Code Pretraining

Daniel Fried

11-891: Neural Code Generation

https://cmu-codegen.github.io/s2024/

Language Technologies Institute

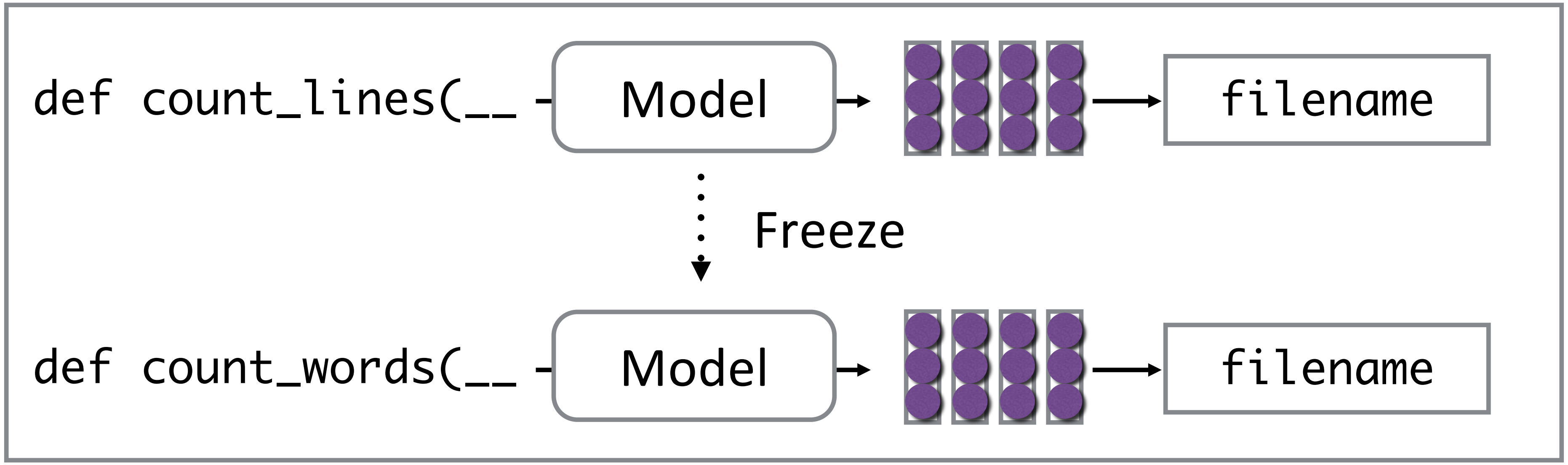
With slides from Greg Durrett, Nikitha Rao, and Zora Wang

Prompting

Train the model to generate language/code, then use -- without updating the model -- on other generation tasks.

Generation

Generation

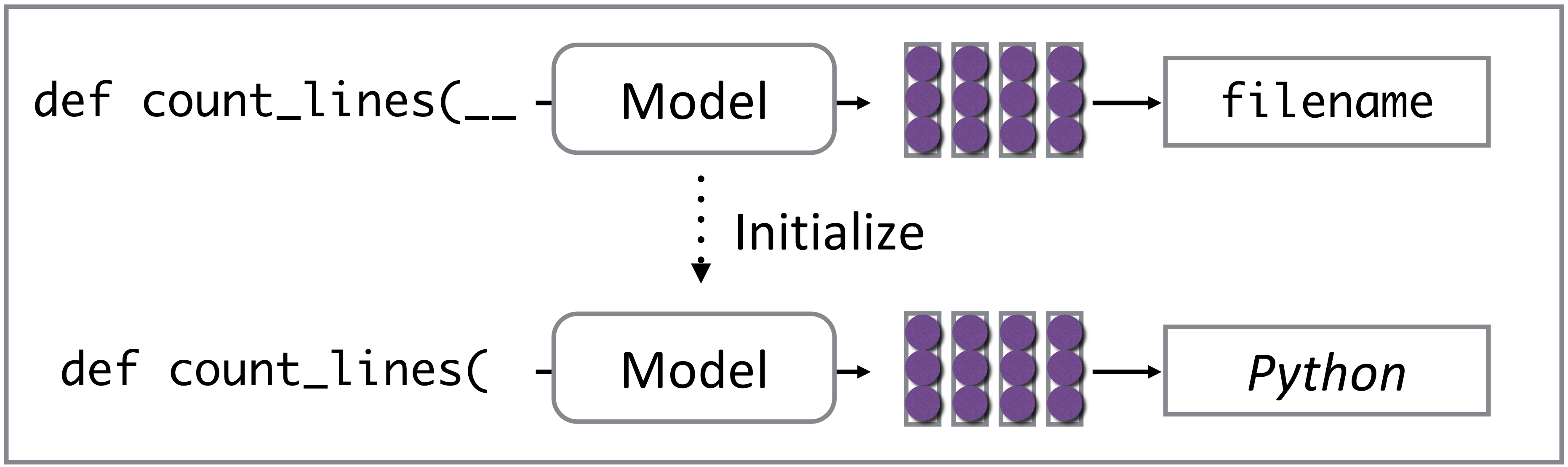


Pre-train and Fine-Tune

First train on one task, then train on another

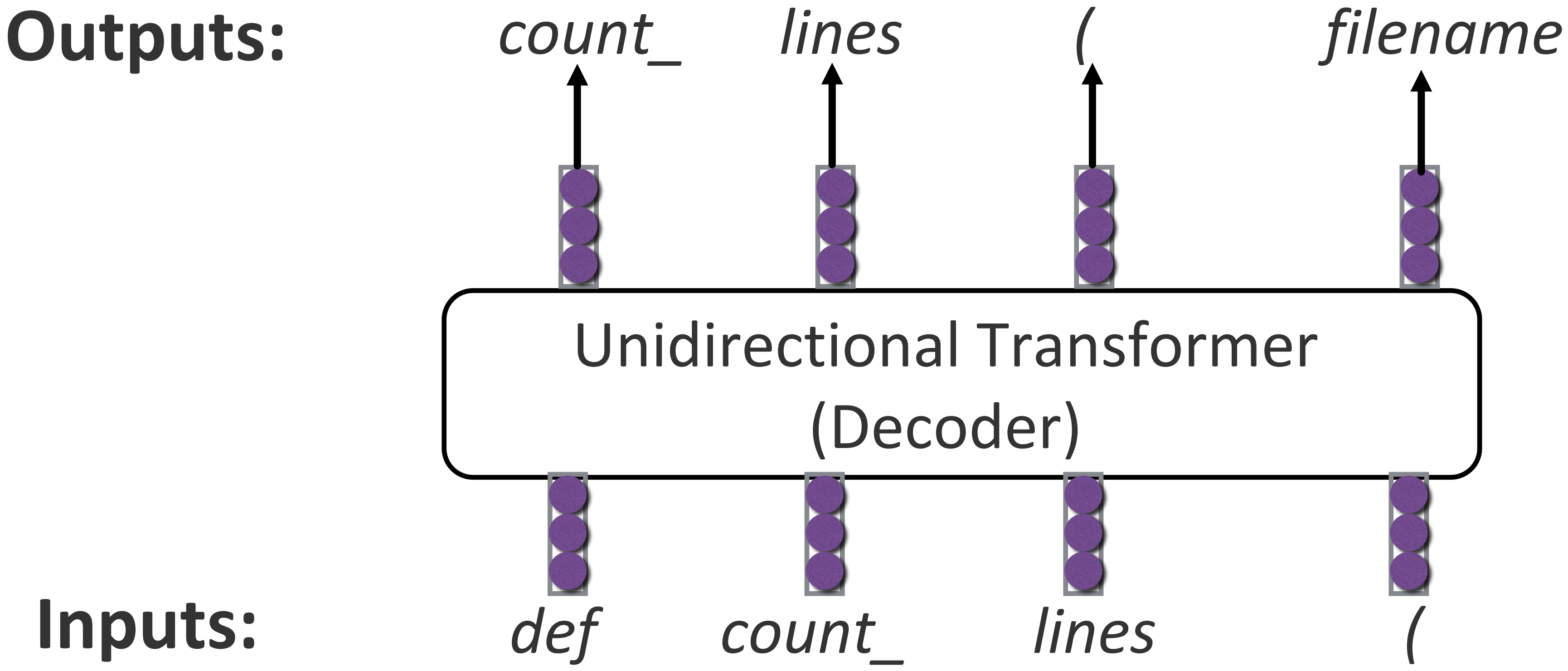
Generation

Classification



Objectives: Autoregressive Language Modeling

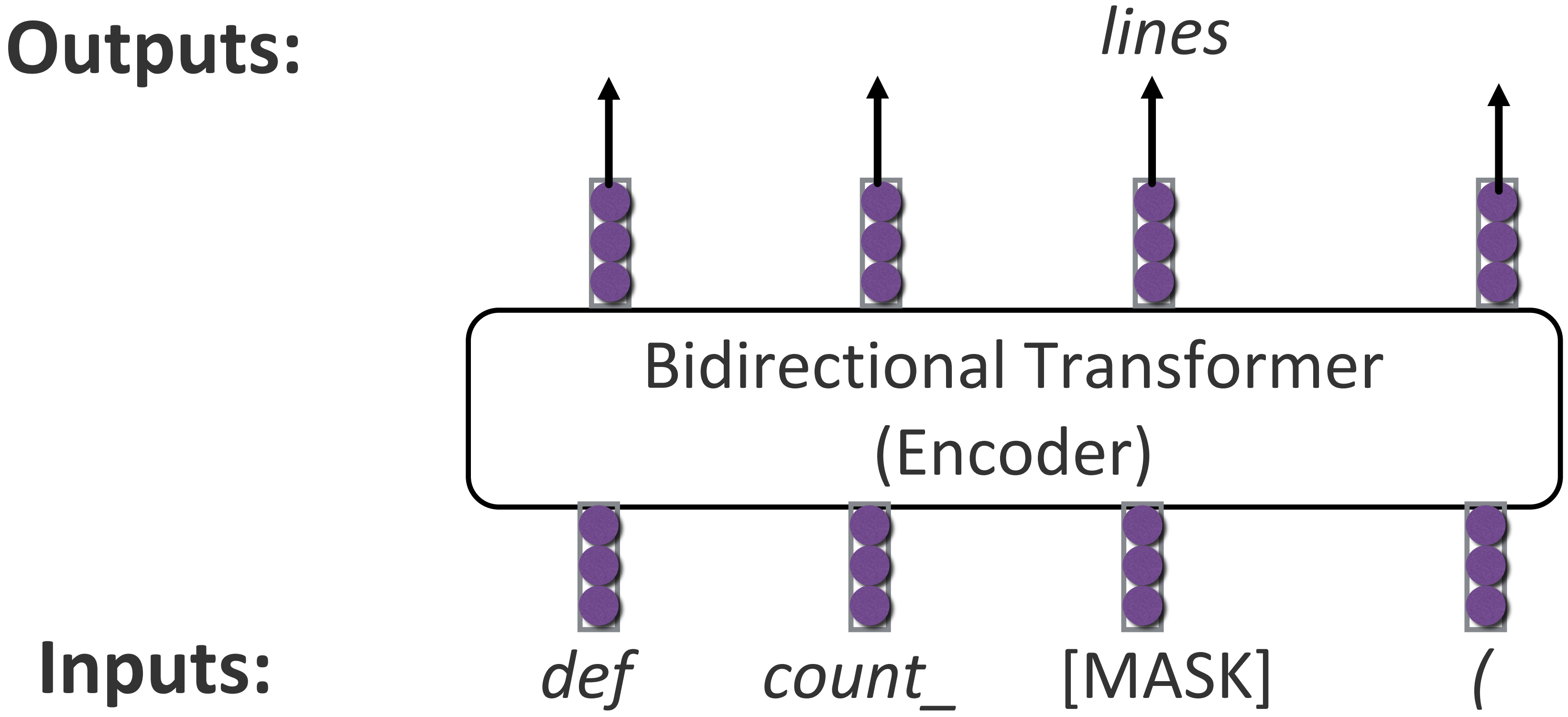
$$P(X) = \prod_{i=1}^{|X|} P(x_i | x_1, \ldots, x_{i-1})$$



Used mostly for generation/prompting

Objectives: Masked Language Modeling

$$P(X) \neq \prod_{i=1}^{|X|} P(x_i | x_{\neq i})$$

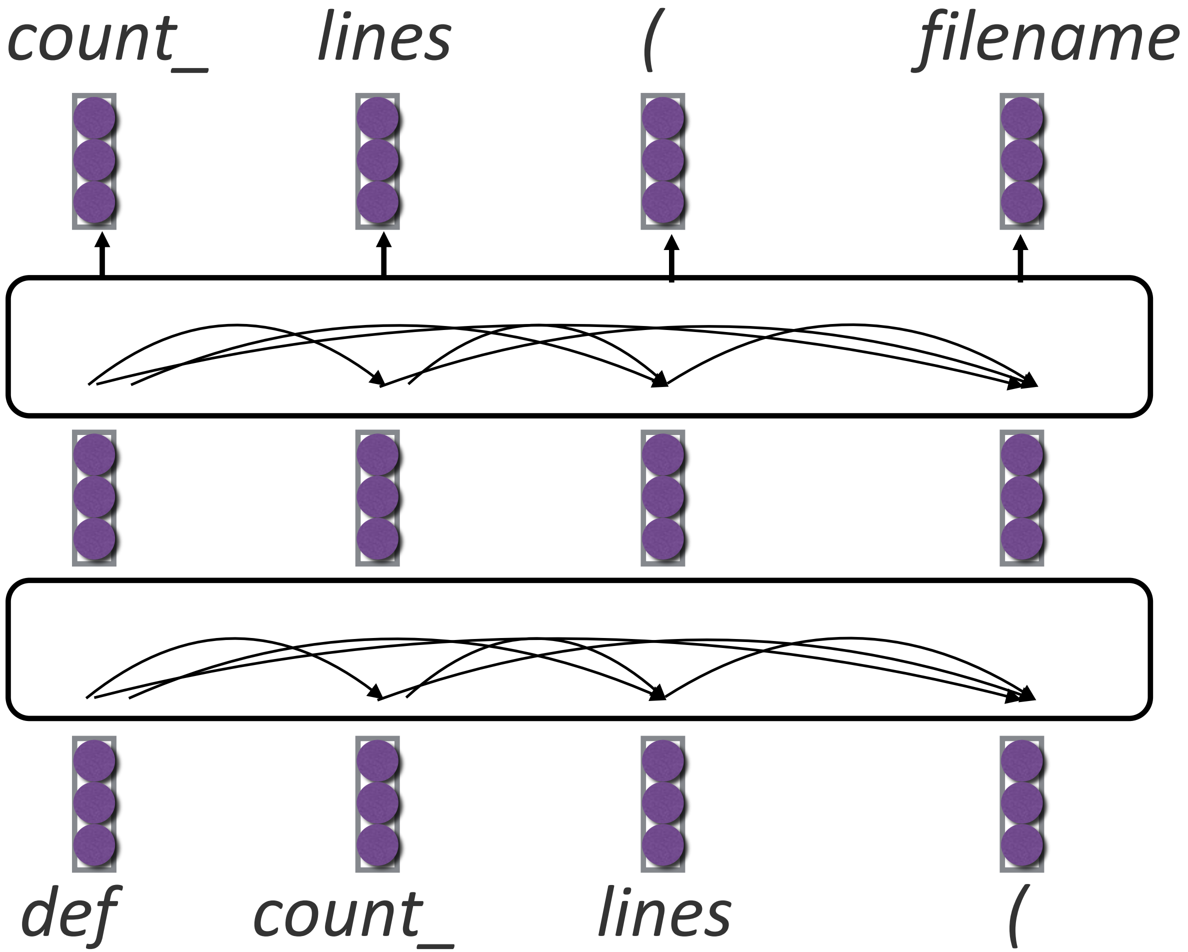


Used mostly for representation learning

Unidirectional vs Bidirectional Transformers

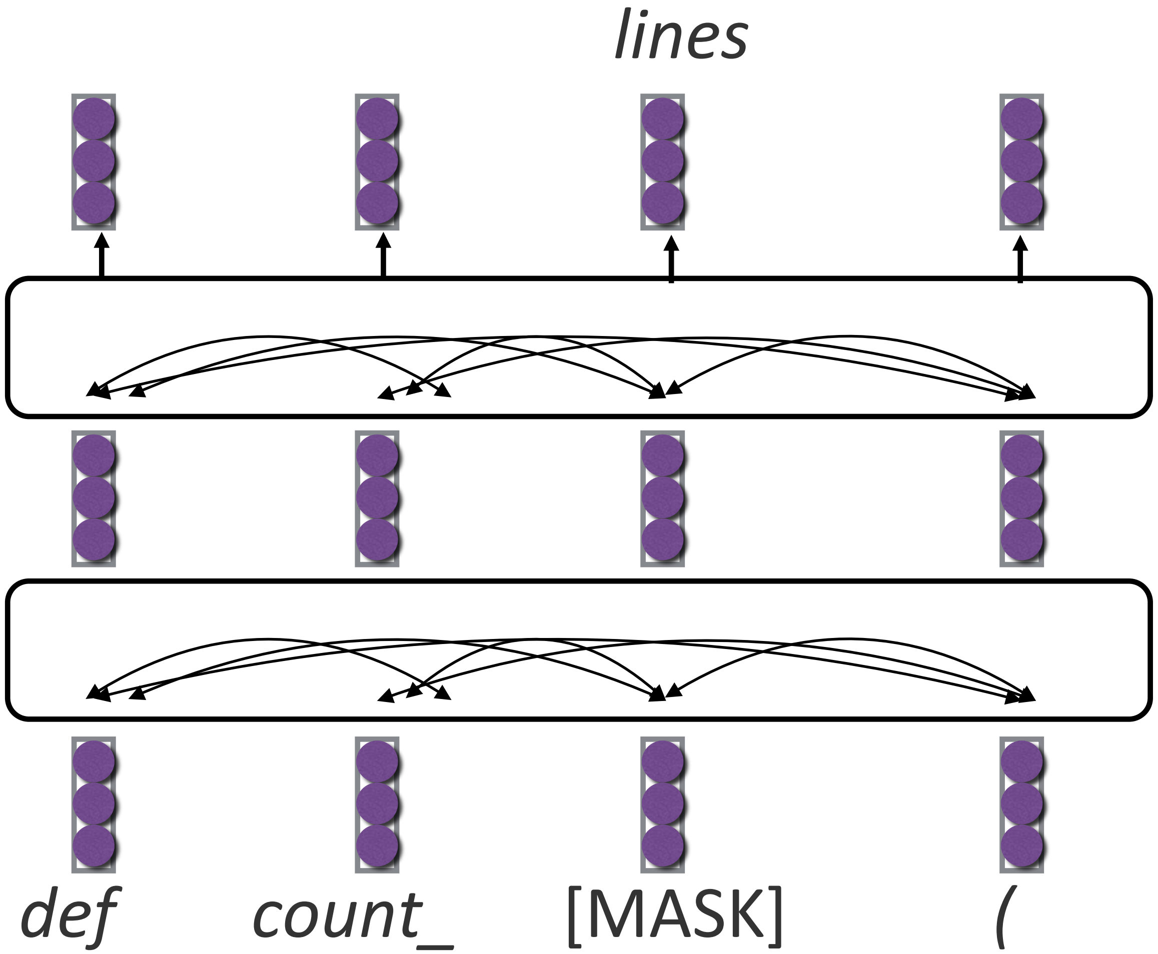
Unidirectional

Each token has info about previous.



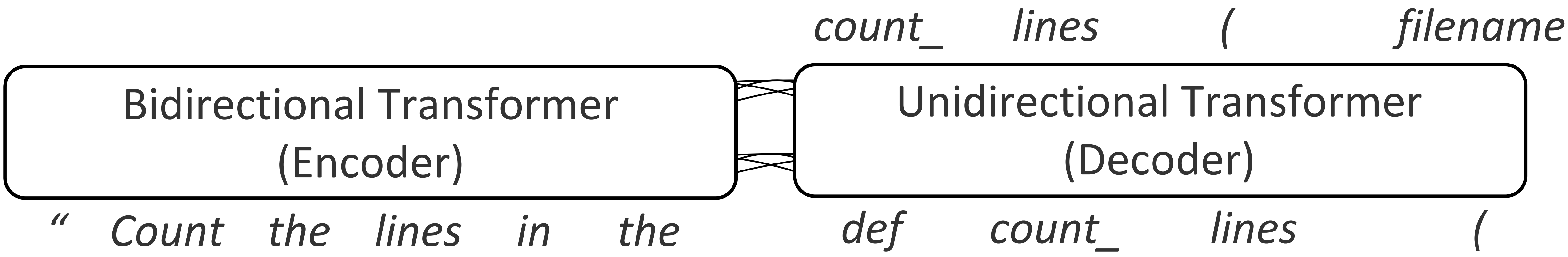
Bidirectional

Each token has info about all others.



Objectives: Sequence-to-Sequence

$$P(Y|X) = \prod_{i=1}^{|Y|} P(y_i|X, y_1, \ldots, y_{i-1})$$



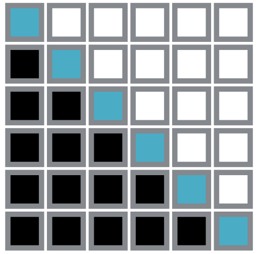
Used mostly for translation tasks, with fine-tuning.

Which Objective?

Autoregressive language modeling

$$P(X) = \prod_{i=1}^{|X|} P(x_i | x_1, \ldots, x_{i-1})$$

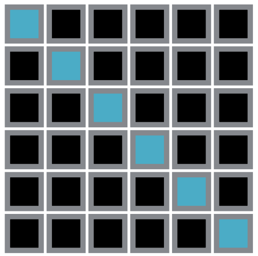
used more for prompting/text generation



Masked language modeling

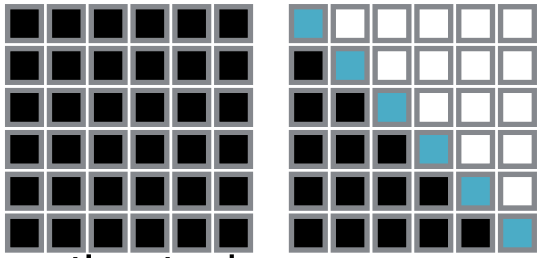
$$P(X) \neq \prod_{i=1}^{|X|} P(x_i | x_{\neq i})$$

used more for pre-training + fine-tuning



Seq-to-seq de-noising

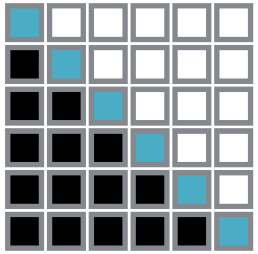
$$P(Y|X) = \prod_{i=1}^{|Y|} P(y_i | X, y_1, \ldots, y_{i-1})$$



used for pre-train + fine-tune on generation tasks

Autoregressive Generation

$$P(X) = \prod_{i=1}^{|X|} P(x_i | x_1, \ldots, x_{i-1})$$



OpenAI GPT/GPT2

- Very large language models using the Transformer architecture
- Straightforward unidirectional decoder language model, trained on raw text
- GPT2: trained on 40GB of text

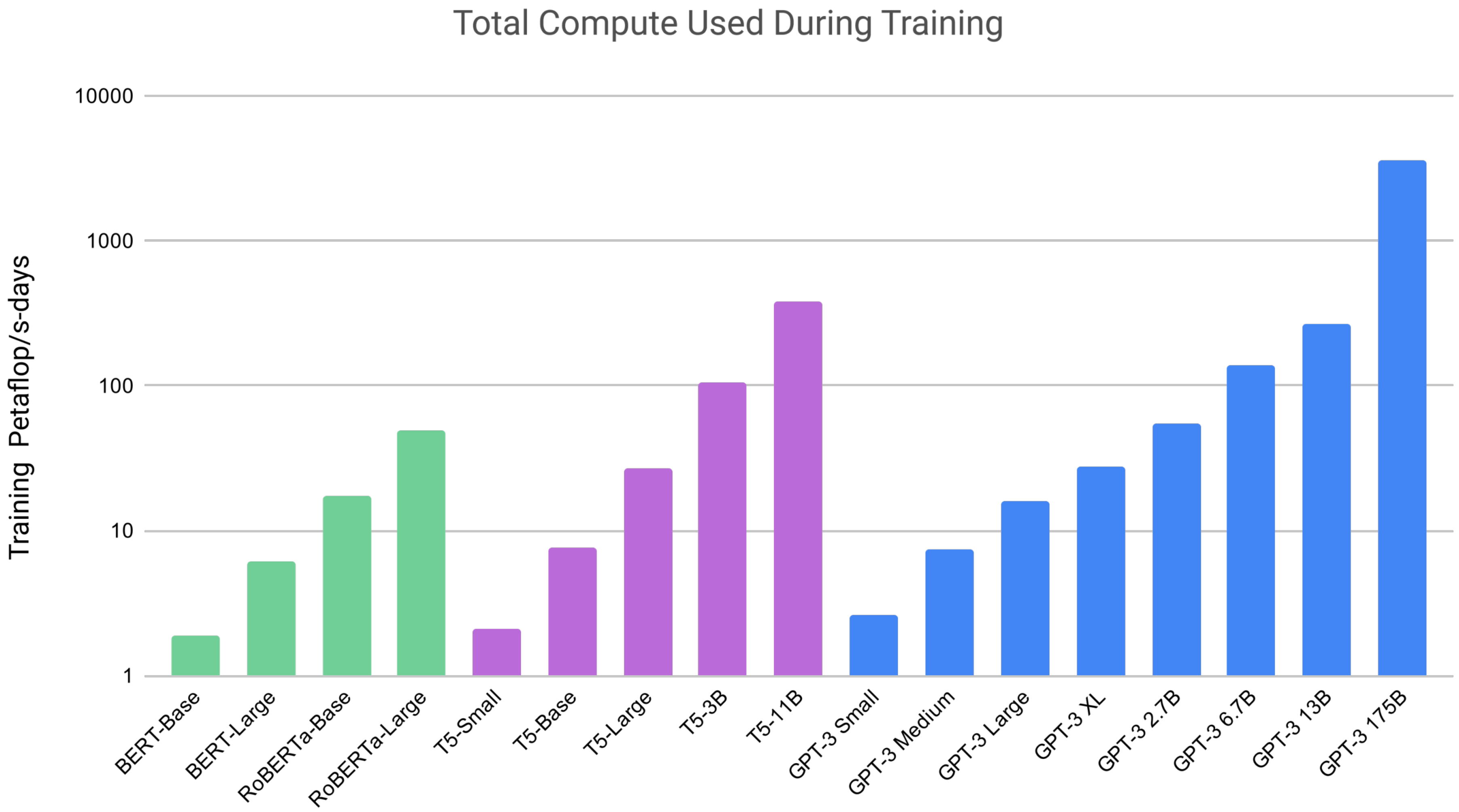
	Parameters	Layers	d_{model}
	117M	12	768
approximate size of BERT	345M	24	1024
	762M	36	1280
GPT-2	1542M	48	1600

- By far the largest of these models trained when it came out in March 2019
- Because it's a language model, we can **generate** from it

Radford et al. (2019)

Pushing the Limits: GPT-3

- 175B parameter model: 96 layers, 96 heads, 12k-dim vectors
- Trained on Microsoft Azure, estimated to cost roughly $10M



Brown et al. (2020)

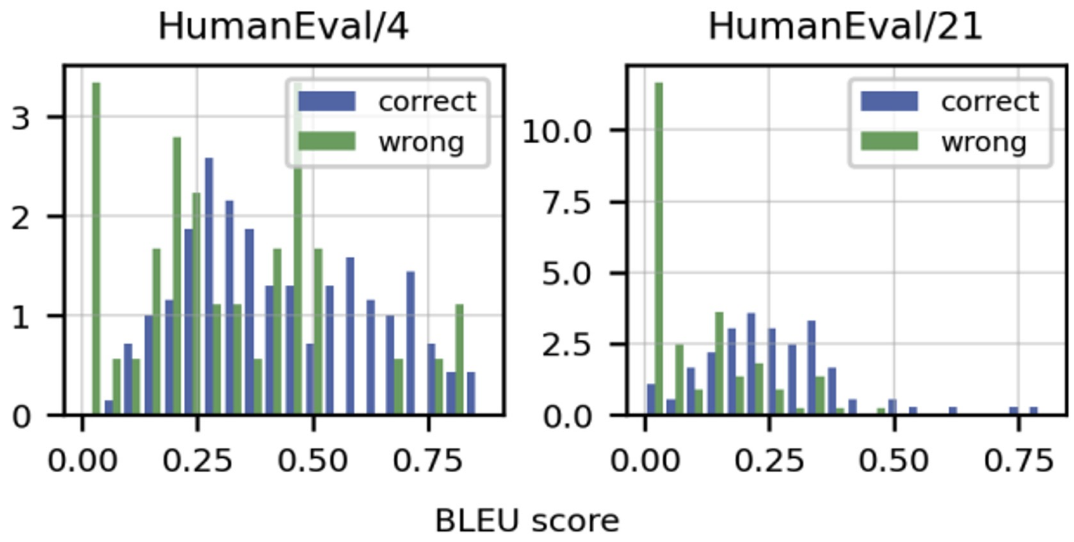
Autoregressive Language Modeling for Code

- Typically trained on lots of code from GitHub, often mixed with text
- Codex (Chen et al. 2021): OpenAI continues to train GPT-3 12B on 160GB of Python data from GitHub
- All GPT 3.5 models are trained on mixtures of code and text.
  https://platform.openai.com/docs/model-index-for-researchers
- Many open-source models since then follow this recipe (PolyCoder, CodeGen, StarCoder)

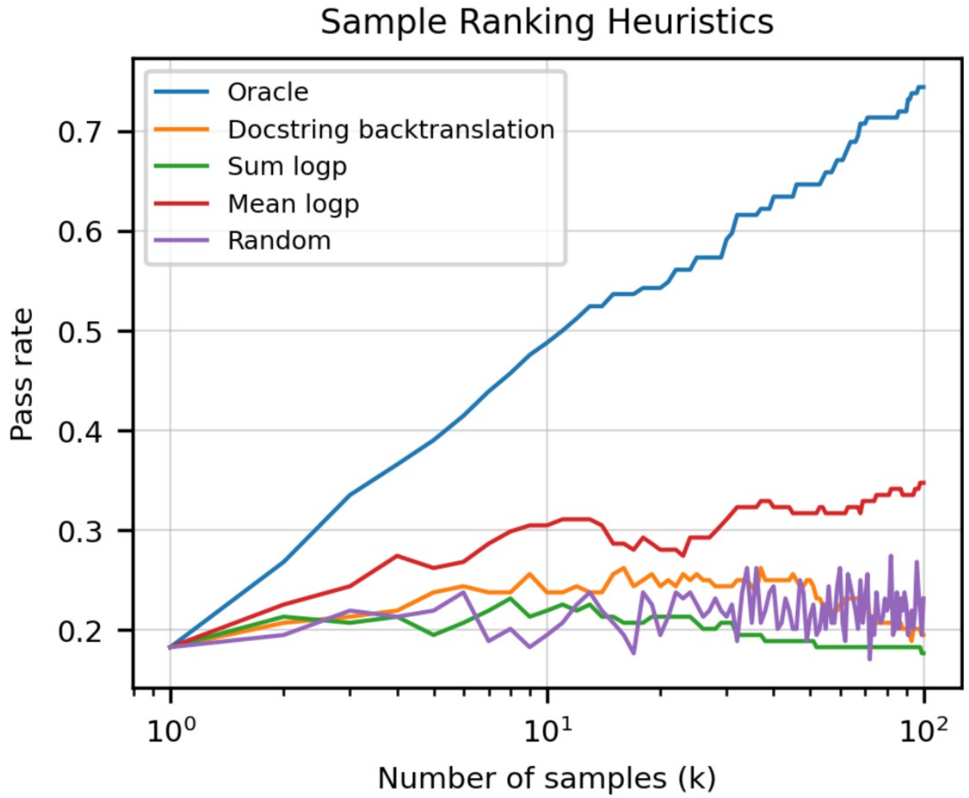
Codex: “HumanEval” Benchmark

- Evaluation: test case execution
- 164 hand-written examples
- Why human-written?
  - “It is important for these tasks to be hand-written, since our models are trained on a large fraction of GitHub, which already contains solutions to problems from a variety of sources. ”
- Optimizing BLEU != Improving Functional Correctness

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) =⇒12
    solution([3, 3, 3, 3, 3]) =⇒9
    solution([30, 13, 24, 321]) =⇒0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

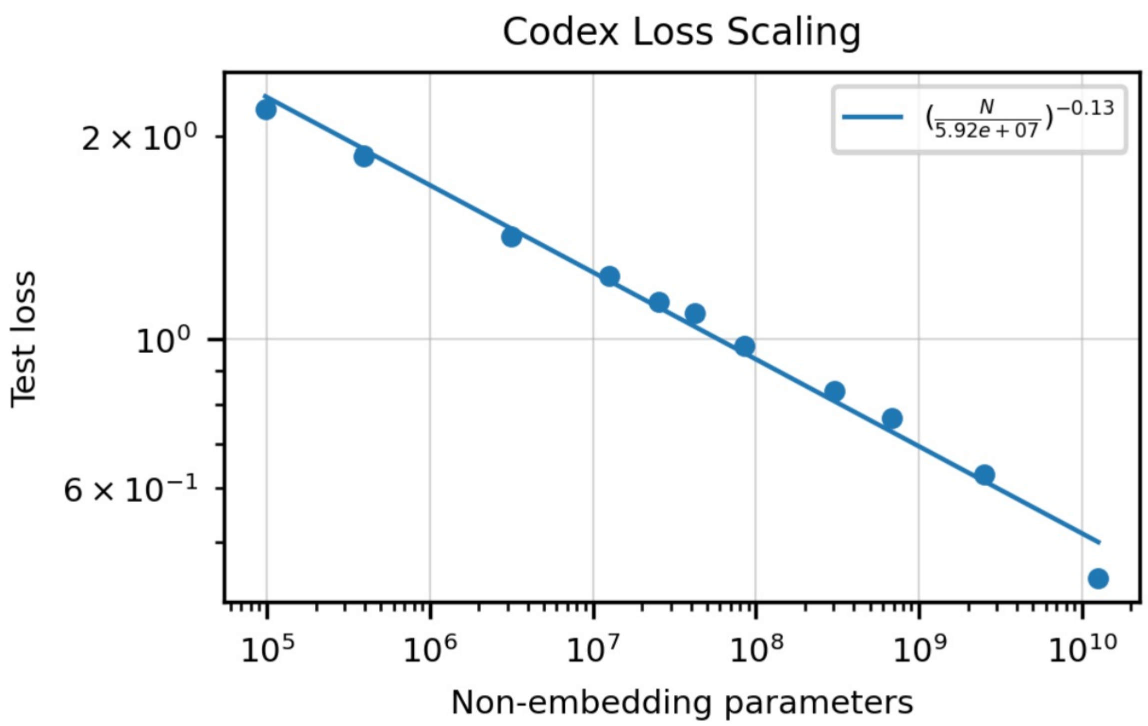


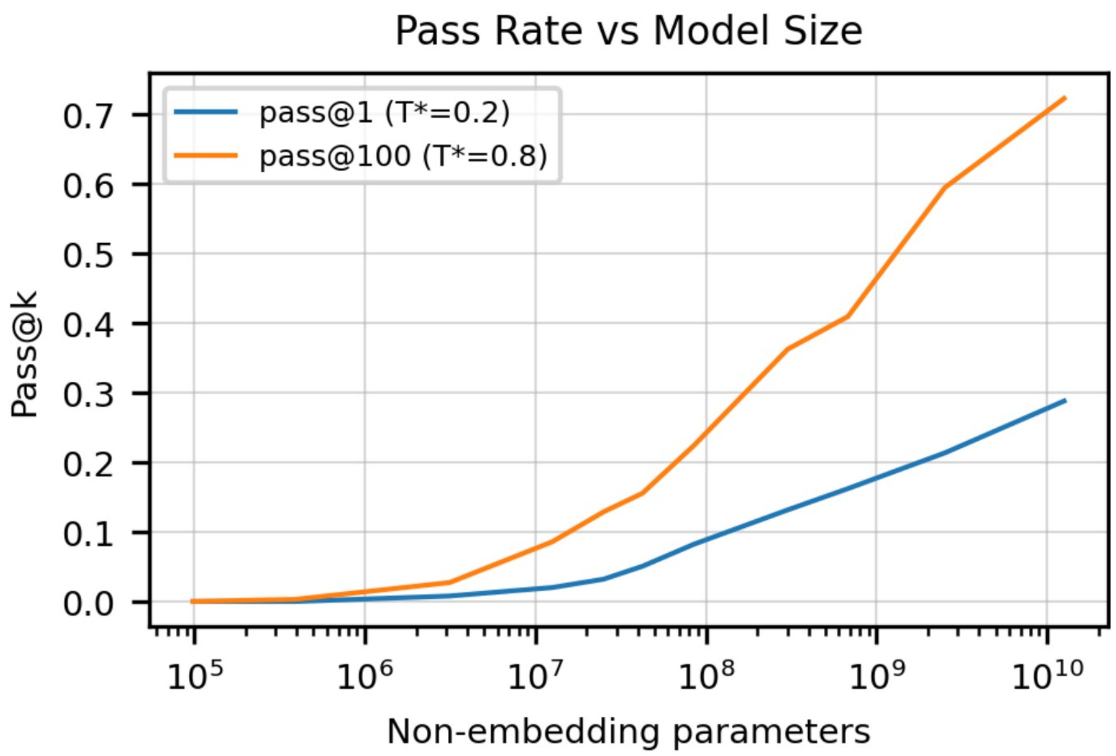
Sampling-Based Evaluation



- Sampling more candidate functions dramatically increases chance of correctness
- pass@k: sample *k* candidate functions; see if any pass
- Many ways of combining/using multiple candidates to help improve code correctness --- more in a future lecture!

Codex: Scaling Laws





Models Generate Good and Bad Code!

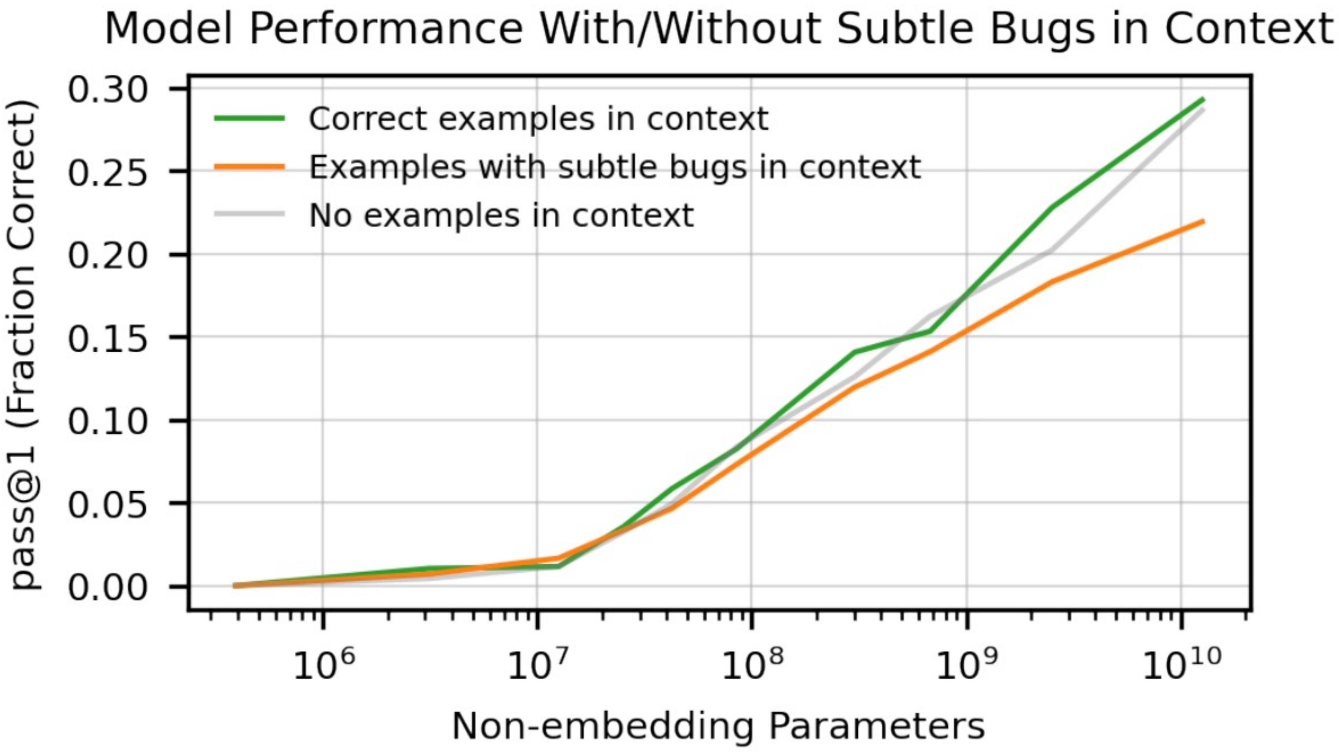
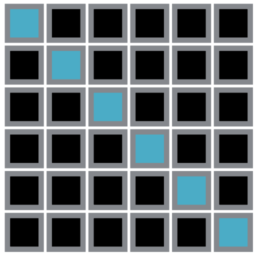


Figure 12. When the prompt includes subtle bugs, Codex tends to produce worse code than it is capable of. This persists when the prompt also includes instructions to write correct code. This gap increases with model size.

Masked Language Modeling

$$P(X) \neq \prod_{i=1}^{|X|} P(x_i | x_{\neq i})$$

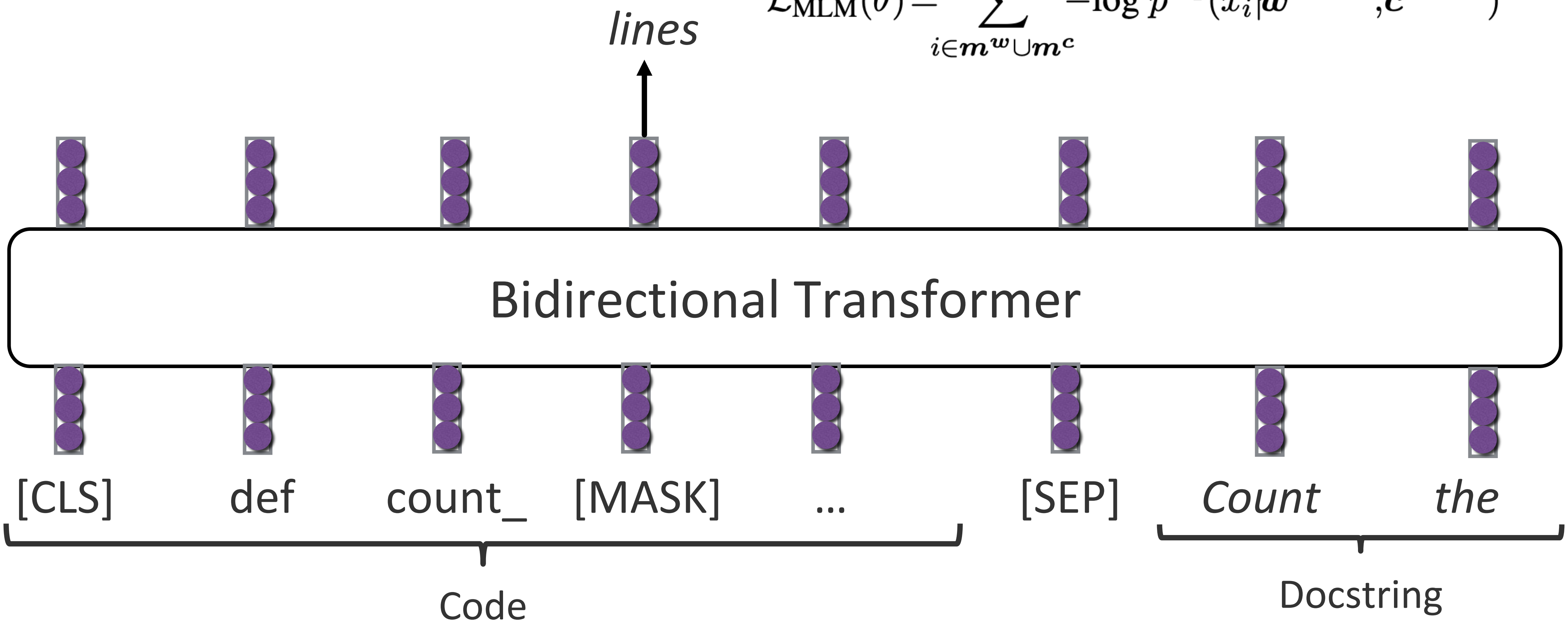


used more for pre-training + fine-tuning

CodeBERT: Masked Language Modeling Objective

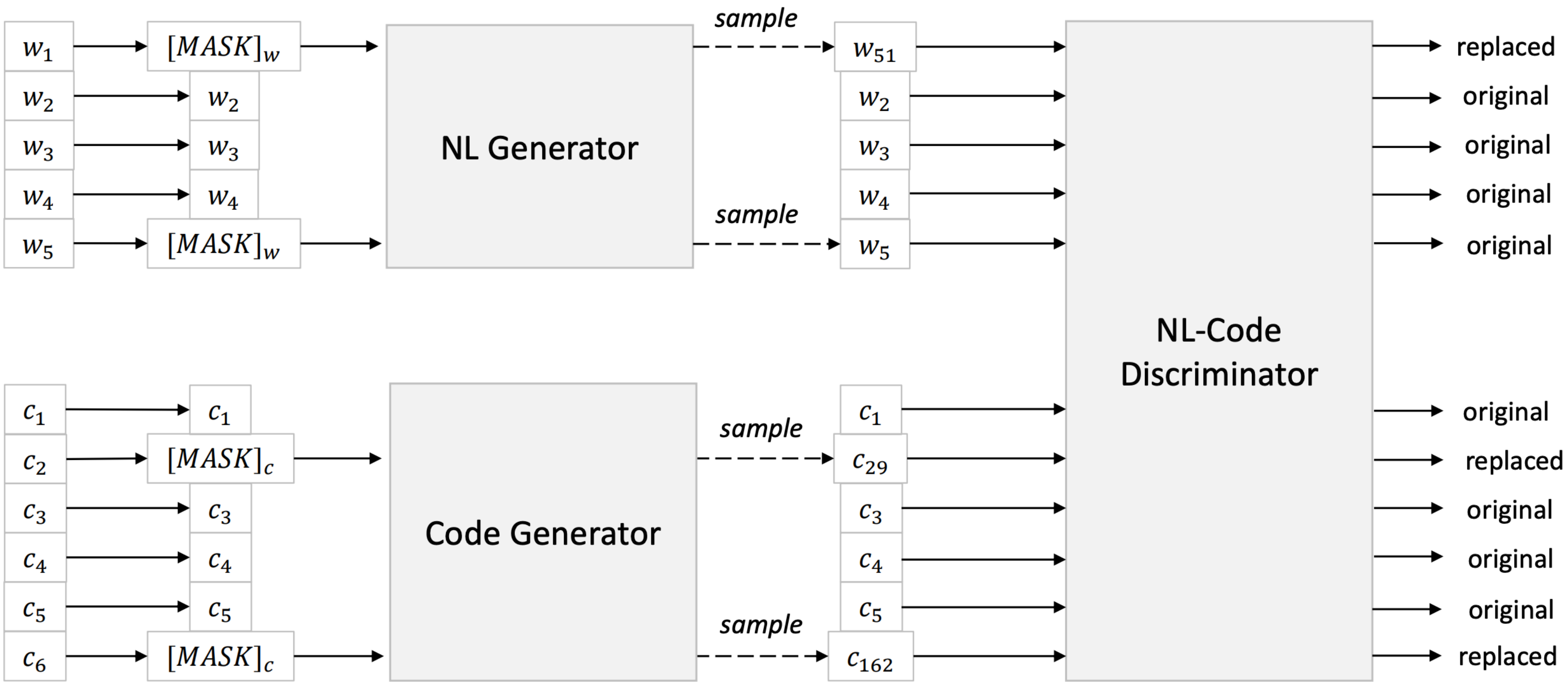
Mask 15% of the tokens, randomly, and try to predict these masked tokens.

$$\mathcal{L}_{\text{MLM}}(\theta) = \sum_{i \in \boldsymbol{m}^w \cup \boldsymbol{m}^c} -\log p^{D_1}(x_i | \boldsymbol{w}^{\text{masked}}, \boldsymbol{c}^{\text{masked}})$$



CodeBERT: Replaced Token Detection Objective

Rather than masked tokens, use tokens replaced by (weaker) LMs, and distinguish original tokens from replaced tokens.



Binary classification

CodeBERT: Pre-Training

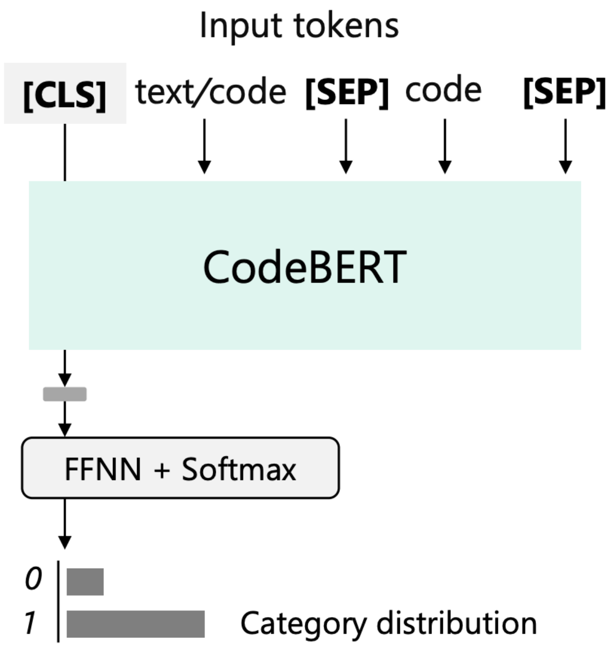
- 125M parameter bidirectional encoder Transformer
- Train on 2M documented functions (text & code) and 6M undocumented functions (code only) from GitHub (**CodeSearchNet**)

TRAINING DATA	*bimodal* DATA	*unimodal* CODES
GO	319,256	726,768
JAVA	500,754	1,569,889
JAVASCRIPT	143,252	1,857,835
PHP	662,907	977,821
PYTHON	458,219	1,156,085
RUBY	52,905	164,048
ALL	2,137,293	6,452,446

CodeBERT: Finetuning

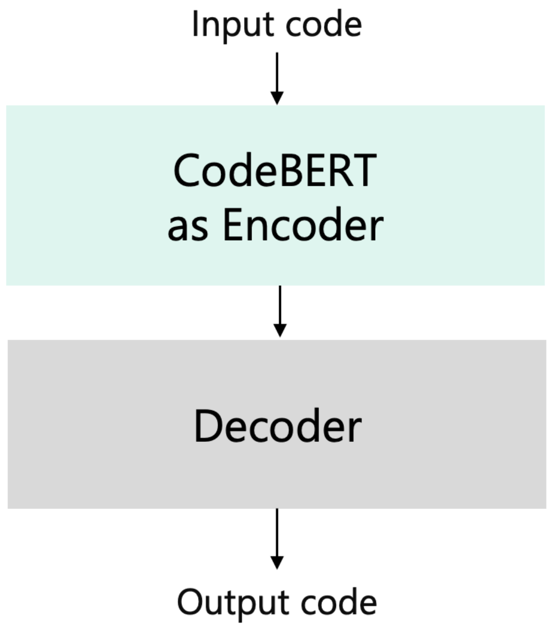
Parts of the task network are initialized with CodeBERT parameters.

Classification Tasks



Supported tasks:
- code search
- code clone detection

Generation Tasks



Supported tasks:
- code repair
- code translation

CodeXGLUE Benchmark

Collection of tasks, largely with natural data mined from GitHub

Category	Task	Dataset Name	Language	Train/Dev/Test Size	Baselines
Code-Code	Clone Detection	BigCloneBench [71]	Java	900K/416K/416K	CodeBERT
		POJ-104 [52]	C/C++	32K/8K/12K	
	Defect Detection	Devign [99]	C	21K/2.7K/2.7K	
	Cloze Test	CT-all	Python,Java,PHP, JavaScript,Ruby,Go	-/-/176K	
		CT-max/min [18]	Python,Java,PHP, JavaScript,Ruby,Go	-/-/2.6K	
	Code Completion	PY150 [62]	Python	100K/5K/50K	CodeGPT
		Github Java Corpus[4]	Java	13K/7K/8K	
	Code Repair	Bugs2Fix [75]	Java	98K/12K/12K	Encoder-Decoder
	Code Translation	CodeTrans	Java-C#	10K/0.5K/1K	
Text-Code	NL Code Search	CodeSearchNet [35], AdvTest	Python	251K/9.6K/19K	CodeBERT
		CodeSearchNet [35], WebQueryTest	Python	251K/9.6K/1K	
	Text-to-Code Generation	CONCODE [38]	Java	100K/2K/2K	CodeGPT
Code-Text	Code Summarization	CodeSearchNet [35]	Python,Java,PHP, JavaScript,Ruby,Go	908K/45K/53K	Encoder-Decoder
Text-Text	Documentation Translation	Microsoft Docs	English-Latvian/Danish /Norwegian/Chinese	156K/4K/4K	

CodeBERT: Results

- Joint training on code and documentation > code alone
- Initializing with a text-only model (RoBERTa) helps

MODEL	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	MA-AVG
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**

Results for function/documentation matching (code retrieval)

CodeBERT: Results

- Joint training on code and documentation > code alone
- Initializing with a text-only model (RoBERTa) helps

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**

Results for function-to-docstring generation

CodeBERT: Masked Prediction Probing

masked NL token

"Transforms a vector np.arange(-N, M, dx) to np.arange(min(|vec|), max(N,M),dx)]"

```
def vec_to_halfvec(vec):

    d = vec[1:] - vec[:-1]
    if ((d/d.mean()).std() > 1e-14) or (d.mean() < 0):
        raise ValueError('vec must be np.arange() in increasing order')
    dx = d.mean()
    lowest = np.abs(vec).min()
    highest = np.abs(vec).max()
    return np.arange(lowest, highest + 0.1*dx, dx).astype(vec.dtype)
```

masked PL token

		max	*min*	*less*	*greater*
NL	Roberta	96.24%	3.73%	0.02%	0.01%
	CodeBERT (MLM)	39.38%	60.60%	0.02%	0.0003%
PL	Roberta	95.85%	4.15%	-	-
	CodeBERT (MLM)	0.001%	99.999%	-	-

Figure 3: Case study on python language. Masked tokens in NL (in blue) and PL (in yellow) are separately applied. Predicted probabilities of RoBERTa and CodeBERT are given.

Filling-in-the-Middle

LLM Training Objectives

Prefix

```
def minimize_in_graph(build_loss_fn, num_steps=200, optimizer=None):
```

Target

```
  """ Minimize a loss function using gradient.
  Args:
     build_loss_fn: a function that returns a loss tensor for a mini-batch of examples.
     num_steps: number of gradient descent steps to perform.
     optimizer: an optimizer to use when minimizing the loss function. If None, will use Adam
  """
```

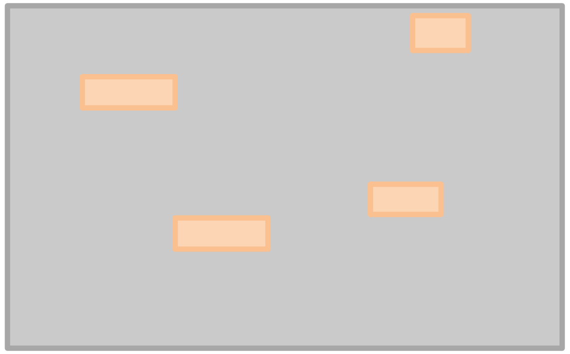
Suffix

```
  optimizer = tf.compat.v1.train.AdamOptimizer(0.1) if optimizer is None else optimizer
  minimize_op = tf.compat.v1.while_loop(
     cond=lambda step: step < num_steps,
     body=train_loop_body,
     loop_vars=[tf.constant(0)], return_same_structure=True)[0]
  return minimize_op
```

"Causal" (L-to-R)

[e.g. GPT-*, Codex]

Masked Infilling



[e.g. BERT, CodeBERT]

"Causal Masking" / Fill-in-the-Middle (FIM)

[Donahue+ 2020, Aghajanyan+ 2022, ours, Bavarian+ 2022]

Causal Masking / FIM Objective

Training

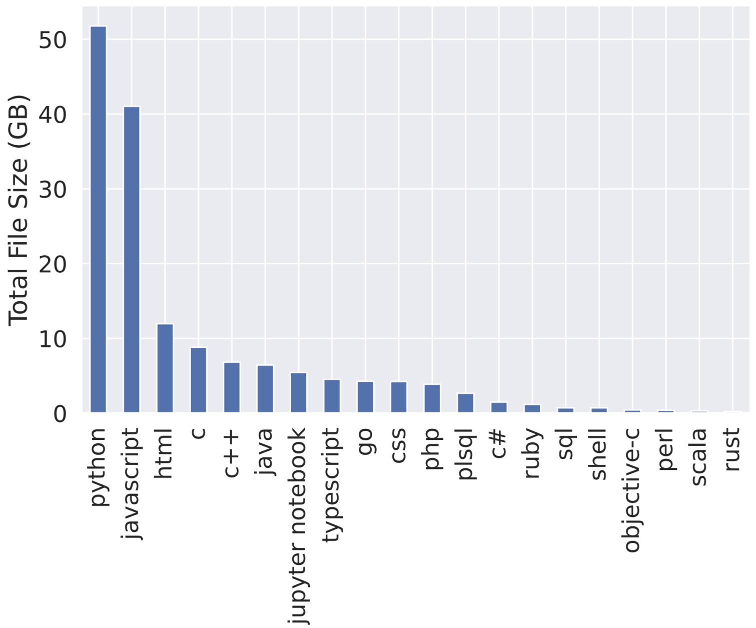
Original Document

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

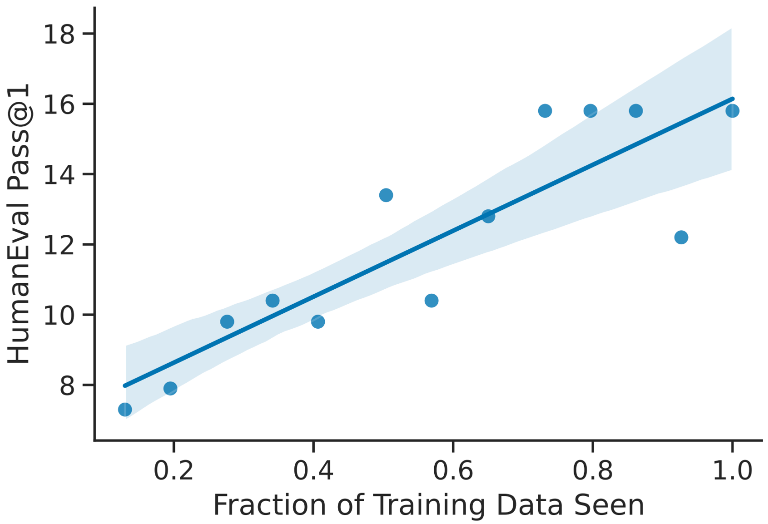
[Donahue et al. 2020, Aghajanyan et al. 2022, Fried et al. 2022, Bavarian et al. 2022]

InCoder: Model Training

- Training Data
  - 600K permissively-licensed repositories from GitHub & GitLab. ~150GB total
  - StackOverflow: questions, answers, comments. ~50GB



- Models
  - Unidirectional, decoder-only Transformer
  - 1B model: ~1 week on 128 V100s
  - 6B model: ~3 weeks on 240 V100s



Zero-Shot Software Tasks via Infilling

Zero-shot Inference

Docstring Generation

```python
def count_words(filename: str) -> Dict[str, int]:
    """
    Counts the number of occurrences of each word in the given file.

    :param filename: The name of the file to count.
    :return: A dictionary mapping words to the number of occurrences.
    """
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

Multi-Region Infilling

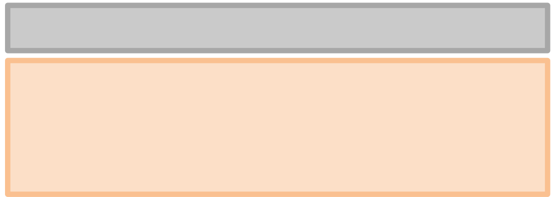
```python
from collections import Counter


def word_count(file_name):
    """Count the number of occurrences of each word in the file."""
    words = []
    with open(file_name) as file:
        for line in file:
            words.append(line.strip())
    return Counter(words)
```

Evaluation

- Zero-shot evaluation on several software development-inspired code infilling tasks (we'll show two).
- Compare the model in three different modes to evaluate benefits of suffix context

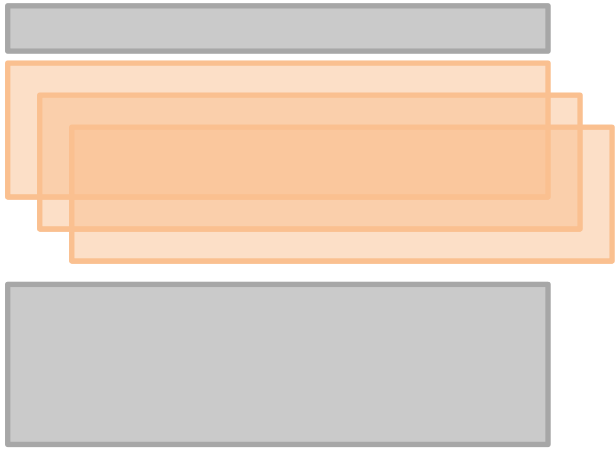
Baselines

Left-to-Right Single



Doesn't use suffix

Left-to-Right Rerank



Only uses suffix when reranking

Ours

Causal Masking

Uses suffix when generating

Evaluation: Function Completion

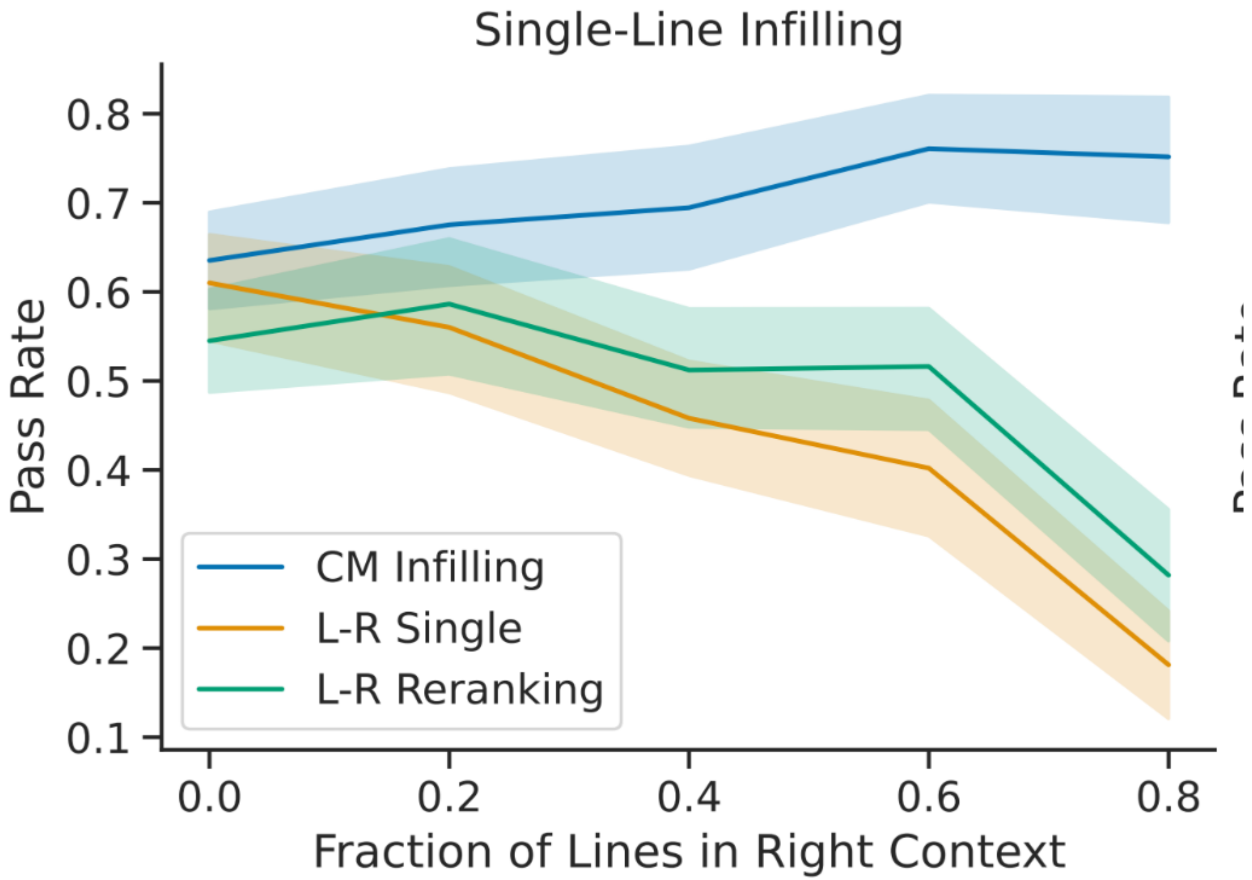
Fill in one or more lines of a function; evaluate with unit tests.

```python
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """
    Check if in given list of numbers, are any two numbers closer to each other
        than given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True
    return False
```

Method	Pass Rate	Exact Match
L-R single	24.9	15.8
L-R reranking	28.2	17.6
CM infilling	38.6	20.6

Constructed from HumanEval [Chen et al. 2021]

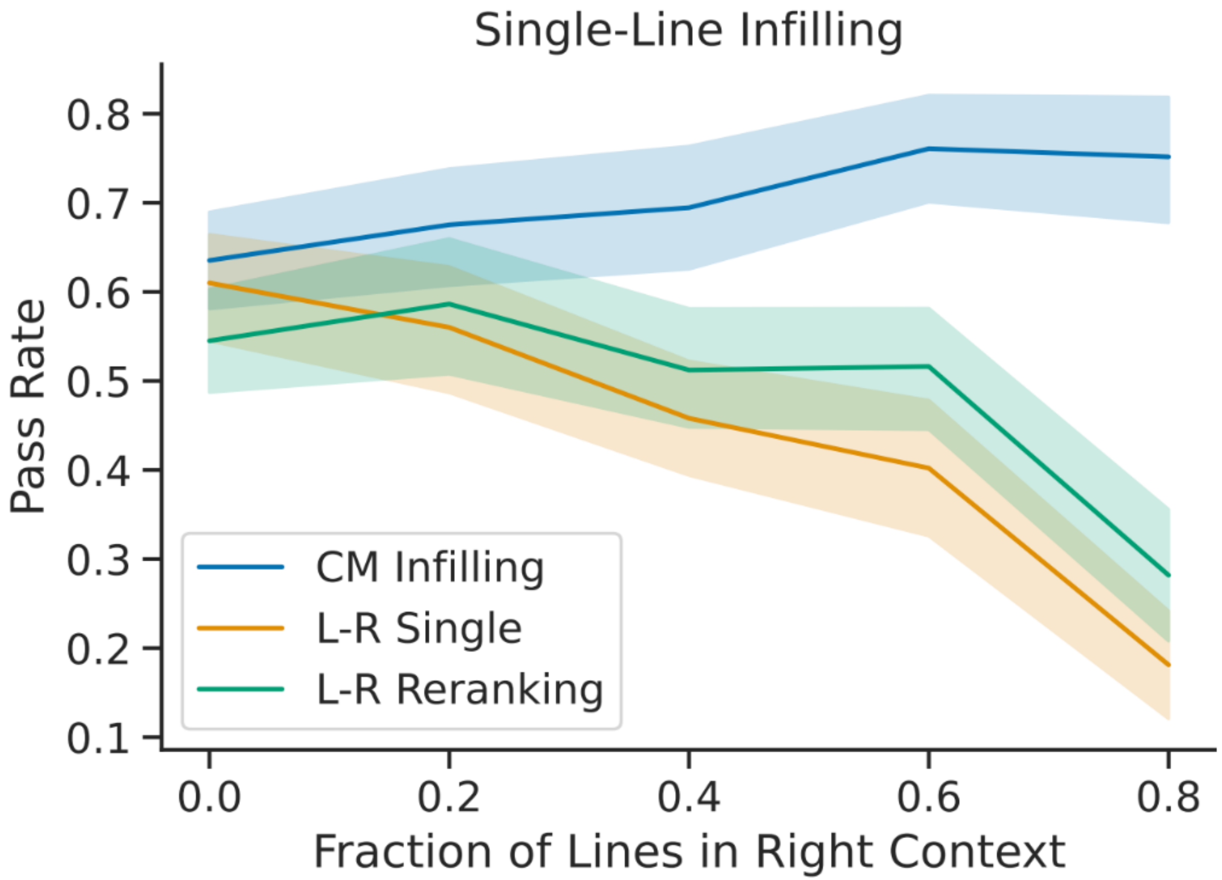
Function completion

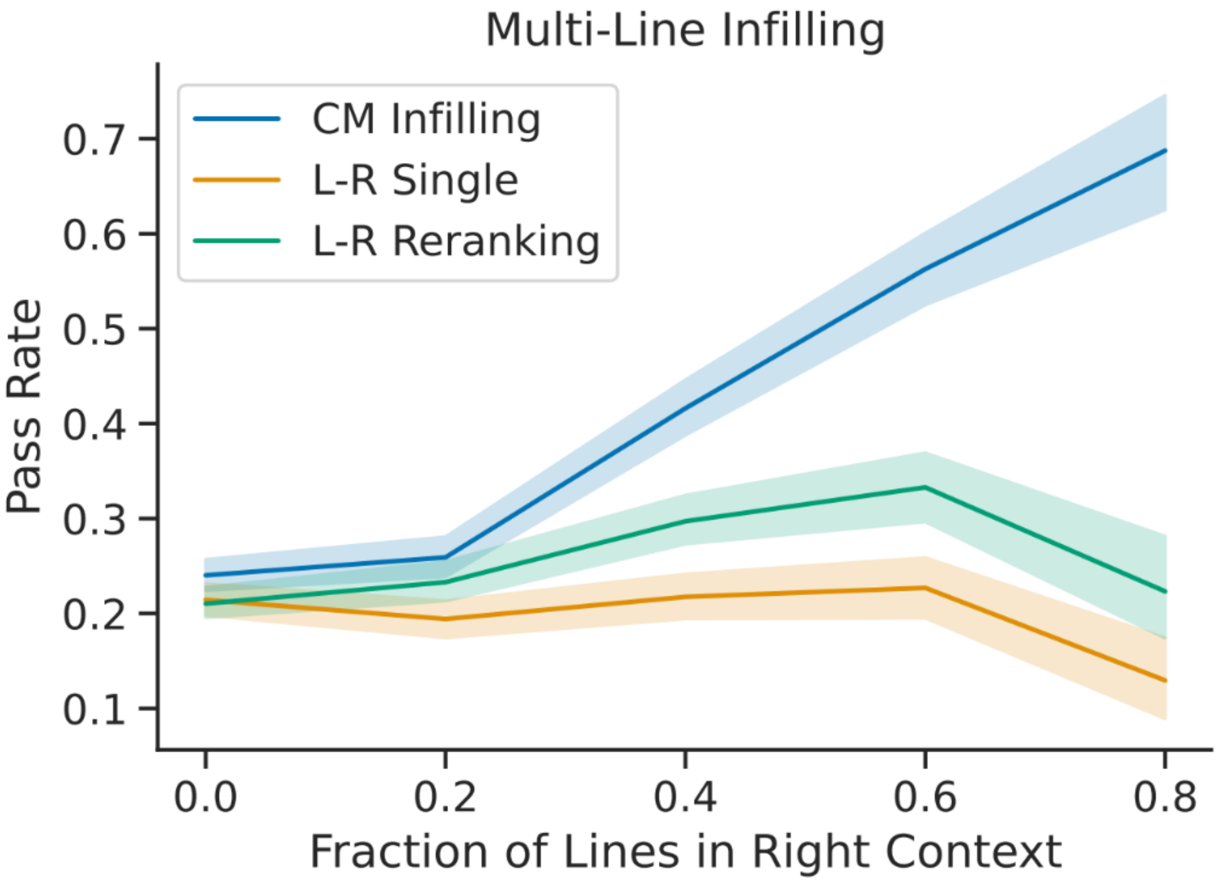



```python
def count_words(filename):
    """Count the number of occurrences of each word in the file"
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```

```python
def count_words(filename):
    """Count the number of occurrences of each word in the file"""
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```

Function completion






```python
def count_words(filename):
    """Count the number of occurrences of each word in the file"
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```

```python
def count_words(filename):
    """Count the number of occurrences of each word in the file"""
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```

```python
def count_words(filename):
    """Count the number of occurrences of each word in the file"""
    words = {}
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words
```

Evaluation: Docstring Generation

```python
def count_words(filename: str) -> Dict[str, int]:
    """
    Counts the number of occurrences of each word in the given file.

    :param filename: The name of the file to count.
    :return: A dictionary mapping words to the number of occurrences.
    """
    with open(filename, 'r') as f:
            word_counts = {}
            for line in f:
                for word in line.split():
                    if word in word_counts:
                        word_counts[word] += 1
                    else:
                        word_counts[word] = 1
        return word_counts
```

Method	BLEU
Ours: L-R single	16.05
Ours: L-R reranking	17.14
Ours: Causal-masked infilling	18.27

[CodeXGlue, Lu et al. 2021]

Evaluation: Return Type Prediction

Type Inference

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

Method	F1
Ours: Left-to-right single	30.8
Ours: Left-to-right reranking	33.3
Ours: Causal-masked infilling	**59.2**
TypeWriter (Supervised)	48.3

[TypeWriter OSS, Pradel et al. 2020]

Evaluation

Variable Name Prediction

```python
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_count = {}
        for line in f:
            for word in line.split():
                if word in word_count:
                    word_count[word] += 1
                else:
                    word_count[word] = 1
    return word_count
```

Method	Accuracy
Left-to-right single	18.4
Left-to-right reranking	23.5
Causal-masked infilling	30.6

Ablations

- StackOverflow data improves performance
- Roughly comparable performance from infilling and non-infilling models (but see Ben Allal et al. 2022 and Nijkamp et al. 2023)

#	Size (B)	Obj.	Training Data	Data Size	Train Tokens	HumanEval Pass@1	MBPP Pass@1
1)	6.7	CM	multi lang + SO	204 GB	52 B	15	19.4
2)	1.3	CM	multi lang + SO	204 GB	52 B	8	10.9
3)	1.3	LM	multi lang + SO	204 GB	52 B	6	8.9
4)	1.3	LM	Python + SO	104 GB	25 B	9	9.8
5)	1.3	LM	Python	49 GB	11 B	5	6.1

Other Infilling Code Models

Efficient Training of Language Models to Fill in the Middle

Mohammad Bavarian * Heewoo Jun * Nikolas Tezak
John Schulman Christine McLeavey Jerry Tworek Mark Chen
OpenAI

SANTACODER: DON'T REACH FOR THE STARS!

Loubna Ben Allal* Hugging Face
Raymond Li* ServiceNow Research
Denis Kocetkov* ServiceNow Research

StarCoder: may the source be with you!

Raymond Li[2] Loubna Ben Allal[1] Yangtian Zi[4] Niklas Muennighoff[1] Denis Kocetkov[2]
Chenghao Mou[5] Marc Marone[8] Christopher Akiki[9,10] Jia Li[5] Jenny Chim[11] Qian Liu[13]

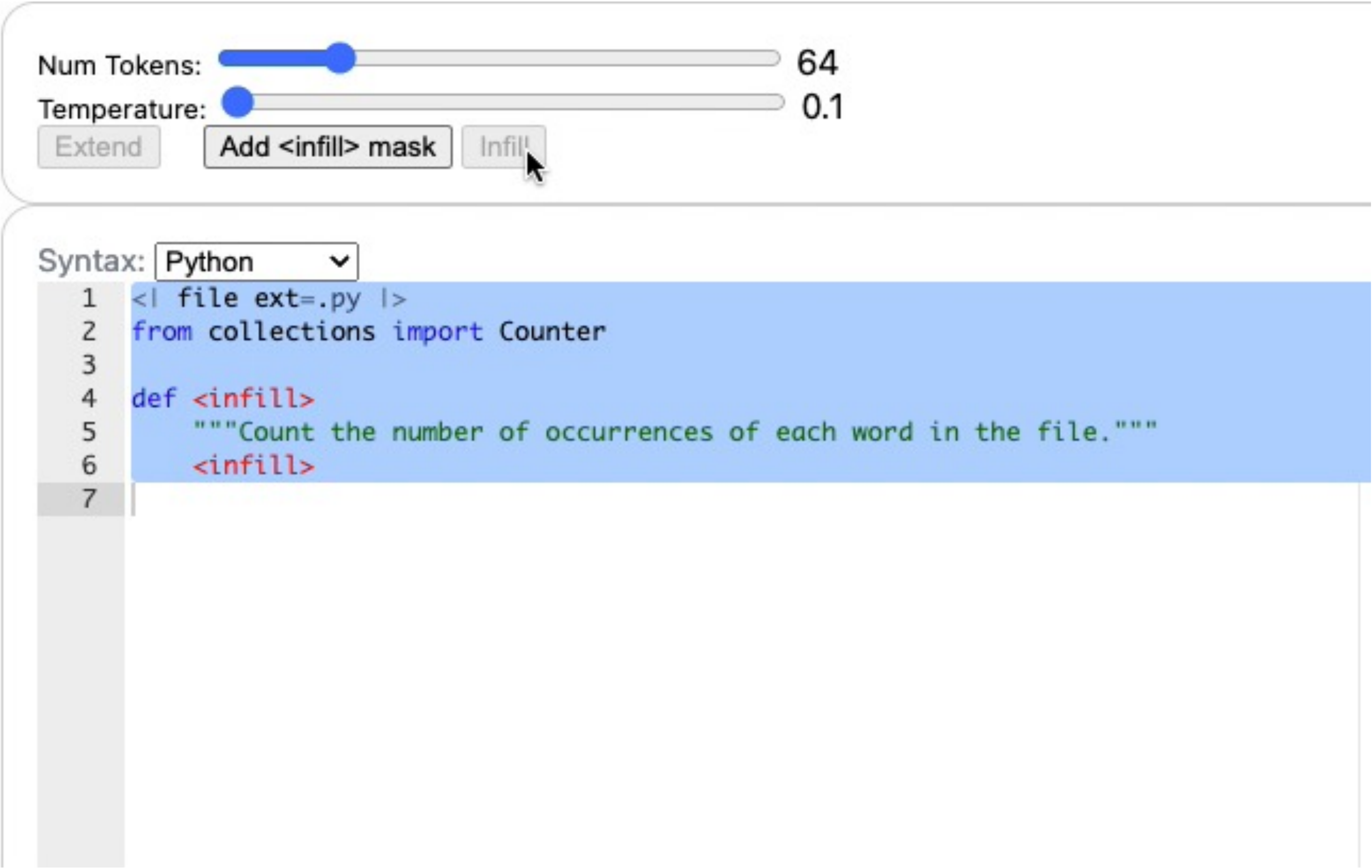
Code Llama: Open Foundation Models for Code

Baptiste Rozière[†], Jonas Gehring[†], Fabian Gloeckle[†,*], Sten Sootla[†], Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, Gabriel Synnaeve[†]

CODEGEN2: LESSONS FOR TRAINING LLMS ON PROGRAMMING AND NATURAL LANGUAGES

Erik Nijkamp*, Hiroaki Hayashi*, Caiming Xiong, Silvio Savarese, Yingbo Zhou

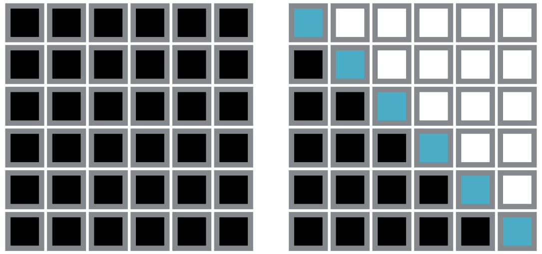
Demo



Demo: huggingface.co/spaces/facebook/incoder-demo

Encoder-Decoder LMs

$$P(Y|X) = \prod_{i=1}^{|Y|} P(y_i|X, y_1, \ldots, y_{i-1})$$

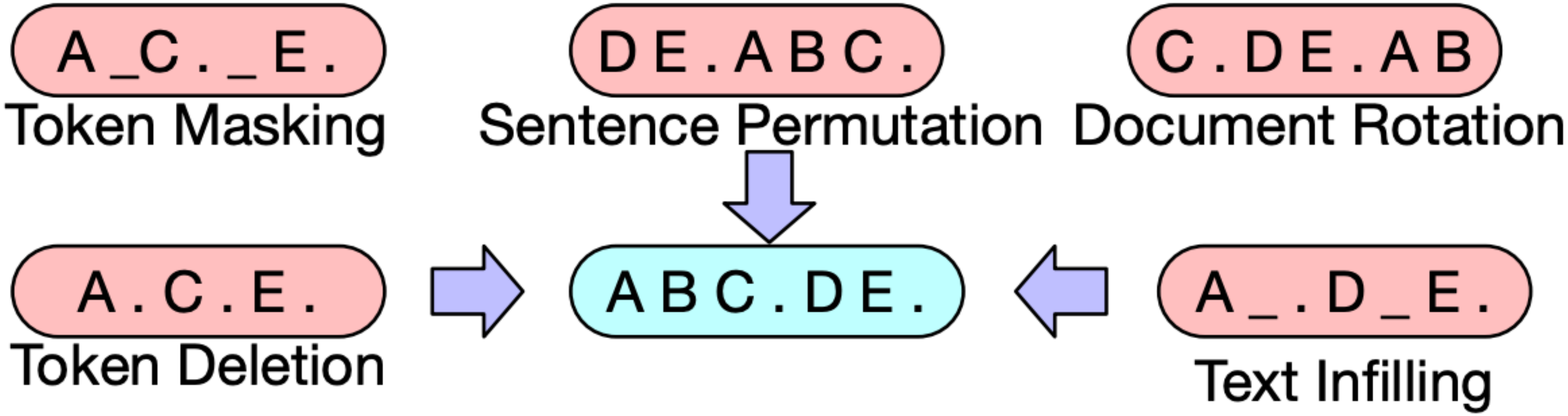


used for pre-train + fine-tune on generation tasks

How do we pre-train seq2seq models?

- LMs P(**x**): trained unidirectionally
- Masked LMs: trained bidirectionally but with masking
- How can we pre-train a model for P(**y**|**x**)?
- Well, why was BERT effective?
  - Predicting a mask requires some kind of text “understanding”.
- What would it take to do the same for sequence prediction?
- Requirements: (1) should use unlabeled data; (2) should force a model to attend from **y** back to **x**

BART



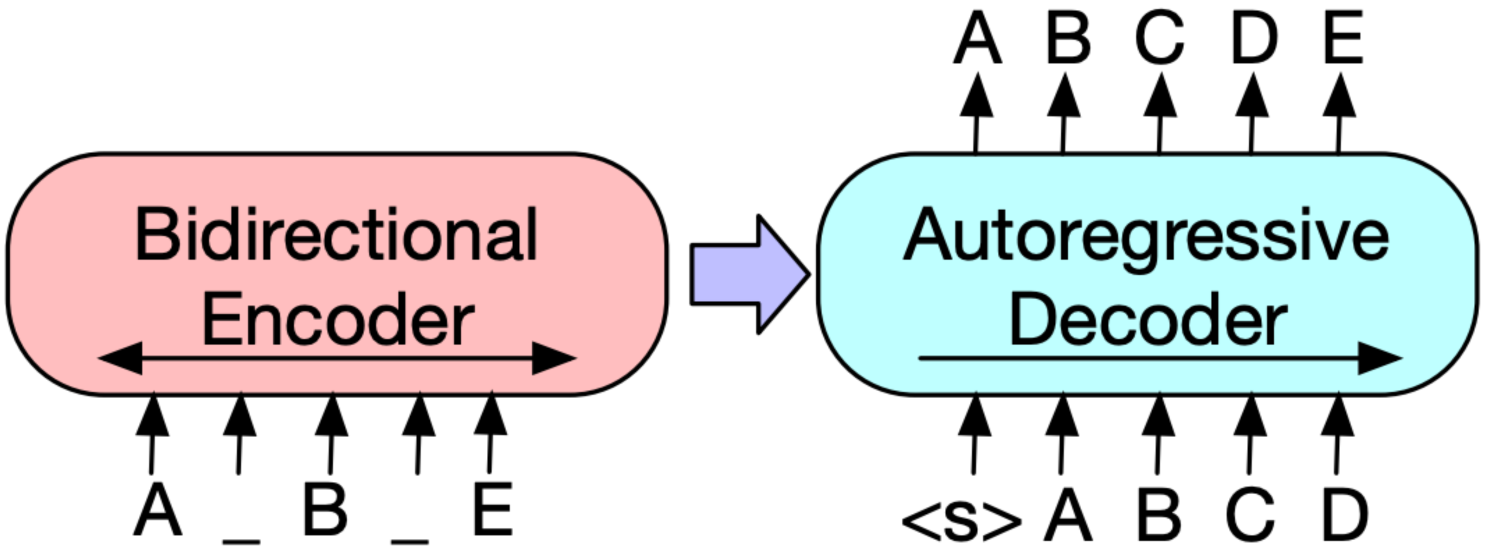
Infilling is longer spans than masking

- Several possible strategies for corrupting a sequence are explored in the BART paper

Lewis et al. (2019)

BART

- **Model & Objective**: Sequence-to-sequence Transformer trained on this data: permute/make/delete tokens, then predict full sequence autoregressively

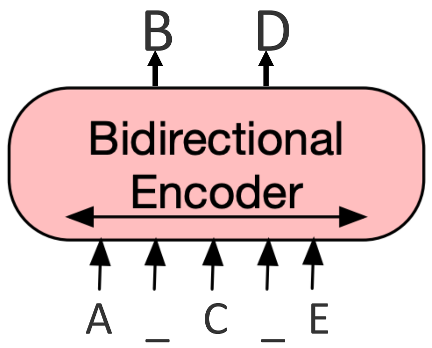


- **Data**: Same as RoBERTa; 160 GB of text

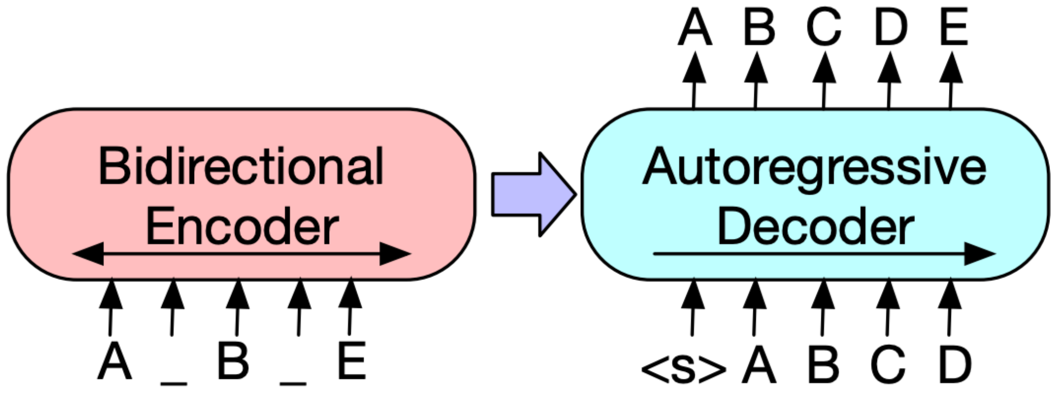
Lewis et al. (2019)

BERT vs. BART

- BERT: only parameters are an encoder, trained with masked language modeling objective. Cannot generate text or do seq2seq tasks



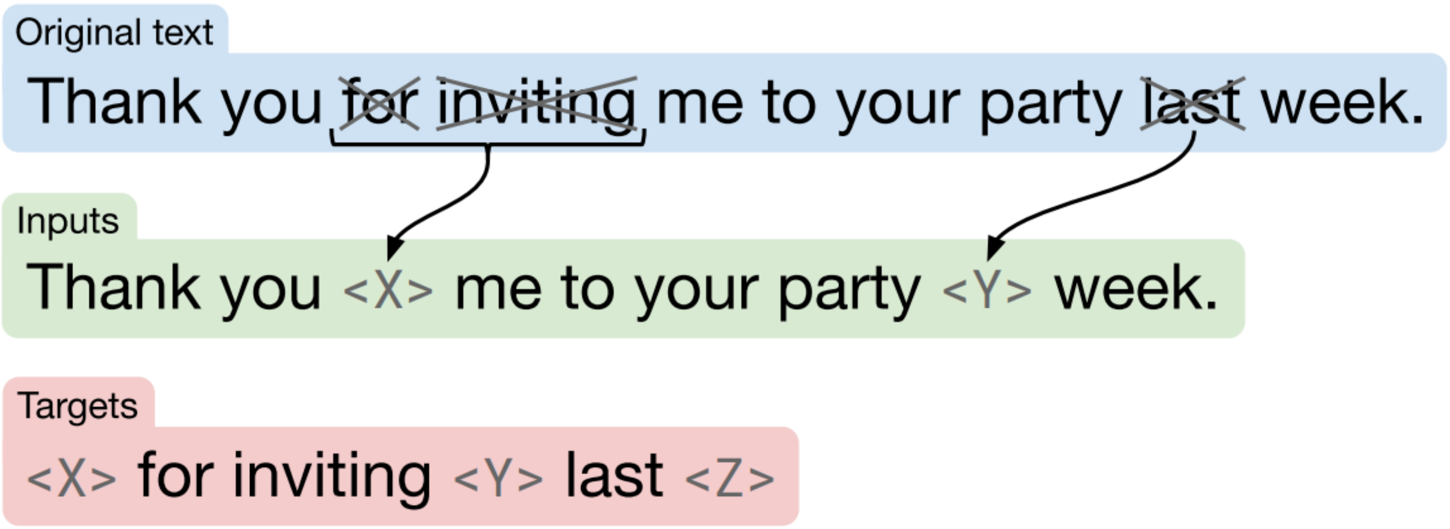
- BART: both an encoder and a decoder. Can also use just the encoder wherever we would use BERT



Lewis et al. (2019)

T5: Text-to-Text Transfer Transformer

- **Objective**: similar denoising scheme to BART (they were released within a week of each other in fall 2019).
- Input: text with gaps. Output: a series of phrases to fill those gaps.
- Lower computational cost compared to BART: predicts fewer tokens.



Raffel et al. (2019)

CodeT5: Objectives

Pre-train like T5 (seq-to-seq denoising/masked span prediction), but add identifier-specific objectives to learn code semantics.

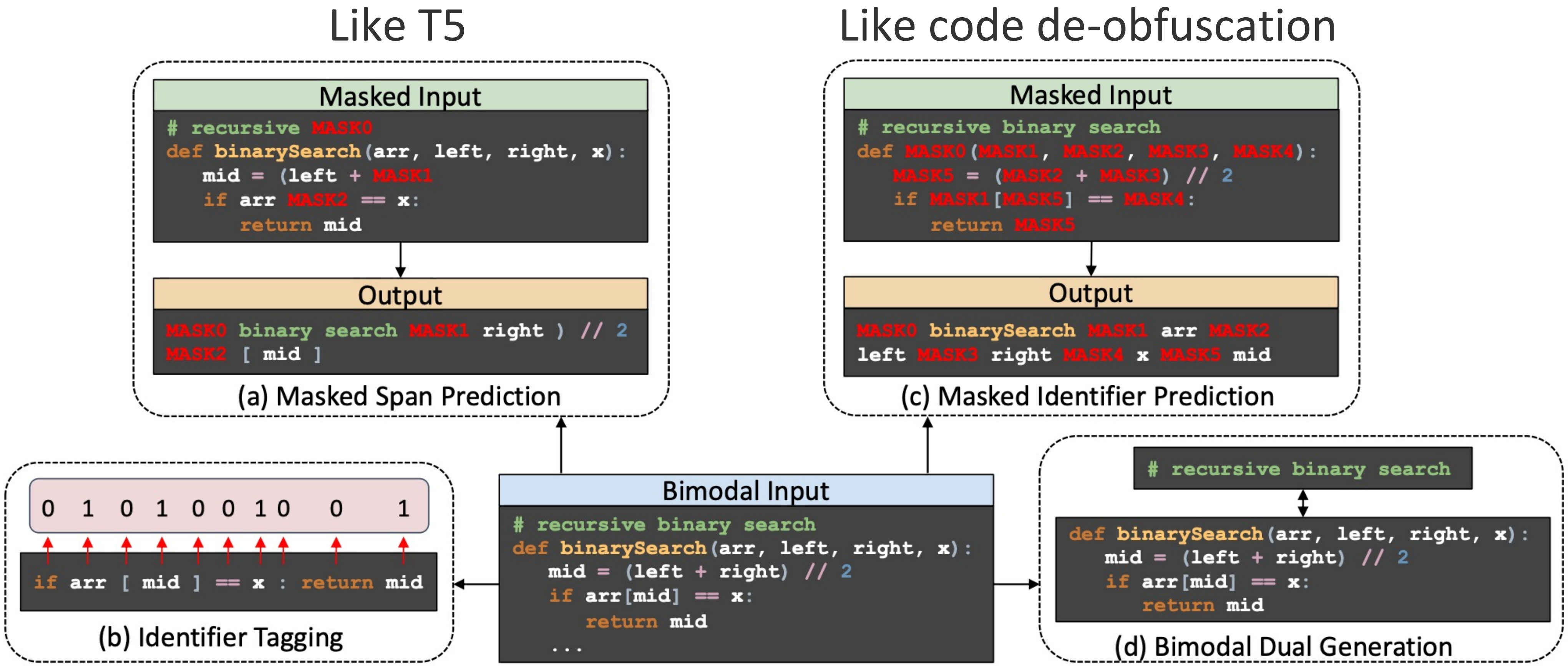


Figure 2: Pre-training tasks of CodeT5. We first alternately train span prediction, identifier prediction, and identifier tagging on both unimodal and bimodal data, and then leverage the bimodal data for dual generation training.

Wang et al. (2021)

CodeT5: Training

- Pre-train on CodeSearchNet (6 PLs) + BigQuery (C & C#); 8.4M instances
  - 60M and 220M parameter models, trained for 5 & 12 days on 16 GPUs.
  - Couldn't initialize with T5, because T5's tokenizer doesn't preserve code-specific symbols like { and }. Train own tokenizer (more in a future class!)
- Then, optionally do **multi-task fine-tuning**: train on multiple seq-to-seq tasks from CodeXGLUE simultaneously (translation, refinement, summarization, …).

CodeT5: Analysis

- All components of the objective help. MSP: masked span prediction. IT: identifier tagging. MIP: masked identifier prediction

Methods	Sum-PY (BLEU)	Code-Gen (CodeBLEU)	Refine Small (EM)	Defect (Acc)
CodeT5	20.04	41.39	19.06	63.40
-MSP	18.93	37.44	15.92	64.02
-IT	19.73	39.21	18.65	63.29
-MIP	19.81	38.25	18.32	62.92

CodeT5: Analysis

- Multi-task fine-tuning sometimes helps and sometimes hurts, with some effects from task similarity.

Methods	Java to C#		C# to Java		Refine Small		Refine Medium	
	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM
CodeBERT	79.92	59.00	72.14	58.80	77.42	16.40	91.07	5.20
GraphCodeBERT	80.58	59.40	72.64	58.80	**80.02**	17.30	**91.31**	9.10
PLBART	83.02	64.60	78.35	65.00	77.02	19.21	88.50	8.98
CodeT5-small	82.98	64.10	79.10	65.60	76.23	19.06	89.20	10.92
+dual-gen	82.24	63.20	78.10	63.40	77.03	17.50	88.99	10.28
+multi-task	83.49	64.30	78.56	65.40	77.03	20.94	87.51	11.11
CodeT5-base	**84.03**	**65.90**	**79.87**	**66.90**	77.43	21.61	87.64	13.96
+dual-gen	81.84	62.00	77.83	63.20	77.66	19.43	90.43	11.69
+multi-task	82.31	63.40	78.01	64.00	78.06	**22.59**	88.90	**14.18**

Code translation and refinement results.

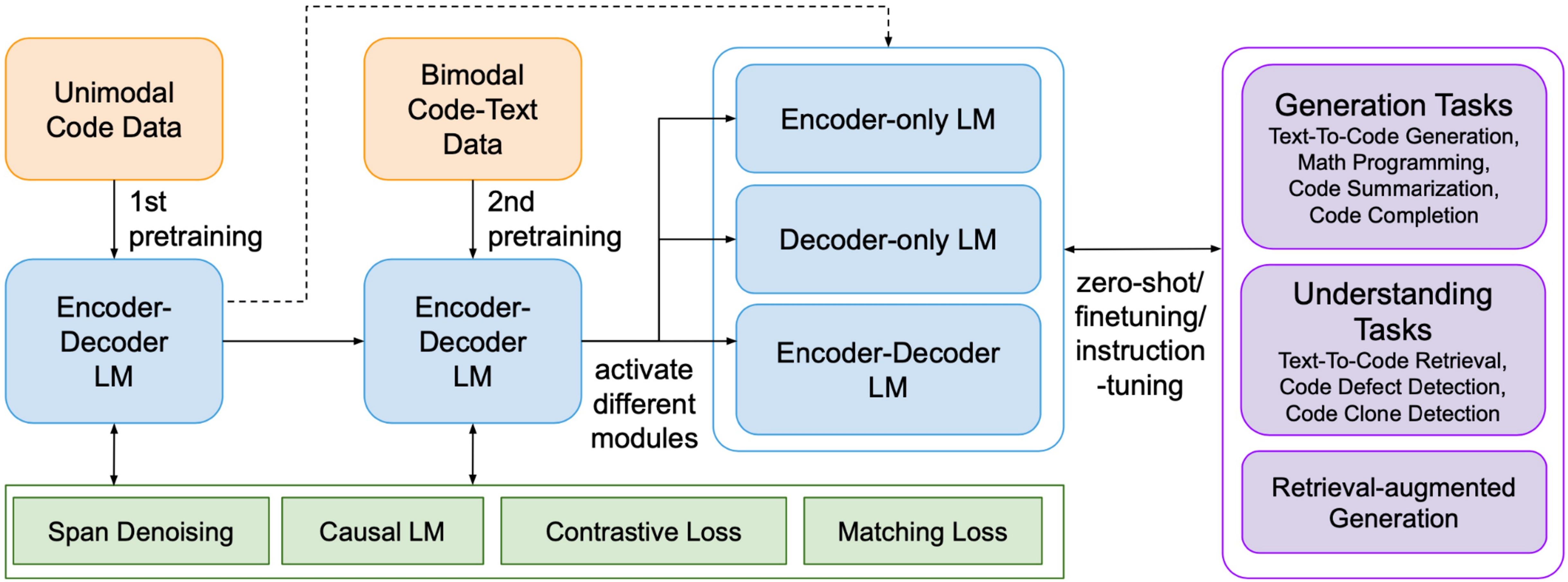
Hybrid Models

CodeT5+

- Specializations of past approaches:
  - For **translation**: T5-like (seq-to-seq denoising) generally best
  - For **generating new content**: GPT-like (unidirectional decoder-only) generally best
  - For **doc-level embeddings**: BERT-like (MLM bidirectional encoder) generally best
- CodeT5+: use a seq-to-seq model but train it with a progression of objectives, and pre-trained initializations

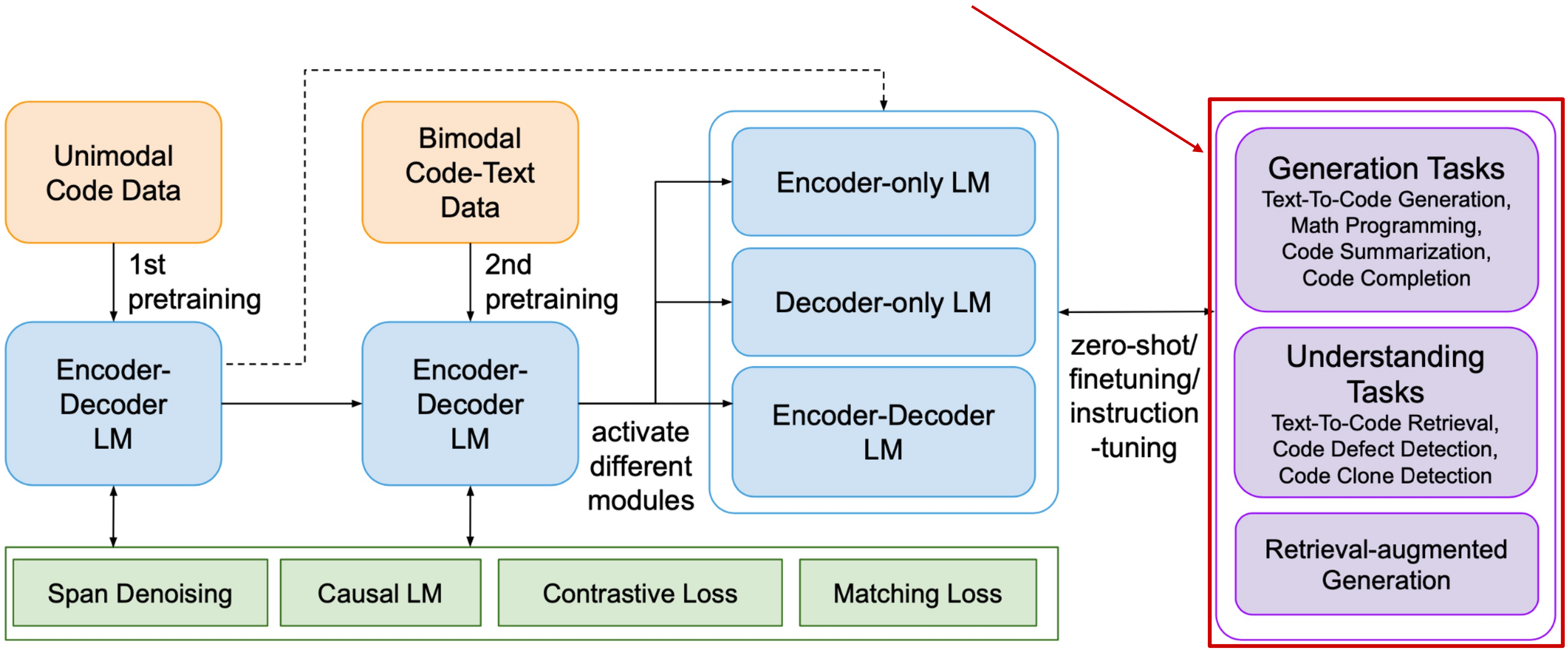
Wang et al. (2023)

CodeT5+: Overview

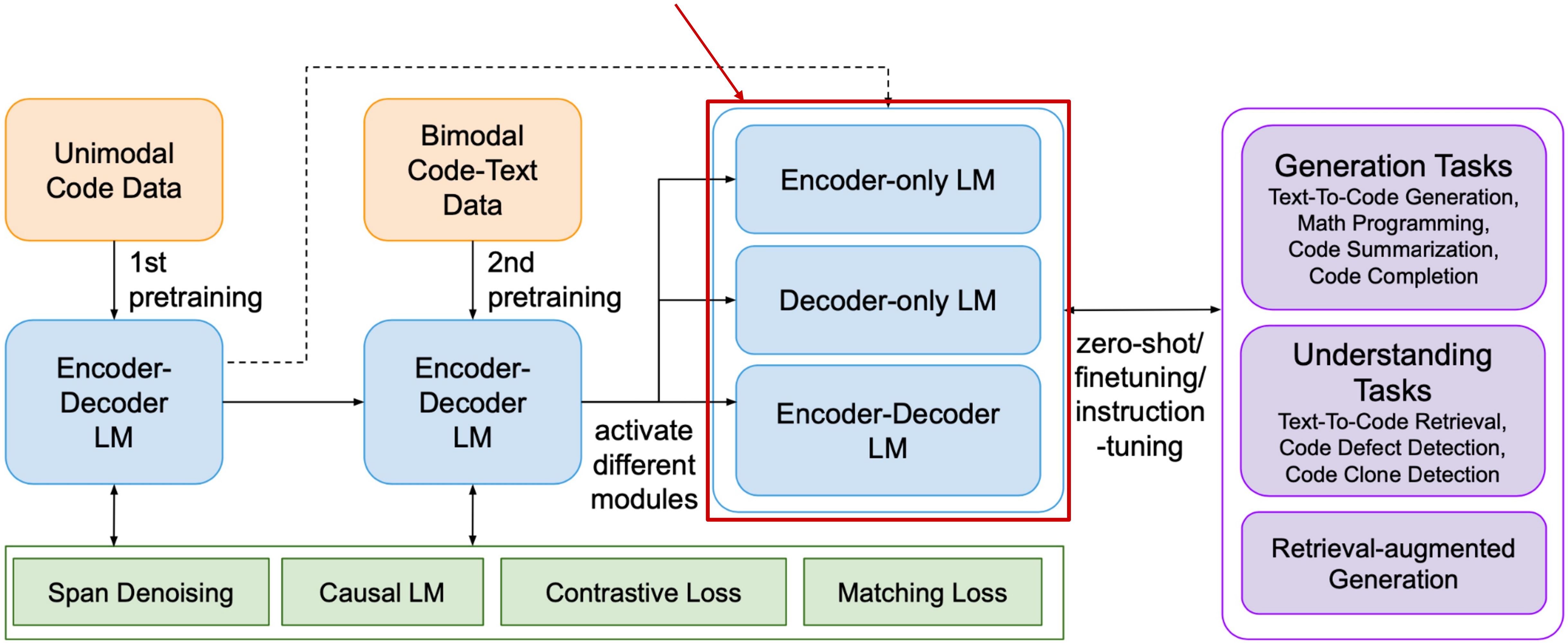


CodeT5+, https://arxiv.org/abs/2305.07922

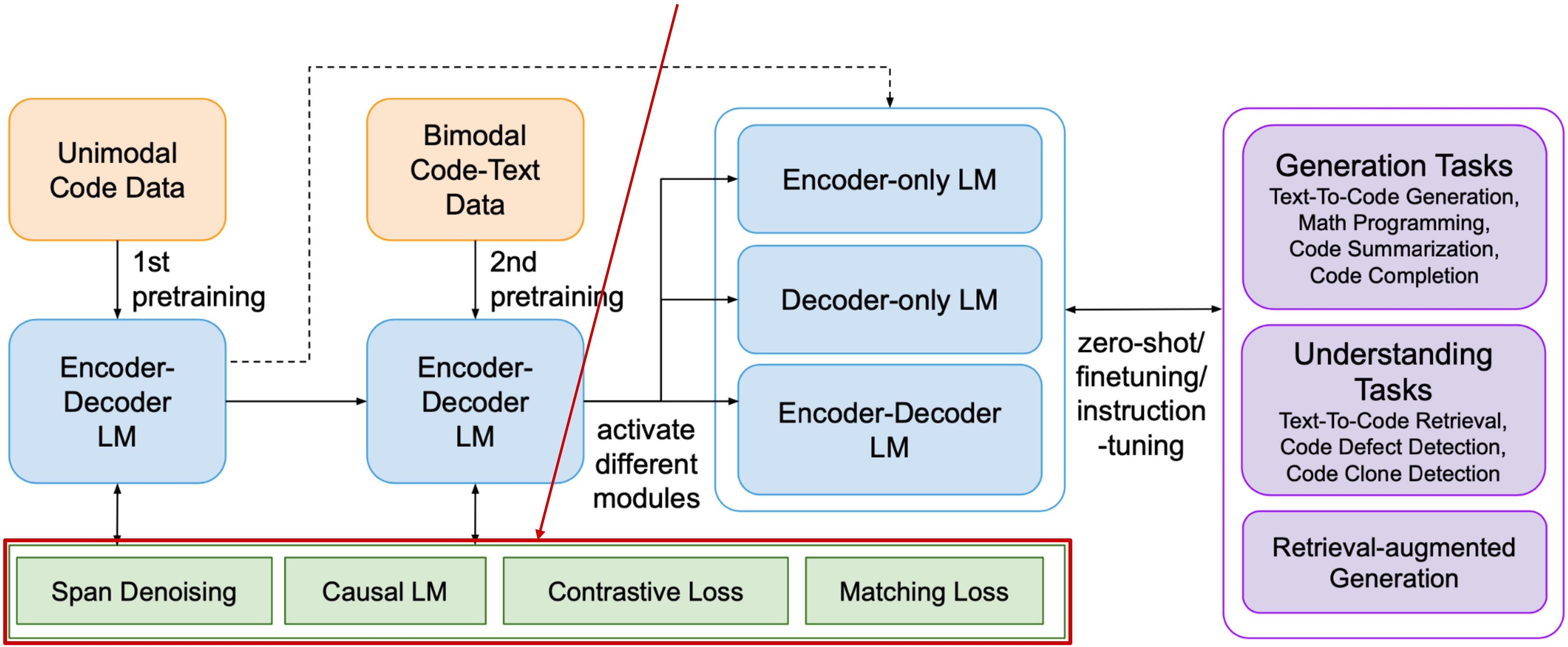
CodeT5+: Supports downstream tasks



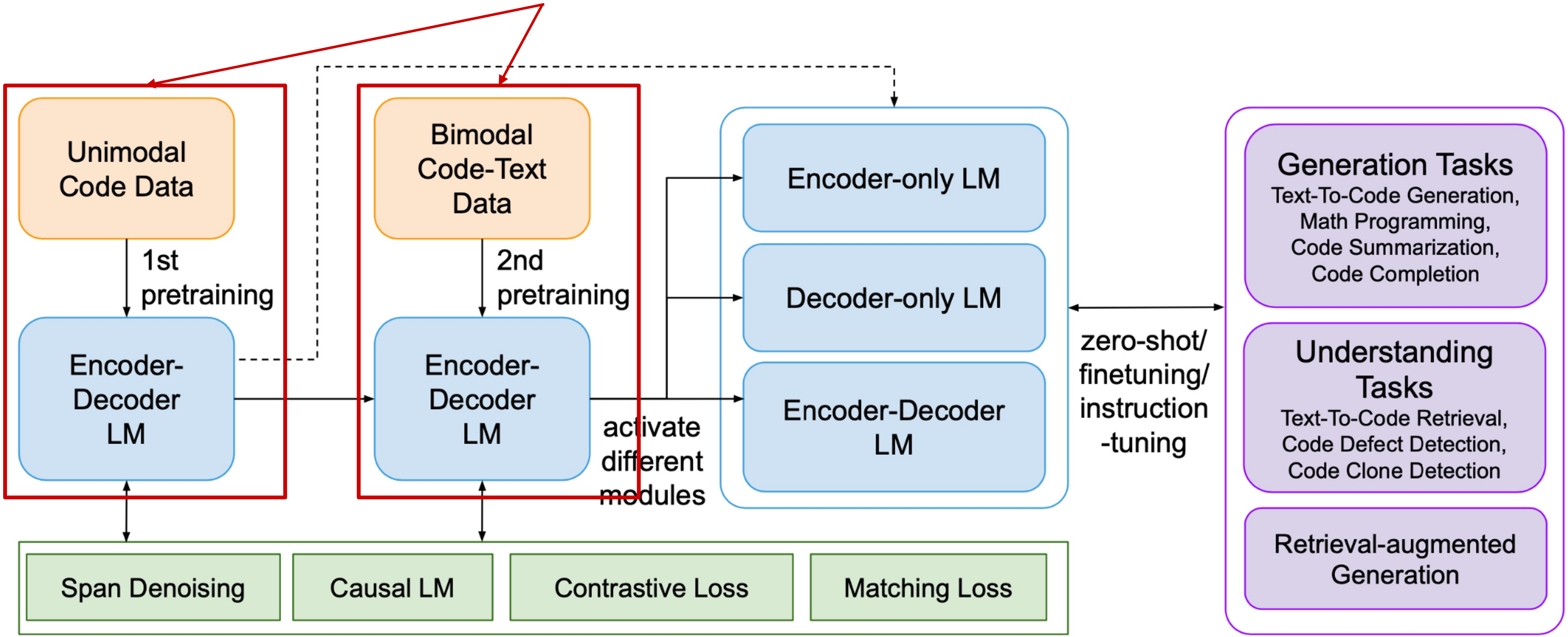
CodeT5+: Can operate in different modes



CodeT5+: Uses several pre-training tasks



CodeT5+: Has two pre-training stages



Stage 1: Code-only pre-training

Goal: Train model to recover code contexts at different scales

Data: Code from GitHub

Tasks:

- Span Denoising (15% masked tokens)
- Causal LM
  - Partial programs
  - Complete programs

Stage 2: Code and text pre-training

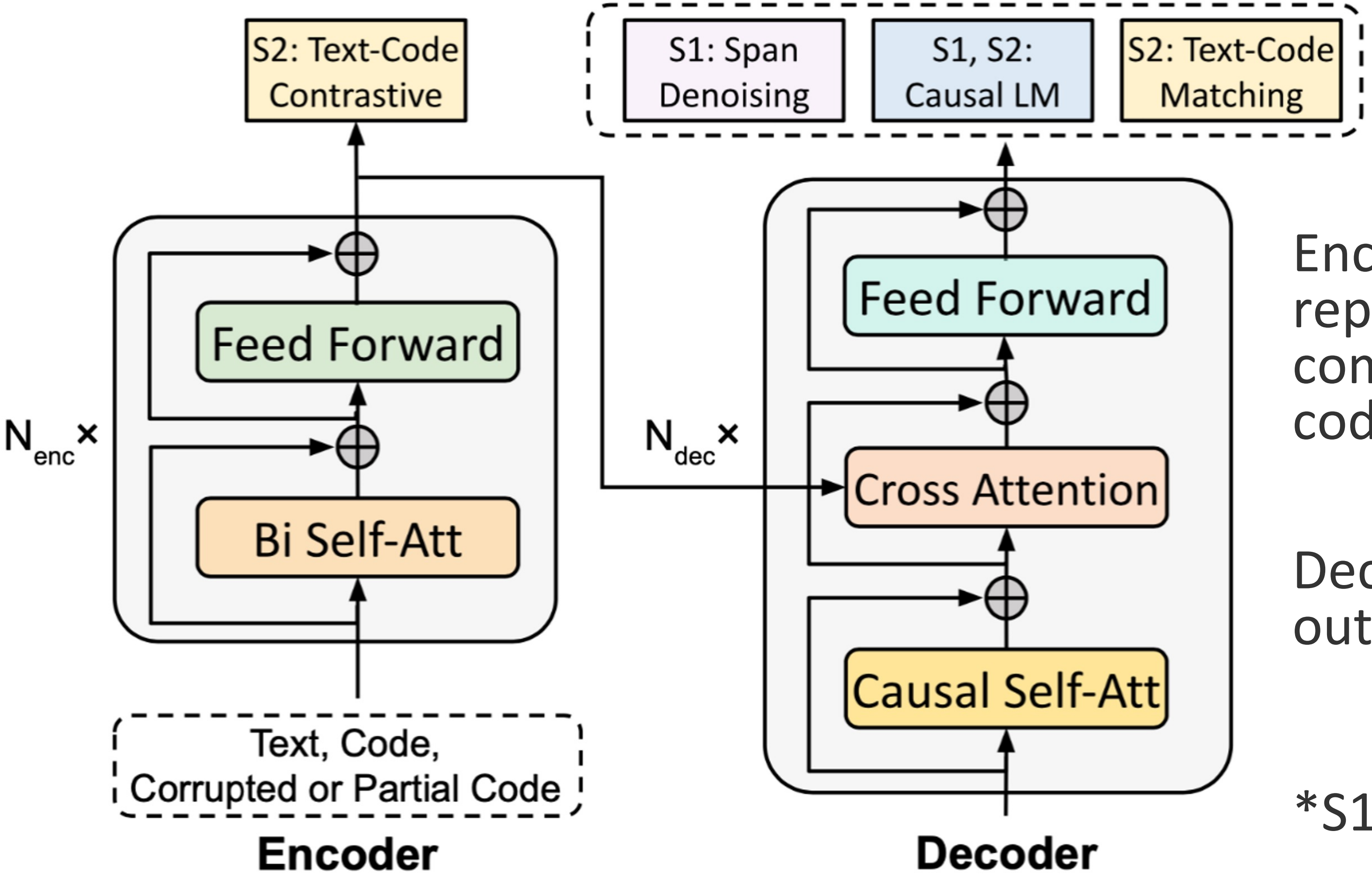
Goal: Train model for cross-modal understanding and generation

Data: CodeSearchNet (Docstring & Code)

Tasks:

- Contrastive Learning (align feature space of code and text representation)
- Text-Code Matching (predict if semantics match)
- Text-Code Causal LM (text-to-code and code-to-text generation)

Code T5+: Architecture



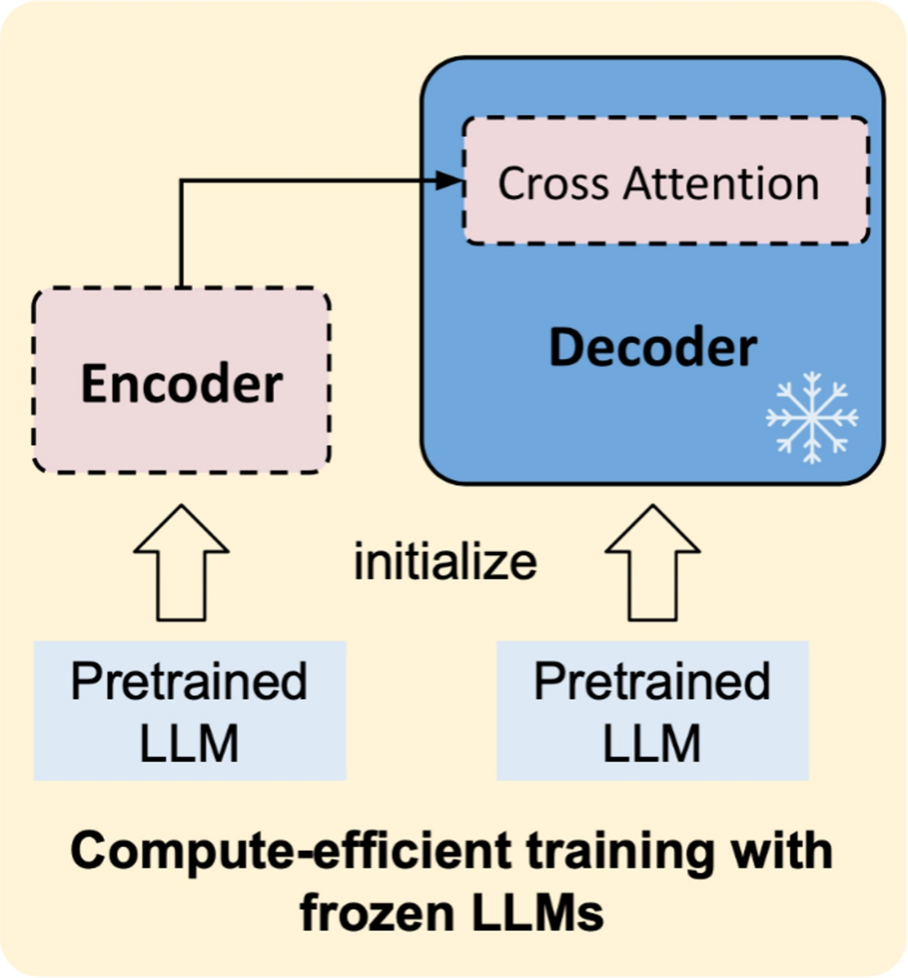
Encoder: encodes contextual representations from either complete, partial or span-masked code/text sequences

Decoder: generates different outputs based on pretraining task

*S1: Stage1 and S2: Stage2

Code T5+: Compute-Efficient Training

- Shallow encoder and deep decoder, initialized with pretrained weights of a decoder code model (CodeGen, Nijkamp et al. 2023)
- Only encoder and cross attention layers are trainable
- Decoder weights are frozen



CodeT5+: Results

HumanEval code generation: slightly outperforms the CodeGen models it is initialized with

Model	Model size	pass@1	pass@10	pass@100
Closed-source models				
Codex	2.5B	21.4	35.4	59.5
Codex	12B	28.8	46.8	72.3
code-cushman-001	-	33.5	54.3	77.4
code-davinci-002	-	47.0	**74.9**	**92.1**
GPT-3.5	-	48.1	-	-
Open-source models				
CodeGen-mono	2B	23.7	36.6	57.0
CodeGen-mono	6B	26.1	42.3	65.8
CodeGen-mono	16B	29.3	49.9	75.0
CodeT5+	220M	12.0	20.7	31.6
CodeT5+	770M	15.5	27.2	42.7
CodeT5+	2B	24.2	38.2	57.8
CodeT5+	6B	28.0	47.2	69.8
CodeT5+	16B	30.9	51.6	76.7

CodeT5+: Results

Code retrieval: outperforms CodeT5 and CodeBERT

Table 6: **Text-to-Code Retrieval results (MRR) on CodeXGLUE:** CodeT5+ achieves consistent performance gains over the original CodeT5 models across all 3 retrieval benchmarks in 7 programming languages. Overall, our models demonstrate remarkable performance, outperforming many strong encoder-based models pretrained with contrastive loss such as SYNCOBERT and UniXcoder.

Model	CodeSearchNet							CosQA	AdvTest
	Ruby	JS	Go	Python	Java	PHP	Overall		
CodeBERT 125M	67.9	62.0	88.2	67.2	67.6	62.8	69.3	65.7	27.2
GraphCodeBERT 125M	70.3	64.4	89.7	69.2	69.1	64.9	71.3	68.4	35.2
SYNCOBERT 125M	72.2	67.7	91.3	72.4	72.3	67.8	74.0	-	38.3
UniXcoder 125M	74.0	68.4	91.5	72.0	72.6	67.6	74.4	70.1	41.3
CodeGen-multi 350M	66.0	62.2	90.0	68.6	70.1	63.9	70.1	64.8	34.8
PLBART 140M	67.5	61.6	88.7	66.3	66.3	61.1	68.6	65.0	34.7
CodeT5 220M	71.9	65.5	88.8	69.8	68.6	64.5	71.5	67.8	39.3
CodeT5+ 220M	77.7	70.8	92.4	75.6	76.1	69.8	77.1	72.7	43.3
CodeT5+ 770M	**78.0**	**71.3**	**92.7**	**75.8**	**76.2**	**70.1**	**77.4**	**74.0**	**44.7**