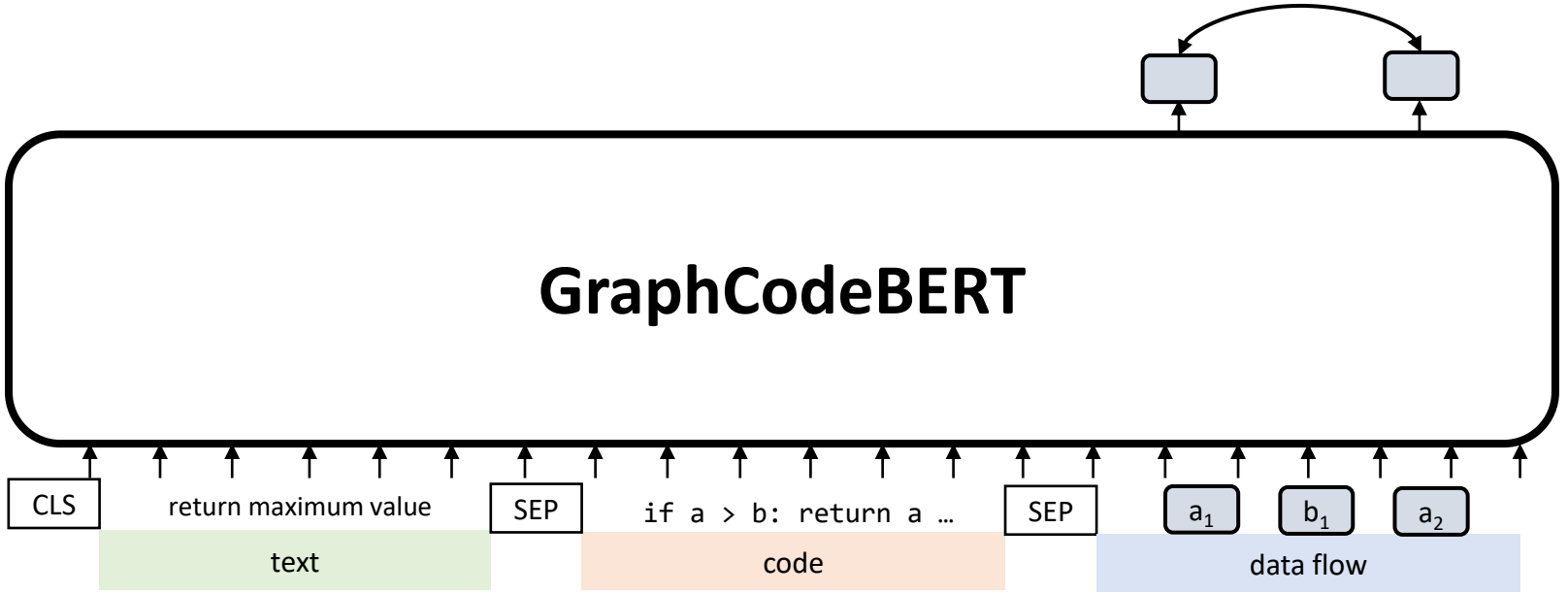
1. Code token that correspond to it
2. Data flow token with edge in relation graph
3. [CLS], [SEP]

GraphCodeBERT (Guo et al., 2021)

- Pre-training tasks:
  - 1) Masked Language Modeling
    - Data flow tokens are never masked
  - 2) Edge Prediction
  - 3) Node Alignment

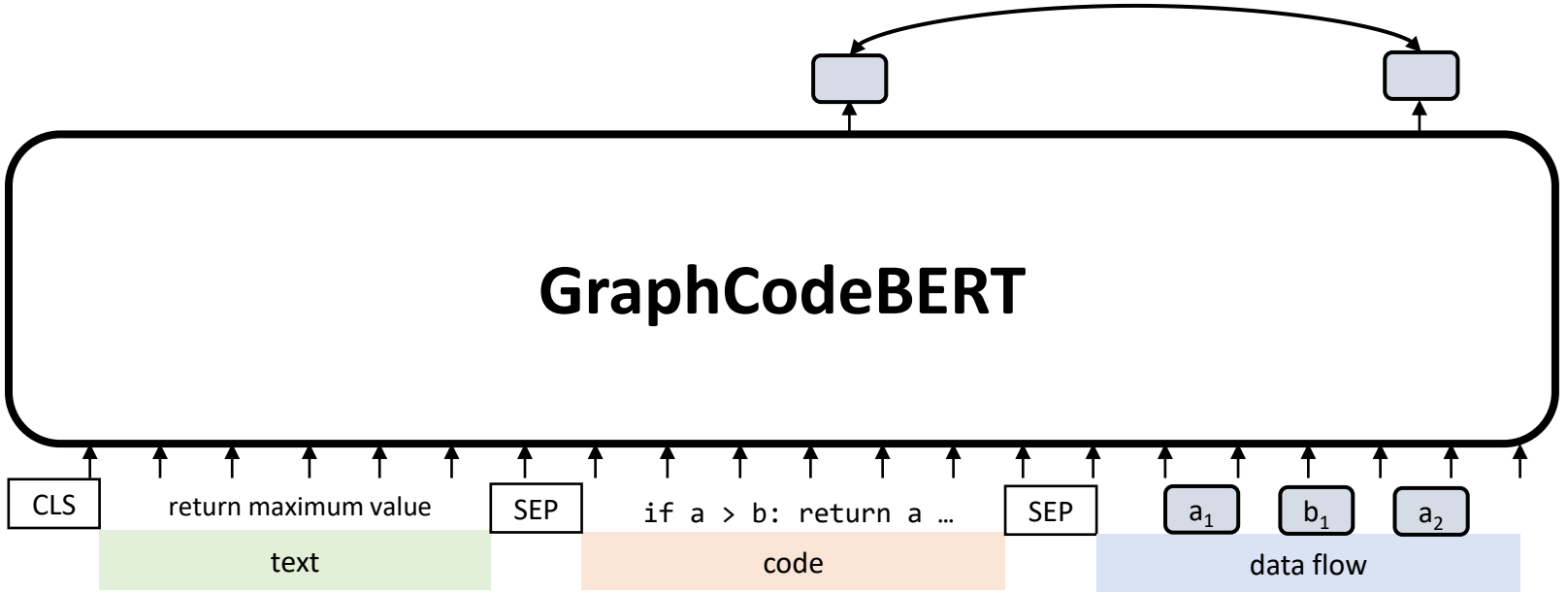
GraphCodeBERT – Edge Prediction

- Edge Prediction
  - Learn to predict edges in data flow (where the value comes from)
  - Procedure:
    - Randomly select data flow tokens
    - Mask them in attention
    - Let the model predict them (product = 1 if exists or 0 otherwise)



GraphCodeBERT – Node Alignment

- Node Alignment
  - Learn to predict edges between code tokens and data flow tokens
  - Procedure:
    - Randomly select data flow and code token pairs
    - Mask them in attention
    - Let the model predict them (product = 1 if exists or 0 otherwise)



SynCoBERT (Wang et al., 2021)

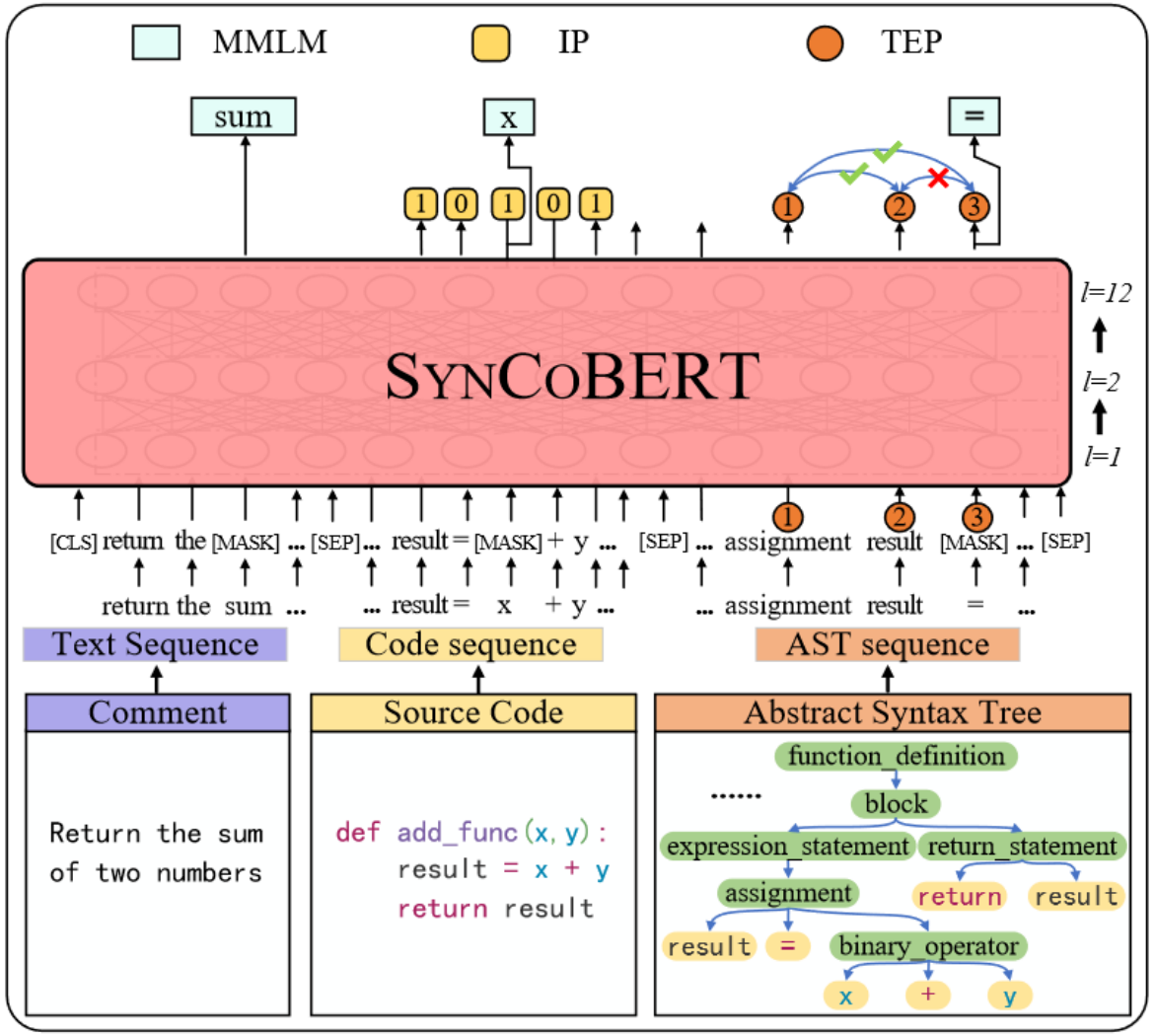
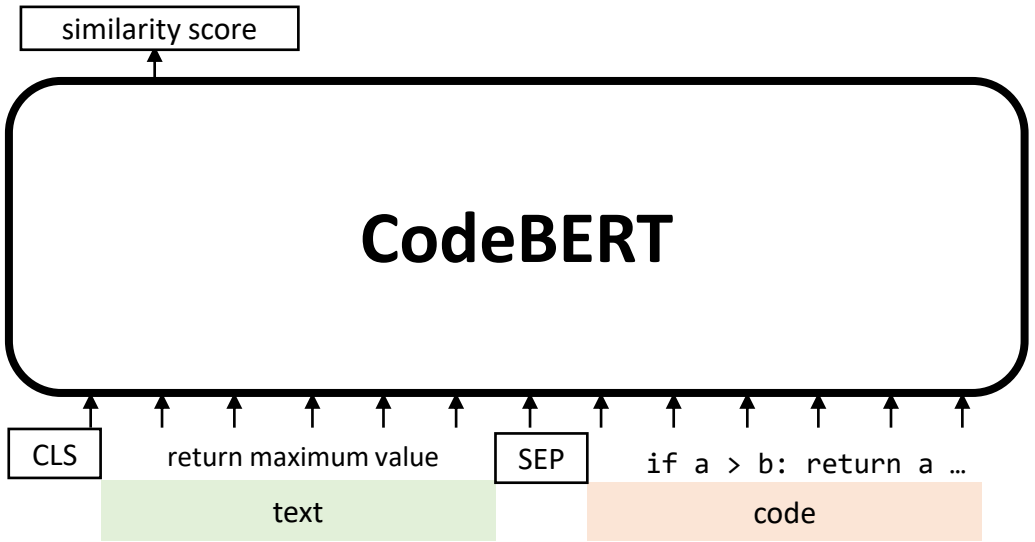


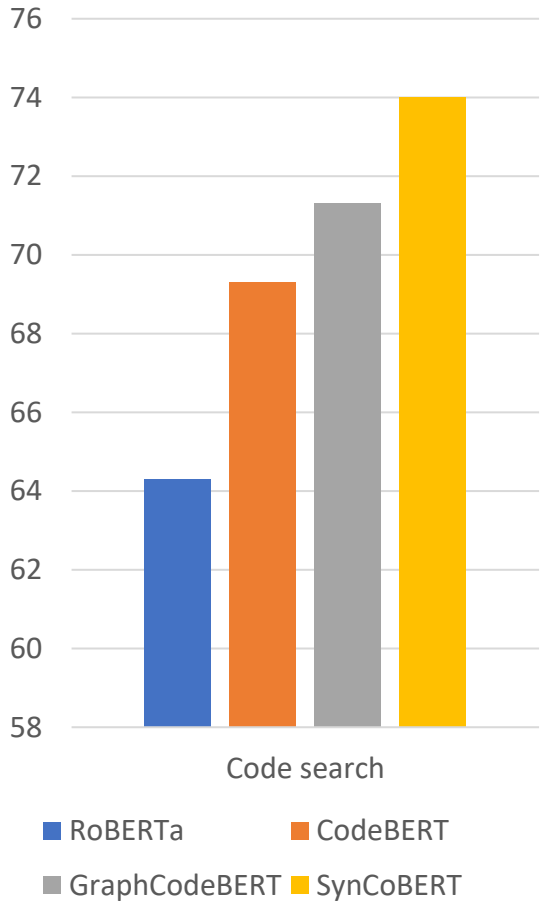
Figure taken from Wang et al., 2021

- Pre-training tasks:
  - 1) (multi-modal) masked LM
    - AST tokens can also be masked
  - 2) identifier prediction
    - Predict whether a token is identifier
  - 3) AST edge prediction
    - Predict masked edges in AST
  - 4) (multi-modal) contrastive learning
    - Solve imbalance problems (high-frequency tokens)
    - Positive and negative samples

Tasks – Natural Language Code Search

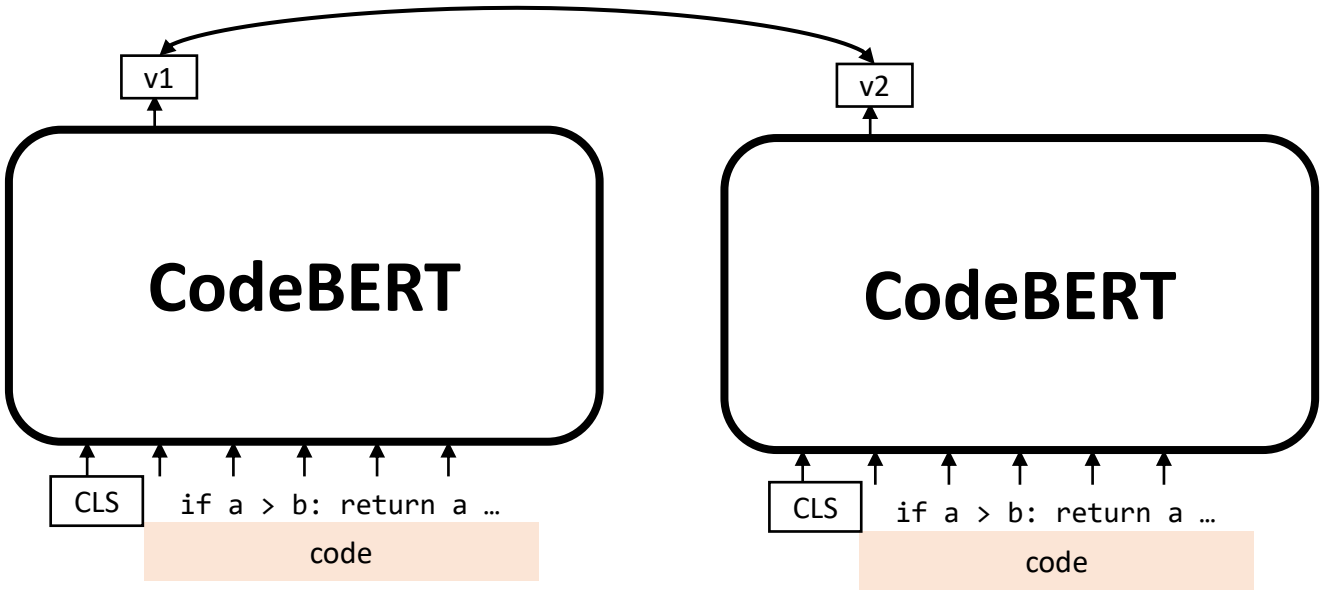
- From a collection, find the most semantically related code given NL description
- Dataset: CodeSearchNet
- Metric: MRR (Mean Reciprocal Rank)
- 999 distractors
- $\mathrm{MRR}=\frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{\mathrm{rank_i}}$

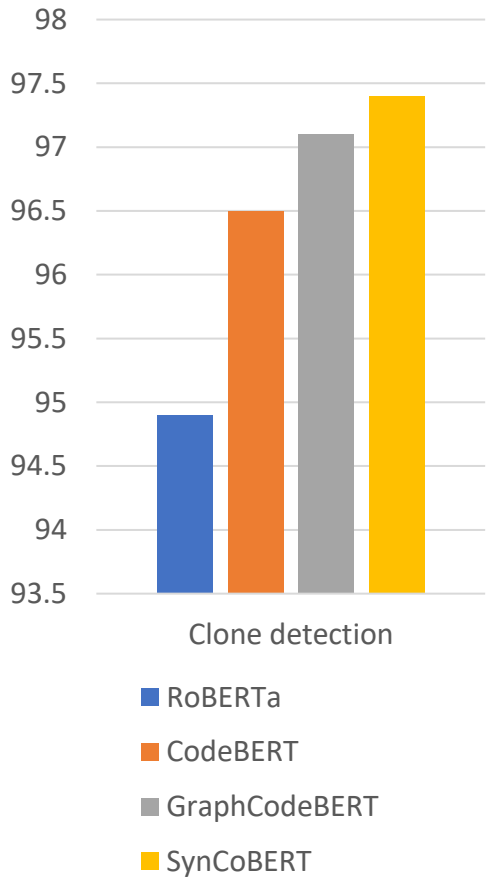




Tasks - Code Clone Detection

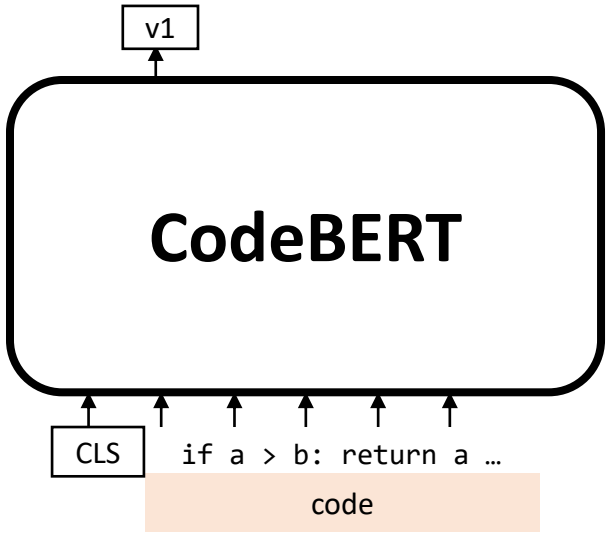
- Detect whether 2 code fragments output same result when given the same input
- Dataset: BigCloneBench
  - Pairs of code -> clone / not clone (binary)
- Metric: F1 score

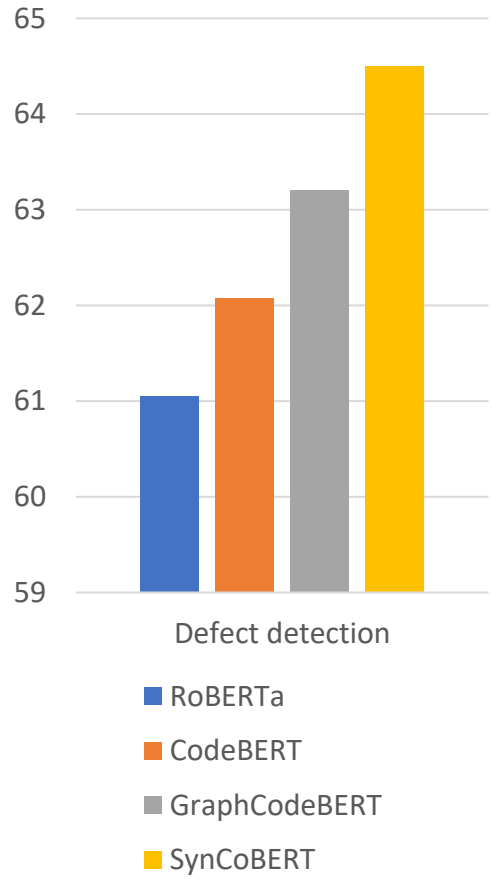




Tasks - Code Defect Detection

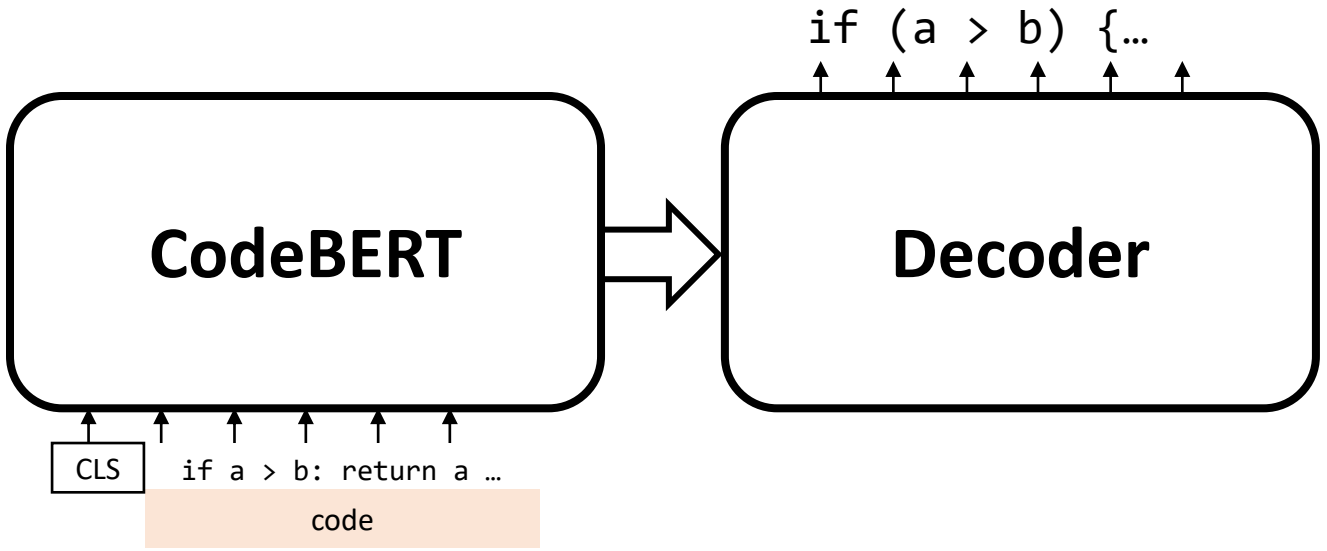
- Detect whether there are defects in the code fragment (binary)
- Dataset: Defects4J (C language)
- Metric: accuracy

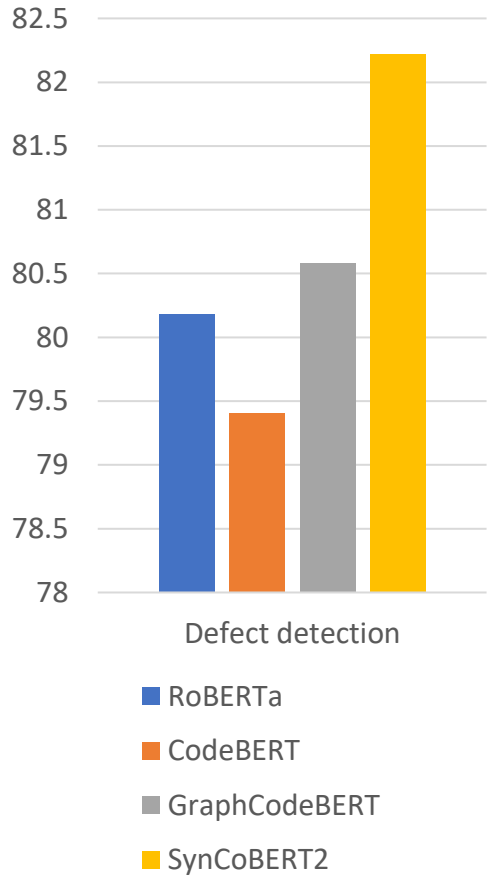




Tasks - Program Translation

- Translate the code from one to another language
- Dataset: C# -> Java
- Metric: BLEU (CodeBLEU)





AlphaCode

- A model that solves competitive programming problems
- Data:
  - Pre-training: public repos from GitHub
  - Fine-tuning: CodeContests – task and solution pairs (including incorrect ones) from codeforces

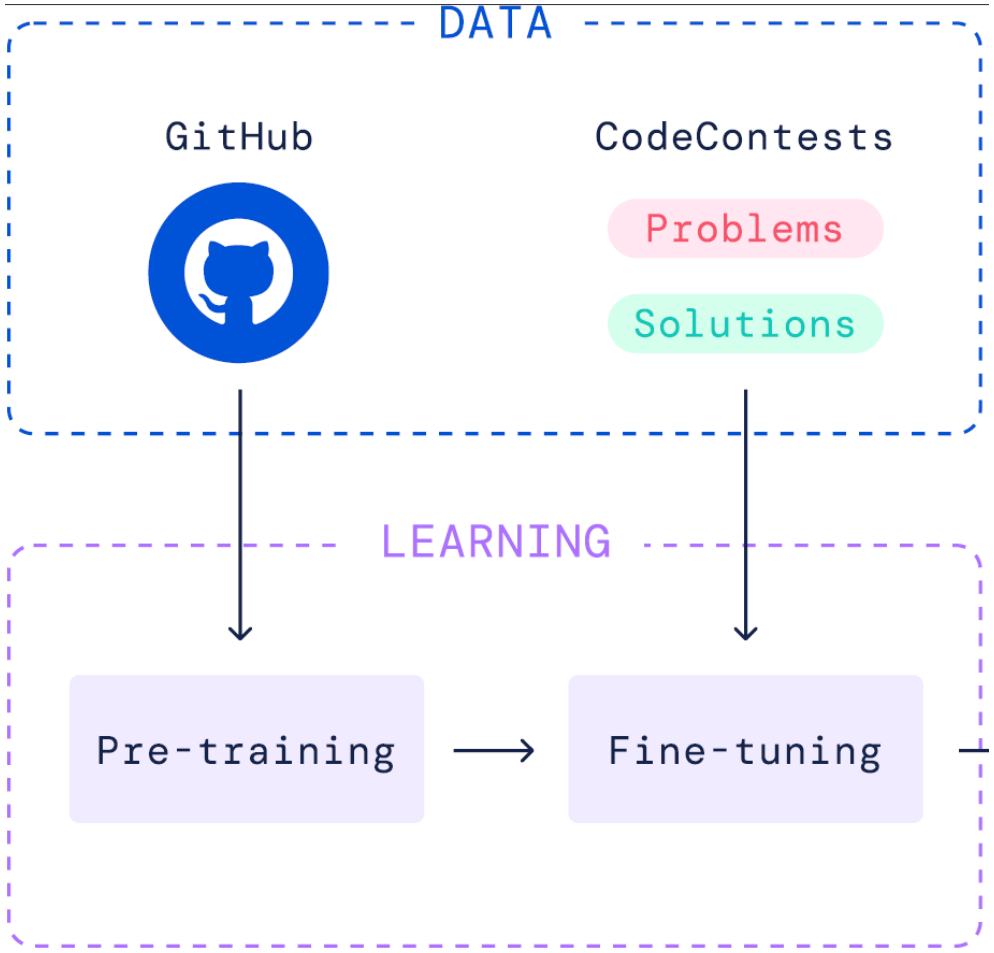
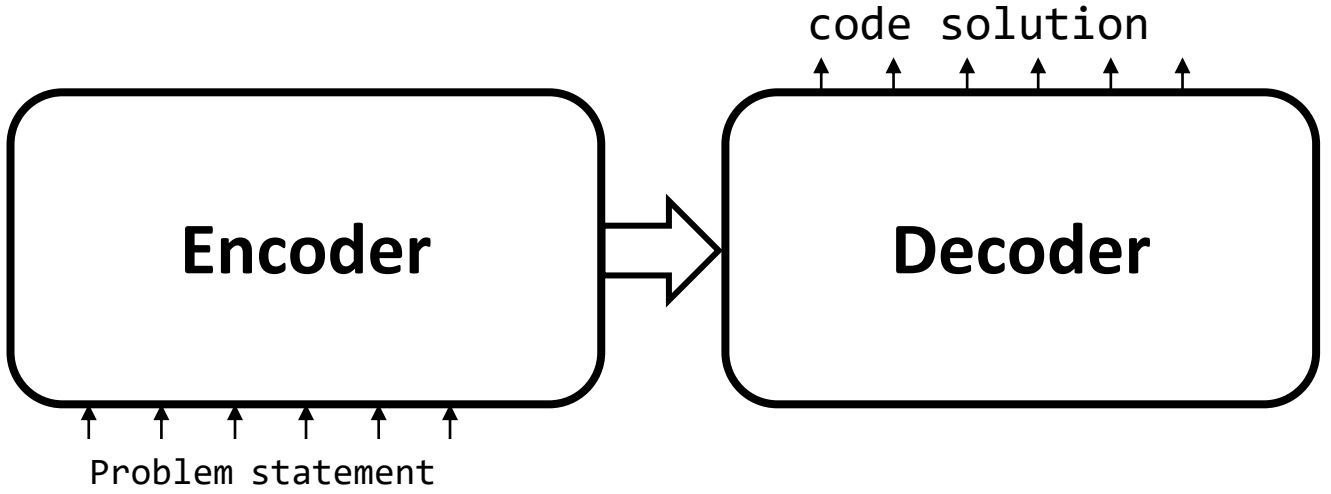


Figure taken from Li et al., 2022

AlphaCode

- Architecture: encoder-decoder transformer
  - problem statement -> solution code pair
  - like for natural language translation
  - no syntax info
- 41B parameters



AlphaCode

- Evaluation:
  - Generate many samples (C++ & python) -> 100k
  - Filter based on whether the generated solution passes simple tests
  - From a separate model generate new tests and cluster candidates
  - Select 10 samples

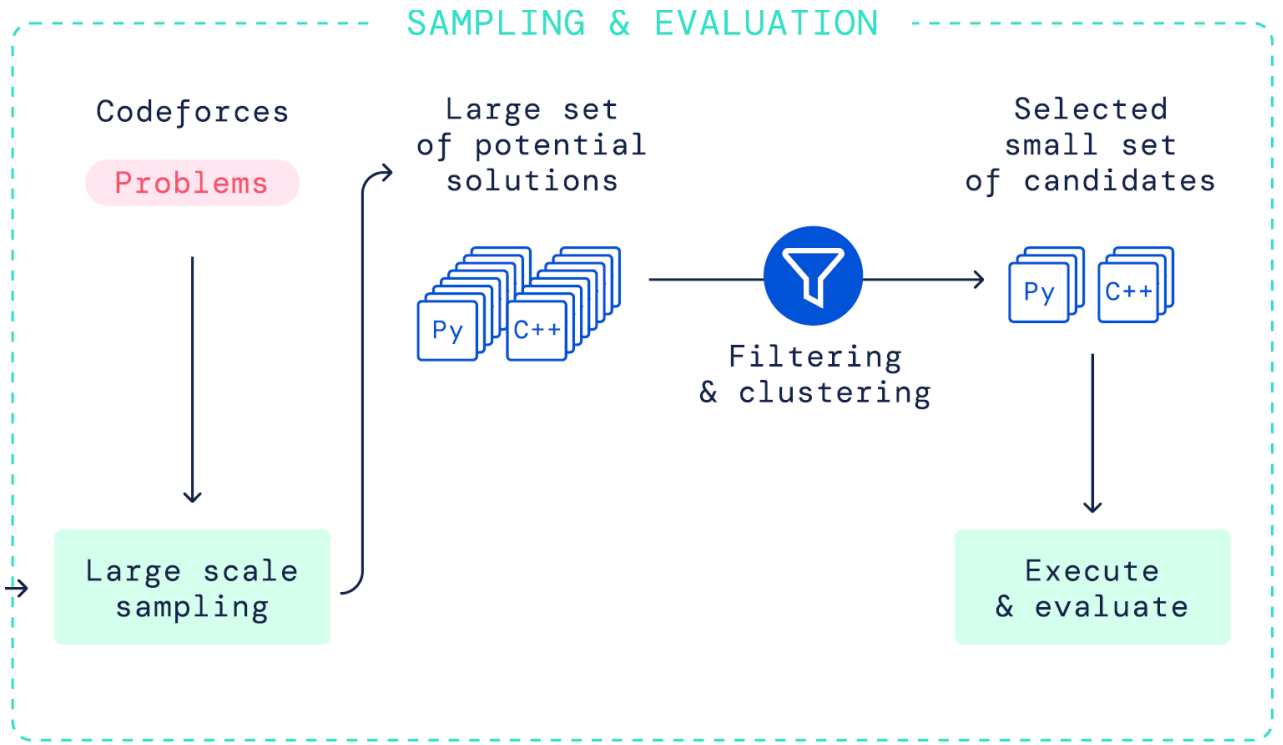


Figure taken from Li et al., 2022

AlphaCode

- Result
  - Within top 54% contestants based on sampled solutions
- Limitations and problems
  - More syntactically correct Python than C++ samples -> syntax understanding?
  - Lack of understanding complex ideas:
    - Better at problems with bitmaps, sorting, greedy
    - Worse at DP problems
  - Generating many samples (100k)
    - During sampling, a random tag (DP, binary search, …) is added
    - Just generating all possible solutions with different algorithms?

AlphaCode

Chess – DeepBlue vs Garry Kasparov

Figure taken from https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov

OpenAI Codex

- Model used for Github Copilot
- Architecture: based on GPT (12B) + modified text tokenization (?)
- Data: Python code from Github public repos
- pass@k evaluation (like AlphaCode)
- Experiments
  - Hand-crafted evaluation set (NL and PL pairs)
  - Additional fine-tuning on correct code (competitive programming + CI)

OpenAI Codex

OpenAI Codex - Problems

```
"""
Fetch tweets from twitter
"""

import os
import time
import json
import twitter
import pickle
import sys
import logging
import argparse
import pdb
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.corpus import sentiwordnet as swn
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from textblob.sentiments import NaiveBayesAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
from textblob.sentiments import PatternAnalyzer
```

```
train = pd.read_csv('./dataset/train.csv')
test = pd.read_csv('./dataset/test.csv')
```

```
import tweepy
import time

# Go to https://apps.twitter.com/ to create your own Twitter app
# and get all the keys and access tokens
CONSUMER_KEY = '...'
CONSUMER_SECRET = '...'
ACCESS_KEY = '...'
ACCESS_SECRET = '...'
```

Conclusion

- Models for textual data can also be used for code (with slight modifications)
- Efficient program representation
  - Improves results
  - More in the following presentation!
- Training a huge model with billions of parameters produces amazing results – advancement, fairness?

Thank you for your attention!

References

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- RoBERTa: A Robustly Optimized BERT Pretraining Approach
- CodeBERT: A Pre-Trained Model for Programming and Natural Languages
- GraphCodeBERT: Pre-training Code Representations with Data Flow
- SynCoBERT: Syntax-Guided Multi-Modal Contrastive Pre-Training for Code Representation
- CodeSearchNet Challenge: Evaluating the State of Semantic Code Search
- Competition-Level Code Generation with AlphaCode
- Evaluating Large Language Models Trained on Code