

# Automatic Code Generation

Arzhantsev Andrey, 202  
Maxim Kiselev, 202

# Содержание

- Виды задач
- Данные для обучения/тестирования
- Трансформеры для кода
  - CodeBert (GraphCodeBert)
  - CodeT5
- Более продвинутые/новые модели для кода
  - DocPrompting
  - CasualMasking
  - InCoder
  - PaL

- Андрей
- Максим

# Виды задач

## Text-Code

- Code from NL description

NL: Gets the value of the tags property. This accessor method returns a reference to the live list, not a snapshot.

**Variables:**  
String validationPattern;  
List<String> tags;  
**Methods:**  
String getValidationPattern  
void setValidationPattern

**Code:**  
List <String> function() {  
 if (tags == null) {  
 tags = new ArrayList <String>(); }  
 return this.tags; }

(a)

NL: Convert mixed case to underscores.

**Variables:**  
NamingStrategy INSTANCE;  
**Methods:**  
String classToTableName  
String collectionTableName  
String tableName  
String columnName  
String addUnderscores

**Code:**  
String function (String arg0) {  
 return addUnderscores (arg0);

(b)

- Code Search

### Query 1: How can I convert a stack trace to a string?

```
public synchronized static String getStackTrace(Exception e) {  
    e.fillInStackTrace();  
    StringBuffer buffer = new StringBuffer();  
    buffer.append(e.getMessage() + "-");  
    for (StackTraceElement el: e.getStackTrace()) {  
        buffer.append(el.toString() + "-");  
    }  
    return buffer.toString();  
}
```

<https://github.com/Dynatrace/Dynatrace-AppMon-REST-Monitor-Plugin/blob/master/src/com/readolmen/dynatrace/restmonitor/RestMonitor.java>

### Query 2: How do I get a platform-dependent new line character?

```
public static String getPlatformLineSeparator() {  
    return System.getProperty("line.separator");  
}
```

<https://github.com/nutritionfactsorg/daily-dozen-android/blob/master/app/src/main/java/org/nutritionfacts/dailydozen/Common.java>

# Виды задач

## Code-Code

- Code completion / Cloze test

Cloze Test-all

Doc.: Open the drop box.

Code:

```
def open(self):
    self.workingArea.<mask>()
    self.rnid_pkgidx_map = {}
    self.rnid_to_return = deque()
```

Answer: open

Cloze Test-maxmin

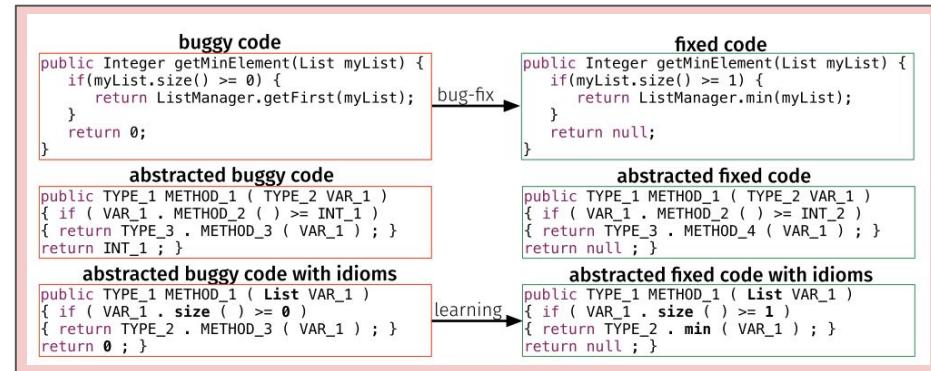
Doc.: Find min and max values of every feature.

Code:

```
def fit(self, X, y=None):
    X = check_array(X)
    self._x_min = X.<mask>(axis=0)
    self._x_max = X.max(axis=0)
    return self
```

Answer: min

- Defect detection / Code repair



- Clone Detection

- Code Translation

# Виды задач

## Code-Text

- Code summarization

### 3. Source Code (SQL):

```
SELECT Max(marks) FROM stud_records  
WHERE marks <  
(SELECT Max(marks) FROM stud_records);
```

#### Descriptions:

- a. Get the second largest value of a column
- b. Retrieve the next max record in a table

## Text-Text

- Documentation Translation

### Input (English):

Multinomial Logistic Regression (Softmax regression) is used to compute the probabilities of several possible outcomes in classification problems.

### Output (Chinese):

多项式逻辑回归（Softmax回归）用于计算分类问题中几种可能结果的概率。

# Данные для обучения/тестирования

## CodeXGLUE

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

(see <https://arxiv.org/pdf/2102.04664.pdf>)

# Данные для обучения/тестирования

## Alternatives

- Other benchmarks
  - HumanEval
  - MBPP
  - ...
- Public Github / Other Open Source
- StackOverflow questions, answers, and comments

```
def is_prime(n):  
    """Return true if a given number is prime, and  
    false otherwise.  
    >>> is_prime(6)  
    False  
    >>> is_prime(101)  
    True  
    >>> is_prime(11)  
    True  
    >>> is_prime(13441)  
    True  
    >>> is_prime(61)  
    True  
    >>> is_prime(4)  
    False  
    >>> is_prime(1)  
    False  
    """
```

```
### COMPLETION 1 (WRONG): ###  
for i in range(2, n):  
    if n % i == 0:  
        return False  
return True
```

```
### COMPLETION 4 (CORRECT): ###  
if n < 2: return False  
if n == 2: return True  
if n%2 == 0: return False  
return not any(n%k == 0 for k in range(3,int(n  
    *0.5)+1,2))
```

```
<| file ext=.py source=github |>  
from typing import Dict  
  
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  
</| file filename=string_utils.py dstars=4>
```

### Removing duplicates in lists

Pretty much I need to write a program to check if a list has any duplicates and if it does it removes them and returns a new list with the items that weren't duplicated/removed. This is what I have but to be honest I do not know what to do.

```
406  
def remove_duplicates():  
    t = ['a', 'b', 'c', 'd']  
    t2 = ['a', 'c', 'd']  
    for t in t2:  
        t.append(t.remove())  
    return t
```

The common approach to get a unique collection of items is to use a `set`. Sets are *unordered* collections of *distinct* objects. To create a set from any iterable, you can simply pass it to the built-in `set()` function. If you later need a real list again, you can similarly pass the set to the `list()` function.

The following example should cover whatever you are trying to do:

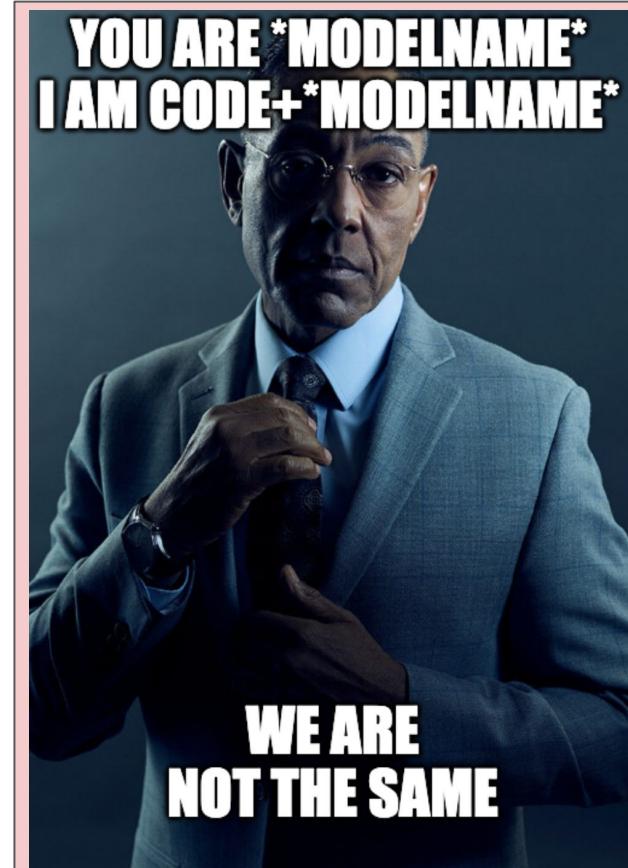
```
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
10010  
10011  
10012  
10013  
10014  
10015  
10016  
10017  
10018  
10019  
10020  
10021  
10022  
10023  
10024  
10025  
10026  
10027  
10028  
10029  
10030  
10031  
10032  
10033  
10034  
10035  
10036  
10037  
10038  
10039  
10040  
10041  
10042  
10043  
10044  
10045  
10046  
10047  
10048  
10049  
10050  
10051  
10052  
10053  
10054  
10055  
10056  
10057  
10058  
10059  
10060  
10061  
10062  
10063  
10064  
10065  
10066  
10067  
10068  
10069  
10070  
10071  
10072  
10073  
10074  
10075  
10076  
10077  
10078  
10079  
10080  
10081  
10082  
10083  
10084  
10085  
10086  
10087  
10088  
10089  
10090  
10091  
10092  
10093  
10094  
10095  
10096  
10097  
10098  
10099  
100100  
100101  
100102  
100103  
100104  
100105  
100106  
100107  
100108  
100109  
100110  
100111  
100112  
100113  
100114  
100115  
100116  
100117  
100118  
100119  
100120  
100121  
100122  
100123  
100124  
100125  
100126  
100127  
100128  
100129  
100130  
100131  
100132  
100133  
100134  
100135  
100136  
100137  
100138  
100139  
100140  
100141  
100142  
100143  
100144  
100145  
100146  
100147  
100148  
100149  
100150  
100151  
100152  
100153  
100154  
100155  
100156  
100157  
100158  
100159  
100160  
100161  
100162  
100163  
100164  
100165  
100166  
100167  
100168  
100169  
100170  
100171  
100172  
100173  
100174  
100175  
100176  
100177  
100178  
100179  
100180  
100181  
100182  
100183  
100184  
100185  
100186  
100187  
100188  
100189  
100190  
100191  
100192  
100193  
100194  
100195  
100196  
100197  
100198  
100199  
100200  
100201  
100202  
100203  
100204  
100205  
100206  
100207  
100208  
100209  
100210  
100211  
100212  
100213  
100214  
100215  
100216  
100217  
100218  
100219  
100220  
100221  
100222  
100223  
100224  
100225  
100226  
100227  
100228  
100229  
100230  
100231  
100232  
100233  
100234  
100235  
100236  
100237  
100238  
100239  
100240  
100241  
100242  
100243  
100244  
100245  
100246  
100247  
100248  
100249  
100250  
100251  
100252  
100253  
100254  
100255  
100256  
100257  
100258  
100259  
100260  
100261  
100262  
100263  
100264  
100265  
100266  
100267  
100268  
100269  
100270  
100271  
100272  
100273  
100274  
100275  
100276  
100277  
100278  
100279  
100280  
100281  
100282  
100283  
100284  
100285  
100286  
100287  
100288  
100289  
100290  
100291  
100292  
100293  
100294  
100295  
100296  
100297  
100298  
100299  
100300  
100301  
100302  
100303  
100304  
100305  
100306  
100307  
100308  
100309  
100310  
100311  
100312  
100313  
100314  
100315  
100316  
100317  
100318  
100319  
100320  
100321  
100322  
100323  
100324  
100325  
100326  
100327  
100328  
100329  
100330  
100331  
100332  
100333  
100334  
100335  
100336  
100337  
100338  
100339  
100340  
100341  
100342  
100343  
100344  
100345  
100346  
100347  
100348  
100349  
100350  
100351  
100352  
100353  
100354  
100355  
100356  
100357  
100358  
100359  
100360  
100361  
100362  
100363  
100364  
100365  
100366  
100367  
100368  
100369  
100370  
100371  
100372  
100373  
100374  
100375  
100376  
100377  
100378  
100379  
100380  
100381  
100382  
100383  
100384  
100385  
100386  
100387  
100388  
100389  
100390  
100391  
100392  
100393  
100394  
100395  
100396  
100397  
100398  
100399  
100400  
100401  
100402  
100403  
100404  
100405  
100406  
100407  
100408  
100409  
100410  
100411  
100412  
100413  
100414  
100415  
100416  
100417  
100418  
100419  
100420  
100421  
100422  
100423  
100424  
100425  
100426  
100427  
100428  
100429  
100430  
100431  
100432  
100433  
100434  
100435  
100436  
100437  
100438  
100439  
100440  
100441  
100442  
100443  
100444  
100445  
100446  
100447  
100448  
100449  
100450  
100451  
100452  
100453  
100454  
100455  
100456  
100457  
100458  
100459  
100460  
100461  
100462  
100463  
100464  
100465  
100466  
100467  
100468  
100469  
100470  
100471  
100472  
100473  
100474  
100475  
100476  
100477  
100478  
100479  
100480  
100481  
100482  
100483  
100484  
100485  
100486  
100487  
100488  
100489  
100490  
100491  
100492  
100493  
100494  
100495  
100496  
100497  
100498  
100499  
100500  
100501  
100502  
100503  
100504  
100505  
100506  
100507  
100508  
100509  
100510  
100511  
100512  
100513  
100514  
100515  
100516  
100517  
100518  
100519  
100520  
100521  
100522  
100523  
100524  
100525  
100526  
100527  
100528  
100529  
100530  
100531  
100532  
100533  
100534  
100535  
100536  
100537  
100538  
100539  
100540  
100541  
100542  
100543  
100544  
100545  
100546  
100547  
100548  
100549  
100550  
100551  
100552  
100553  
100554  
100555  
100556  
100557  
100558  
100559  
100560  
100561  
100562  
100563  
100564  
100565  
100566  
100567  
100568  
100569  
100570  
100571  
100572  
100573  
100574  
100575  
100576  
100577  
100578  
100579  
100580  
100581  
100582  
100583  
100584  
100585  
100586  
100587  
100588  
100589  
100590  
100591  
100592  
100593  
100594  
100595  
100596  
100597  
100598  
100599  
100600  
100601  
100602  
100603  
100604  
100605  
100606  
100607  
100608  
100609  
100610  
100611  
100612  
100613  
100614  
100615  
100616  
100617  
100618  
100619  
100620  
100621  
100622  
100623  
100624  
100625  
100626  
100627  
100628  
100629  
100630  
100631  
100632  
100633  
100634  
100635  
100636  
100637  
100638  
100639  
100640  
100641  
100642  
100643  
100644  
100645  
100646  
100647  
100648  
100649  
100650  
100651  
100652  
100653  
100654  
100655  
100656  
100657  
100658  
100659  
100660  
100661  
100662  
100663  
100664  
100665  
100666  
100667  
100668  
100669  
100670  
100671  
100672  
100673  
100674  
100675  
100676  
100677  
100678  
100679  
100680  
100681  
100682  
100683  
100684  
100685  
100686  
100687  
100688  
100689  
100690  
100691  
100692  
100693  
100694  
100695  
100696  
100697  
100698  
100699  
100700  
100701  
100702  
100703  
100704  
100705  
100706  
100707  
100708  
100709  
100710  
100711  
100712  
100713  
100714  
100715  
100716  
100717  
100718  
100719  
100720  
100721  
100722  
100723  
100724  
100725  
100726  
100727  
100728  
100729  
100730  
100731  
100732  
100733  
100734  
100735  
100736  
100737  
100738  
100739  
100740  
100741  
100742  
100743  
100744  
100745  
100746  
100747  
100748  
100749  
100750  
100751  
100752  
100753  
100754  
100755  
100756  
100757  
100758  
100759  
100760  
100761  
100762  
100763  
100764  
100765  
100766  
100767  
100768  
100769  
100770  
100771  
100772  
100773  
100774  
100775  
100776  
100777  
100778  
100779  
100780  
100781  
100782  
100783  
100784  
100785  
100786  
100787  
100788  
100789  
100790  
100791  
100792  
100793  
100794  
100795  
100796  
100797  
100798  
100799  
100800  
100801  
100802  
100803  
100804  
100805  
100806  
100807  
100808  
100809  
100810  
100811  
100812  
100813  
100814  
100815  
100816  
100817  
100818  
100819  
100820  
100821  
100822  
100823  
100824  
100825  
100826  
100827  
100828  
100829  
100830  
100831  
100832  
100833  
100834  
100835  
100836  
100837  
100838  
100839  
100840  
100841  
100842  
100843  
100844  
100845  
100846  
100847  
100848  
100849  
100850  
100851  
100852  
100853  
100854  
100855  
100856  
100857  
100858  
100859  
100860  
100861  
100862  
100863  
100864  
100865  
100866  
100867  
100868  
100869  
100870  
100871  
100872  
100873  
100874  
100875  
100876  
100877  
100878  
100879  
100880  
100881  
100882  
100883  
100884  
100885  
100886  
100887  
100888  
100889  
100890  
100891  
100892  
100893  
100894  
100895  
100896  
100897  
100898  
100899  
100900  
100901  
100902  
100903  
100904  
100905  
100906  
100907  
100908  
100909  
100910  
100911  
100912  
100913  
100914  
100915  
100916  
100917  
100918  
100919  
100920  
100921  
100922  
100923  
100924  
100925  
100926  
100927  
100928  
100929  
100930  
100931  
100932  
100933  
100934  
100935  
100936  
100937  
100938  
100939  
100940  
100941  
100942  
100943  
100944  
100945  
100946  
100947  
100948  
100949  
100950  
100951  
100952  
100953  
100954  
100955  
100956  
100957  
100958  
100959  
100960  
100961  
100962  
100963  
100964  
100965  
100966  
100967  
100968  
100969  
100970  
100971  
100972  
100973  
100974  
100975  
100976  
100977  
100978  
100979  
100980  
100981  
100982  
100983  
100984  
100985  
100986  
100987  
100988  
100989  
100990  
100991  
100992  
100993  
100994  
100995  
100996  
100997  
100998  
100999  
1001000  
1001001  
1001002  
1001003  
1001004  
1001005  
1001006  
1001007  
1001008  
1001009  
10010010  
10010011  
10010012  
10010013  
10010014  
10010015  
10010016  
10010017  
10010018  
10010019  
100100100  
100100101  
100100102  
100100103  
100100104  
100100105  
100100106  
100100107  
100100108  
100100109  
100100110  
100100111  
100100112  
100100113  
100100114  
100100115  
100100116  
100100117  
100100118  
100100119  
100100120  
100100121  
100100122  
100100123  
100100124  
100100125  
100100126  
100100127  
100100128  
100100129  
100100130  
100100131  
100100132  
100100133  
100100134  
100100135  
100100136  
100100137  
100100138  
100100139  
100100140  
100100141  
100100142  
100100143  
100100144  
100100145  
100100146  
100100147  
100100148  
100100149  
100100150  
100100151  
100100152  
100100153  
100100154  
100100155  
100100156  
100100157  
100100158  
100100159  
100100160  
100100161  
100100162  
100100163  
100100164  
100100165  
100100166  
100100167  
100100168  
100100169  
100100170  
100100171  
100100172  
100100173  
100100174  
100100175  
100100176  
100100177  
100100178  
100100179  
100100180  
100100181  
100100182  
100100183  
100100184  
100100185  
100100186  
100100187  
100100188  
100100189  
100100190  
100100191  
100100192  
100100193  
100100194  
100100195  
100100196  
100100197  
100100198  
100100199  
100100100  
100100101  
100100102  
100100103  
100100104  
100100105  
100100106  
100100107  
100100108  
100100109  
100100110  
100100111  
100100112  
100100113  
100100114  
100100115  
100100116  
100100117  
100100118  
100100119  
100100120  
100100121  
100100122  
100100123  
100100124  
100100125  
100100126  
100100127  
100100128  
100100129  
100100130  
100100131  
100100132  
100100133  
100100134  
100100135  
100100136  
100100137  
100100138  
100100139  
100100140  
100100141  
100100142  
100100143  
100100144  
100100145  
100100146  
100100147  
100100148  
100100149  
100100150  
100100151  
100100152  
100100153  
100100154  
100100155  
100100156  
100100157  
100100158  
100100159  
100100160  
100100161  
100100162  
100100163  
100100164  
100100165  
100100166  
100100167  
100100168  
100100169  
100100170  
100100171  
100100172  
100100173  
100100174  
100100175  
100100176  
100100177  
100100178  
100100179  
100100180  
100100181  
100100182  
100100183  
10010018
```

# Трансформеры для кода

## Basic transformers

- BERT -> CodeBERT
- GPT -> CodeGPT
- T5 -> CodeT5

Same model architecture  
But other data, other objectives



# Трансформеры для кода

## CodeBert PreTraining Naive

- Masked Language Modeling  
for bimodal data ( $w+c$ )

$$m_i^w \sim \text{unif}\{1, |w|\} \text{ for } i = 1 \text{ to } |w|$$

$$m_i^c \sim \text{unif}\{1, |c|\} \text{ for } i = 1 \text{ to } |c|$$

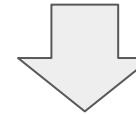
$$w^{\text{masked}} = \text{REPLACE}(w, m^w, [\text{MASK}])$$

$$c^{\text{masked}} = \text{REPLACE}(c, m^c, [\text{MASK}])$$

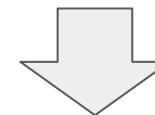
$$x = w + c$$

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

WORDS + [SEP] + CODE + [EOS]



Mask 15% Tokens



Learn to predict masks

# Трансформеры для кода

## CodeBert PreTraining

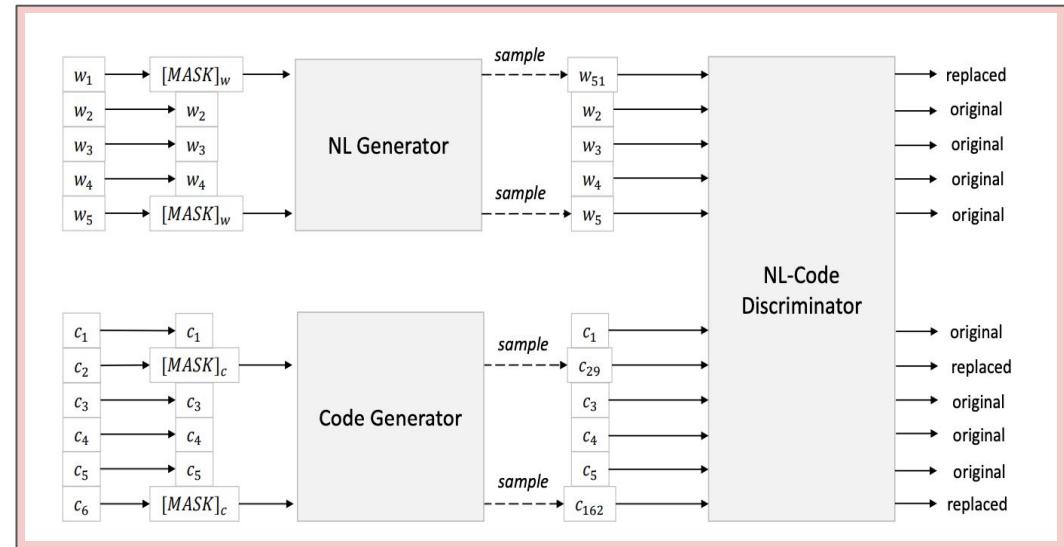
- Add Replaced Token Detection for unimodal and bimodal data

$$\hat{w}_i \sim p^{G_w}(w_i | \mathbf{w}^{\text{masked}}) \text{ for } i \in \mathbf{m}^w$$
$$\hat{c}_i \sim p^{G_c}(c_i | \mathbf{c}^{\text{masked}}) \text{ for } i \in \mathbf{m}^c$$

$$\mathbf{w}^{\text{corrupt}} = \text{REPLACE}(\mathbf{w}, \mathbf{m}^w, \hat{\mathbf{w}})$$
$$\mathbf{c}^{\text{corrupt}} = \text{REPLACE}(\mathbf{c}, \mathbf{m}^c, \hat{\mathbf{c}})$$
$$\mathbf{x}^{\text{corrupt}} = \mathbf{w}^{\text{corrupt}} + \mathbf{c}^{\text{corrupt}}$$

$$\mathcal{L}_{\text{RTD}}(\theta) = \sum_{i=1}^{|w|+|c|} \left( \delta(i) \log p^{D_2}(\mathbf{x}^{\text{corrupt}}, i) + (1 - \delta(i)) (1 - \log p^{D_2}(\mathbf{x}^{\text{corrupt}}, i)) \right)$$

$$\delta(i) = \begin{cases} 1, & \text{if } x_i^{\text{corrupt}} = x_i. \\ 0, & \text{otherwise.} \end{cases}$$



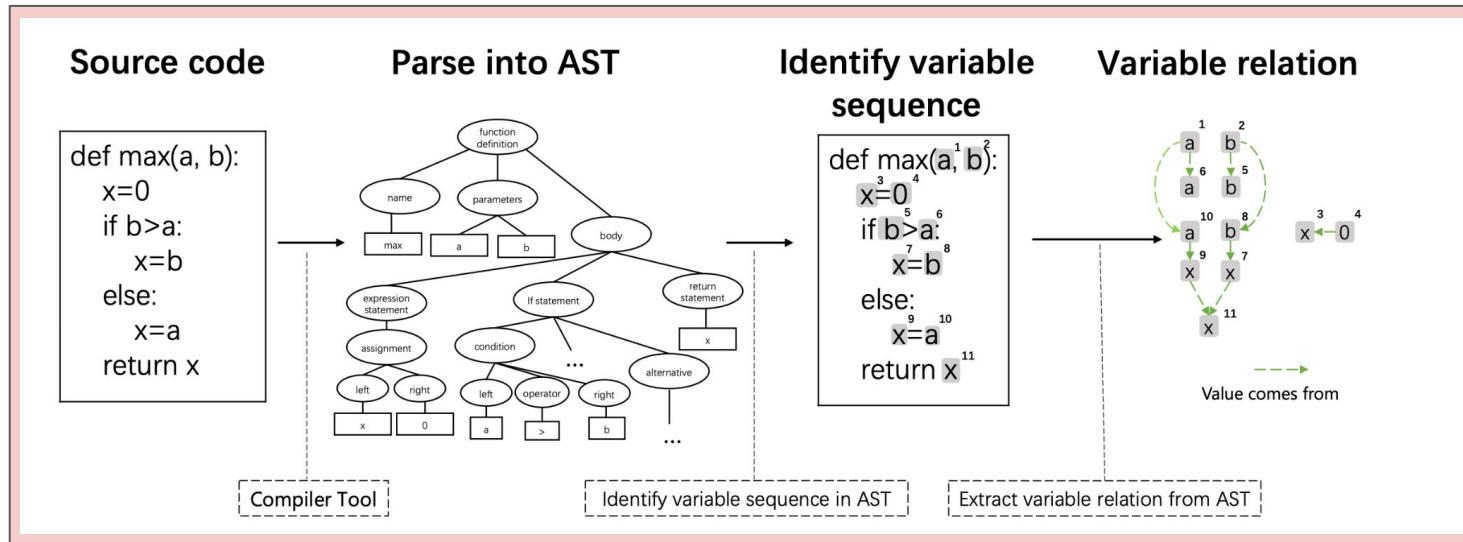
- Final loss function

$$\min_{\theta} \mathcal{L}_{\text{MLM}}(\theta) + \mathcal{L}_{\text{RTD}}(\theta)$$

# Трансформеры для кода

## GraphCodeBert

- Transform input to Words+Code+VariableSequence



# Трансформеры для кода

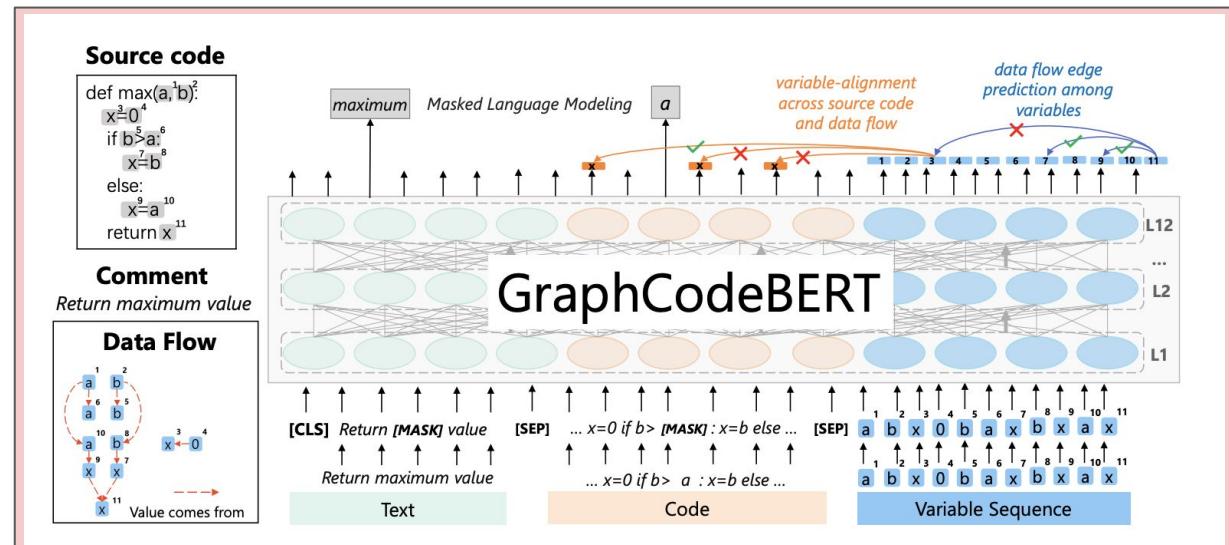
## GraphCodeBert PreTraining

- Add EdgePrediction between nodes in Dataflow

$$\begin{aligned} loss_{EdgePred} = & - \sum_{e_{ij} \in E_c} [\delta(e_{ij} \in E_{mask}) \log p_{e_{ij}} + \\ & + (1 - \delta(e_{ij} \in E_{mask})) \log(1 - p_{e_{ij}})] \end{aligned}$$

- Add NodeAlignment between nodes in Dataflow and Code

$$\begin{aligned} loss_{NodeAlign} = & - \sum_{e_{ij} \in E'_c} [\delta(e_{ij} \in E'_{mask}) \log p_{e_{ij}} + \\ & + (1 - \delta(e_{ij} \in E'_{mask})) \log(1 - p_{e_{ij}})] \end{aligned}$$



# Трансформеры для кода

## CodeT5 Pretraining

- Use Masked Span Prediction instead of casual masks

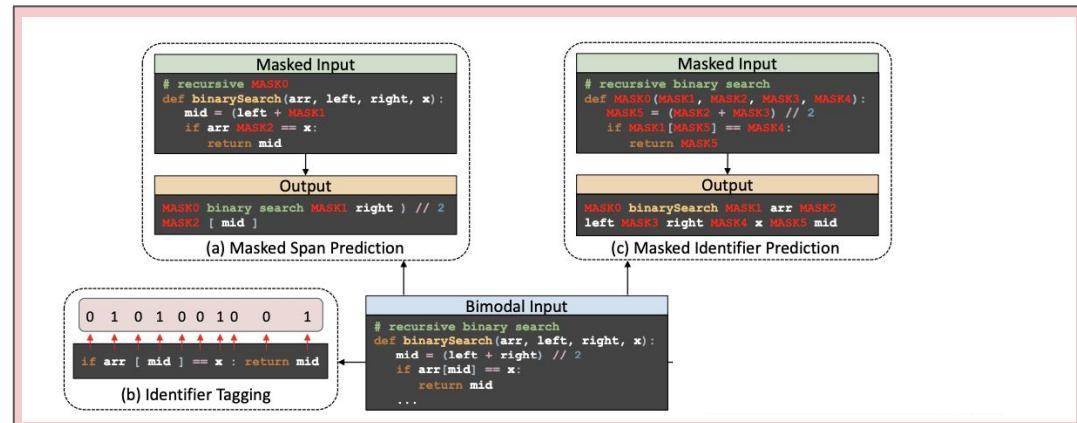
$$\mathcal{L}_{MSP}(\theta) = \sum_{t=1}^k -\log P_\theta(x_t^{\text{mask}} | \mathbf{x}^{\setminus \text{mask}}, \mathbf{x}_{<t}^{\text{mask}})$$

- Add IdentifierTagging to Code tokens

$$\mathcal{L}_{IT}(\theta_e) = \sum_{i=1}^m -[y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

- Add MaskedIdentifierPrediction for Code tokens

$$\mathcal{L}_{MIP}(\theta) = \sum_{j=1}^{|I|} -\log P_\theta(I_j | \mathbf{x}^{\setminus I}, \mathbf{I}_{<j})$$



# Трансформеры для кода

## Fine-Tuning



Yes.

# Трансформеры для кода

## Experiments

- Code Search

model	Ruby	Javascript	Go	Python	Java	Php	Overall
NBow	0.162	0.157	0.330	0.161	0.171	0.152	0.189
CNN	0.276	0.224	0.680	0.242	0.263	0.260	0.324
BiRNN	0.213	0.193	0.688	0.290	0.304	0.338	0.338
selfAtt	0.275	0.287	0.723	0.398	0.404	0.426	0.419
RoBERTa	0.587	0.517	0.850	0.587	0.599	0.560	0.617
RoBERTa (code)	0.628	0.562	0.859	0.610	0.620	0.579	0.643
CodeBERT	0.679	0.620	0.882	0.672	0.676	0.628	0.693
GraphCodeBERT	<b>0.703</b>	<b>0.644</b>	<b>0.897</b>	<b>0.692</b>	<b>0.691</b>	<b>0.649</b>	<b>0.713</b>

- Clone Detection

Model	Precision	Recall	F1
Deckard	0.93	0.02	0.03
RtvNN	0.95	0.01	0.01
CDLH	0.92	0.74	0.82
ASTNN	0.92	0.94	0.93
FA-AST-GMN	<b>0.96</b>	0.94	0.95
RoBERTa (code)	0.949	0.922	0.935
CodeBERT	0.947	0.934	0.941
GraphCodeBERT	0.948	<b>0.952</b>	<b>0.950</b>

- Code Translation

Methods	Java to C#		C# to Java		Refine Small		Refine Medium	
	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM
Naive Copy	18.54	0	18.69	0	78.06	0	90.91	0
RoBERTa (code)	77.46	56.10	71.99	57.90	77.30	15.90	90.07	4.10
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	<b>80.02</b>	17.30	<b>91.31</b>	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	<b>84.03</b>	<b>65.90</b>	<b>79.87</b>	<b>66.90</b>	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	<b>22.59</b>	88.90	<b>14.18</b>

- Code Summarization

Methods	Ruby	JavaScript	Go	Python	Java	PHP	Overall
RoBERTa	11.17	11.90	17.72	18.14	16.47	24.02	16.57
CodeBERT	12.16	14.90	18.07	19.06	17.65	25.16	17.83
DOBF	-	-	-	18.24	19.05	-	-
PLBART	14.11	15.56	18.91	19.30	18.45	23.58	18.32
CodeT5-small	14.87	15.32	19.25	20.04	19.92	25.46	19.14
+dual-gen	15.30	15.61	19.74	19.94	19.78	26.48	19.48
+multi-task	15.50	15.52	19.62	20.10	19.59	25.69	19.37
CodeT5-base	15.24	16.16	19.56	20.01	20.31	26.03	19.55
+dual-gen	<b>15.73</b>	16.00	19.71	20.11	20.41	<b>26.53</b>	19.75
+multi-task	15.69	<b>16.24</b>	<b>19.76</b>	<b>20.36</b>	<b>20.46</b>	26.09	<b>19.77</b>

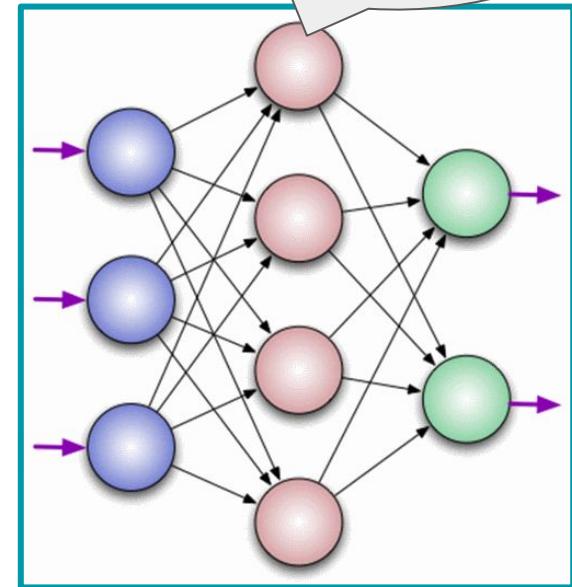
## Sub-contents

- Problem №1
- Problem №2
- Problem №3

# Problem №1



Write code to  
launch Krylov  
iterations to  
solve  $Ax=b$



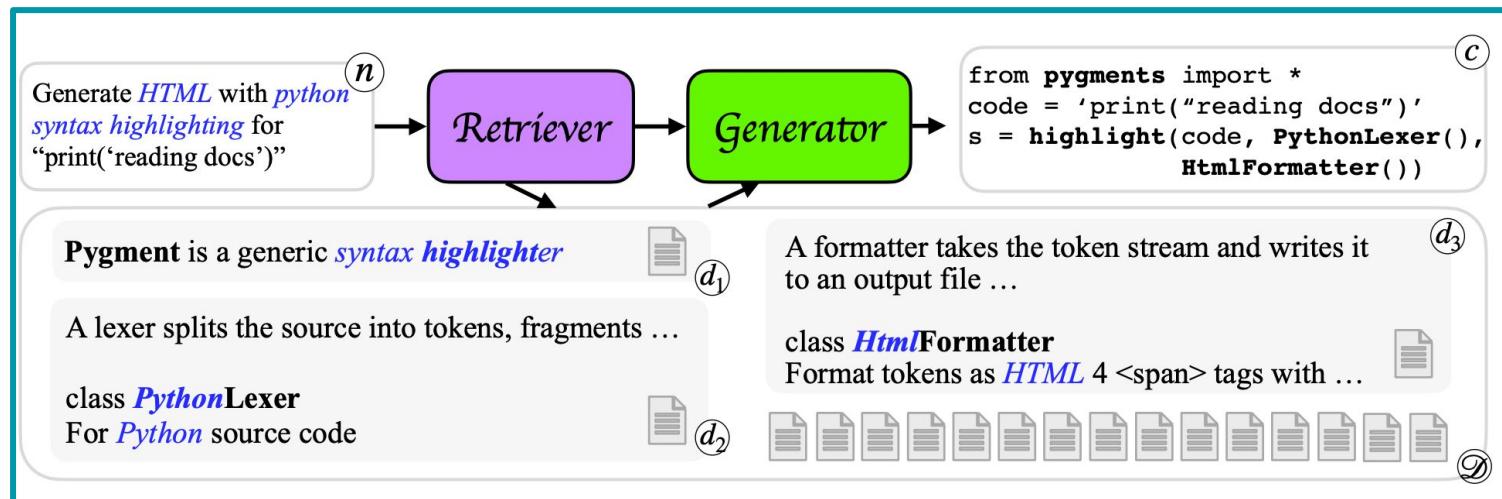
10100101...  
(idk what is  
Krylov  
iterations)

# DocPrompting

# Idea. Overall scheme

**Retriever:** Language model retrieving relevant docs ( $d_i$ ) from corpora of docs D for natural language intent n.

**Generator:** any NL->Code model.  
“Retriever is agnostic to Generator”



# Training approach

Retriever's job:

Given

n - natural language intent (n is not a number!!!)

D - document corpora

We want to maximize

$$P(c | \mathcal{D}, n)$$

# Training approach

Retriever's job:

Given

n - natural language intent (n is not a number!!!)

D - document corpora

We want to maximize

$$P(c | \mathcal{D}, n)$$

Decompose

$$P(c | \mathcal{D}, n) \approx P(c | \hat{\mathcal{D}}_n, n) \cdot P(\hat{\mathcal{D}}_n | \mathcal{D}, n)$$

where

$$\hat{\mathcal{D}}_n := \operatorname{argmax}_{\mathcal{D}_n \subseteq \mathcal{D}} P(\mathcal{D}_n | \mathcal{D}, n)$$

\*we assume c is independent of D given  $\mathcal{D}_n$

# Training approach

Retriever's job:

Given

n - natural language intent (n is not a number!!!)

D - document corpora

We want to maximize

$$P(c | \mathcal{D}, n)$$

Decompose

$$P(c | \mathcal{D}, n) \approx P(c | \hat{\mathcal{D}}_n, n) \cdot P(\hat{\mathcal{D}}_n | \mathcal{D}, n)$$

where

$$\hat{\mathcal{D}}_n := \operatorname{argmax}_{\mathcal{D}_n \subseteq \mathcal{D}} P(\mathcal{D}_n | \mathcal{D}, n)$$

\*we assume c is independent of D given  $\mathcal{D}_n$

$|\text{concat}(n, \mathcal{D}_n)|$  is new prompt for Generator

# Training approach

Retriever's job:

Given

n - natural language intent (n is not a number!!!)

D - document corpora

We want to maximize

$$P(c | \mathcal{D}, n)$$

thus now we have 2 separate tasks:  
Retriever maximizes 2nd probability,  
Generator maximizes 1st probability

Decompose

where

$$P(c | \mathcal{D}, n) \approx P(c | \hat{\mathcal{D}}_n, n) \cdot P(\hat{\mathcal{D}}_n | \mathcal{D}, n)$$

$$\hat{\mathcal{D}}_n := \operatorname{argmax}_{\mathcal{D}_n \subseteq \mathcal{D}} P(\mathcal{D}_n | \mathcal{D}, n)$$

\*we assume c is independent of D given D\_n

|concat(n, D\_n)| is new prompt for Generator

# Training approach

Retriever's job:

Given

n - natural language intent (n is not a number!!!)

D - document corpora

We want to maximize

$$P(c | \mathcal{D}, n)$$

Decompose

where

$$P(c | \mathcal{D}, n) \approx P(c | \hat{\mathcal{D}}_n, n) \cdot P(\hat{\mathcal{D}}_n | \mathcal{D}, n)$$

$$\hat{\mathcal{D}}_n := \operatorname{argmax}_{\mathcal{D}_n \subseteq \mathcal{D}} P(\mathcal{D}_n | \mathcal{D}, n)$$

\*we assume c is independent of D given D\_n

but how do we handle this task?

|concat(n, D^\_n)| is new prompt for Generator

# ВОТ КАК ОНО УЧИТСЯ ОЧЕНЬ ИНТЕРЕСНО

$D^*_n$  is ground truth docs for prompt n

$d^+_i$  in  $D^*_n$  are relative for prompt n

$d^-_i$  not in  $D^*_n$  are not relative for n

Retriever's job is just to generate embeddings  $h$  for both docs and prompts.

$$\mathcal{L}^r = -\log \frac{\exp(\text{sim}(\mathbf{h}_n, \mathbf{h}_{d_i^+}))}{\exp(\text{sim}(\mathbf{h}_n, \mathbf{h}_{d_i^+})) + \sum_{d_j^- \in \mathcal{B}/\mathcal{D}_n^*} \exp(\text{sim}(\mathbf{h}_n, \mathbf{h}_{d_j^-}))}$$


We define  $\text{sim}(hx, hy)$  as the cosine similarity between  $hx$  and  $hy$ .

Choosing K most relevant docs to be  $D^+_n$ .

## $D^*_n$ example

**NL** Show slurm jobs queued by a user 'xyz' every 5 seconds

**Code** `squeue -u xyz -i 5`

**squeue** is used to view job and job step for Slurm jobs

**-u** Request jobs or job steps from a list of users.

**-i** Repeatedly report the information at the interval specified

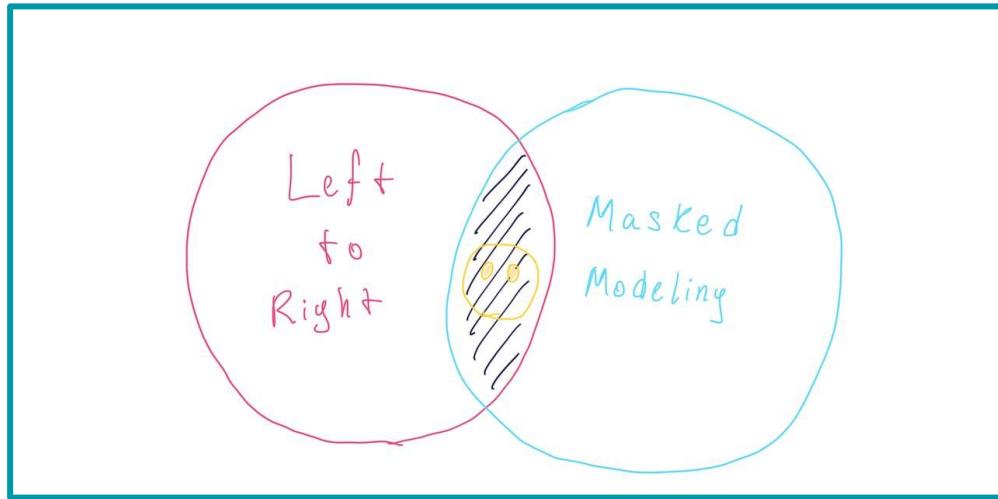
# Where the docs corpora is taken from??

Providing our model with newest docs is literally adding these docs to the corpora D. Meaning no re-training or fine-tuning needed!



## Problem №2

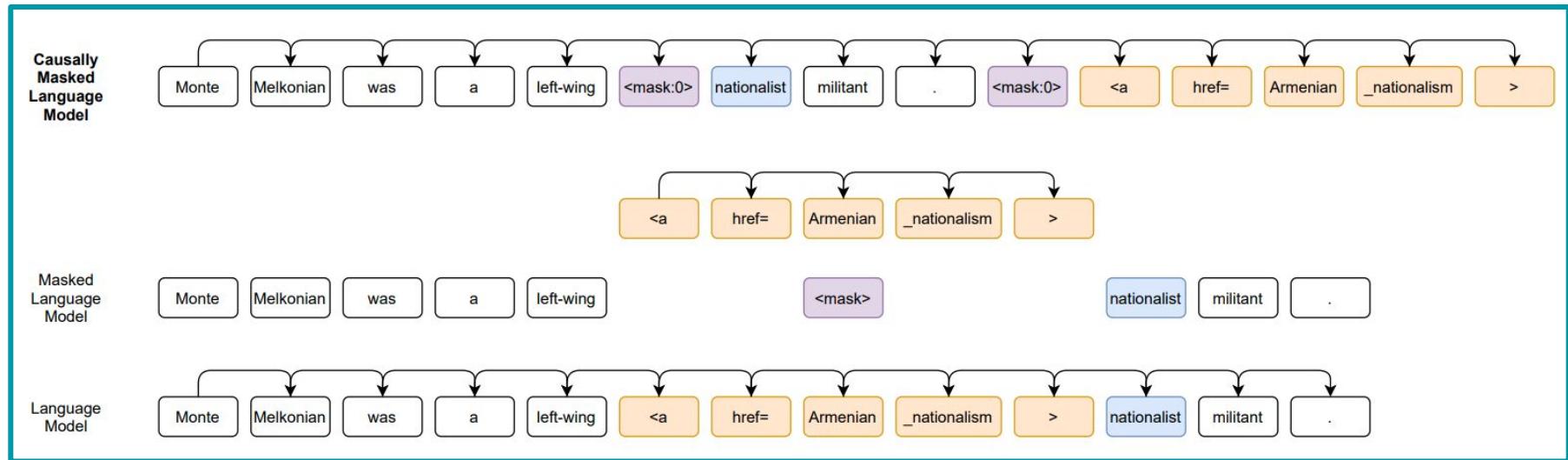
- + left-to-right generation
- infilling



- + infilling
- restricted generation

# InCoder

# Overall idea. Casual Masking (CM) briefly



- per-token generation only
- capable of left-to-right generation as so as infilling
- surprisingly able to perform a wide range of zero-shot tasks!!!

Now to InCoder itself

# Training approach

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
```

Masked 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:
        <MASK:> in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1
    return word_counts
<MASK:> word_counts = {}
for line in f:
    for word in line.split():
        if word <EOM>
```

- randomly sampling few code spans for masking
- using decoder model to predict next token in masked document
- resulted logits augmented to avoid predicting <MASK> tokens

# Why does it work?!

# Inference

## 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
```

## Variable Name Prediction

```
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
```

## Docstring Generation

```
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

```
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)
```

For common left-to-right generation just iteratively sampling next token from predicted distribution.

For infilling iteratively insert <MASK> tokens to desired positions and to the end of document, then generate until <EOM> (or other stop criterion).

# Experiment setup

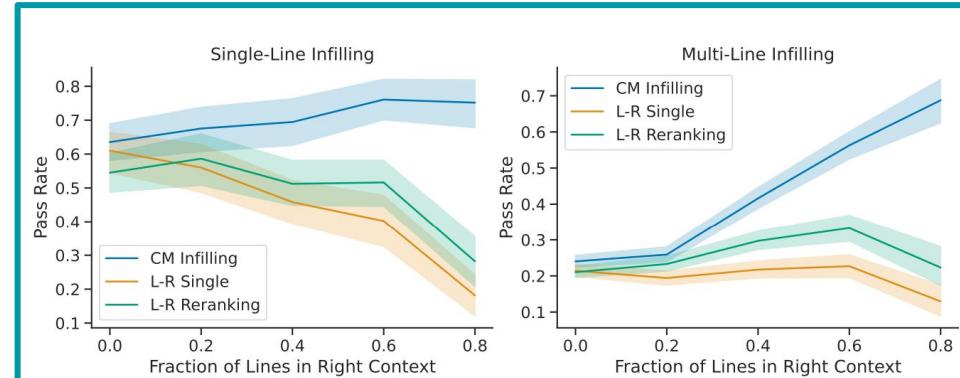
Except CM model, all over 2 baselines are used:

- Left-to-right single: generate completion according to code to the left only.
- Left-to-right reranking: generate K code-candidates according to code to the left only, choose one by substituting it into the blank and scoring the completed document.

Left-to-right single and Left-to-right reranking (K=3) example

```
a = []
Left - to - right single
    ↘ a.append(a)
a.append(a)
a.extend([0,1]) ✓
del a
a.remove(1)
```

single- and multi- line infilling results



# Results in zero-shot tasks

## docstring generation

Method	BLEU
Ours: L-R single	16.05
Ours: L-R reranking	17.14
Ours: Causal-masked infilling	18.27
RoBERTa (Finetuned)	18.14
CodeBERT (Finetuned)	19.06
PLBART (Finetuned)	19.30
CodeT5 (Finetuned)	20.36

## predicting return type (Python)

Method	Accuracy
Left-to-right single	12.0
Left-to-right reranking	12.4
Causal-masked infilling	<b>58.1</b>

## max/min cloze task

Method	Python	JavaScript	Ruby	Go	Java	PHP
Left-to-right single	76.9	77.6	65.8	70.4	74.1	77.1
Left-to-right reranking	<b>87.9</b>	90.1	76.3	92.8	<b>91.7</b>	90.4
CM infill-token	81.8	73.9	81.6	<b>95.4</b>	77.6	87.0
CM infill-region	86.2	<b>91.2</b>	78.9	94.7	89.8	<b>91.4</b>
CodeBERT	82.2	86.4		<b>86.8</b>	90.8	90.5
						88.2

## variable renaming

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

# Problem №3

# LLMs be like:

## Standard Prompting

### Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The answer is 27. 

# Program-aided Language Models: PAL

# Chain of Thought Prompting. Idea

The idea is to force model to output intermediate reasoning steps, like generating a *chain of thoughts*, which significantly improves its ability to perform complex reasoning.

Standard Prompting	Chain of Thought Prompting
<p><b>Input</b></p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p><b>Input</b></p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. <math>5 + 6 = 11</math>. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p><b>Model Output</b></p> <p>A: The answer is 27. <span style="color:red;">X</span></p>	<p><b>Model Output</b></p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had <math>23 - 20 = 3</math>. They bought 6 more apples, so they have <math>3 + 6 = 9</math>. The answer is 9. <span style="color:green;">✓</span></p>

# PAL. Idea

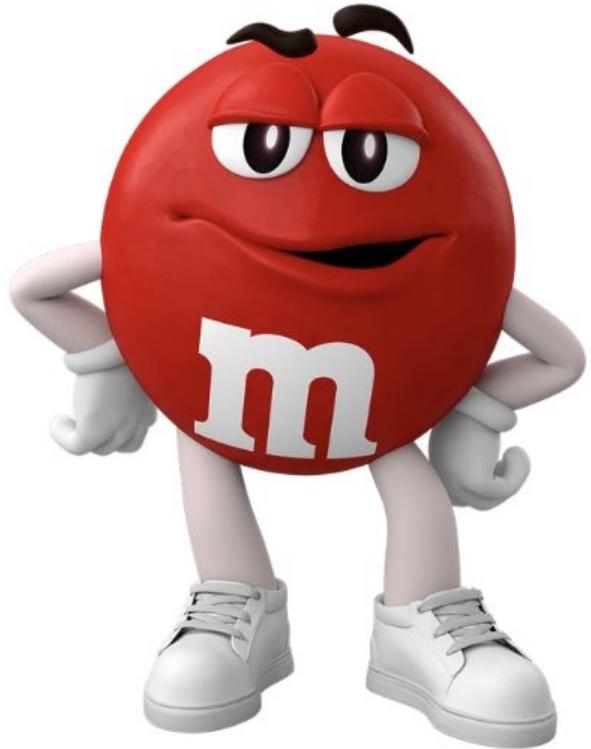
1. decompose the query on few simple steps.
2. use NL->Code model to generate code for each step, if needed
3. execute code by external interpreter

Chain-of-Thought (Wei et al., 2022)	Program-aided Reasoning (this work)
<p><b>Input</b></p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. <math>5 + 6</math>. The answer is 11.</p> <p>Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?</p> <p><b>Model Output</b></p> <p>A: The bakers started with 200 loaves. They sold 93 in the morning and 39 in the afternoon. So they sold <math>93 + 39 = 132</math> loaves. The grocery store returned 6 loaves. So they had <math>200 - 132 - 6 = 62</math> loaves left. The answer is 62.</p>	<p><b>Input</b></p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 tennis balls. <code>tennis_balls = 5</code> 2 cans of 3 tennis balls each is <code>bought_balls = 2 * 3</code> tennis balls. The answer is <code>answer = tennis_balls + bought_balls</code></p> <p>Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?</p> <p><b>Model Output</b></p> <p>A: The bakers started with 200 loaves <code>loaves_baked = 200</code> They sold 93 in the morning and 39 in the afternoon <code>loaves_sold_morning = 93</code> <code>loaves_sold_afternoon = 39</code> The grocery store returned 6 loaves. <code>loaves_returned = 6</code> The answer is <code>answer = loaves_baked - loaves_sold_morning - loaves_sold_afternoon + loaves_returned</code></p> <p><code>&gt;&gt;&gt; print(answer)</code> </p>

blue denotes NL steps, pink in grey is code, lime in black is interpreter output.

# Results

- Mathematical Reasoning
  - arithmetics
  - algebra
- Symbolic Reasoning
  - color recognition
  - “tables”
  - NL->Date
- Algorithmic Tasks
  - object counting (of exact type)
  - repeat copy (add to sequence)



*Red*