《Competition-Level Code Generation with AlphaCode》阅读报告

研究现状

现有的大型Language Model已经被证明可以用于生成代码，但是只能生成一些短小的代码片段，或者在解决复杂、不可见的、需要编程技巧的问题上，这些模型的性能还很弱。面临的挑战有：

1. 需要搜索大量的代码：

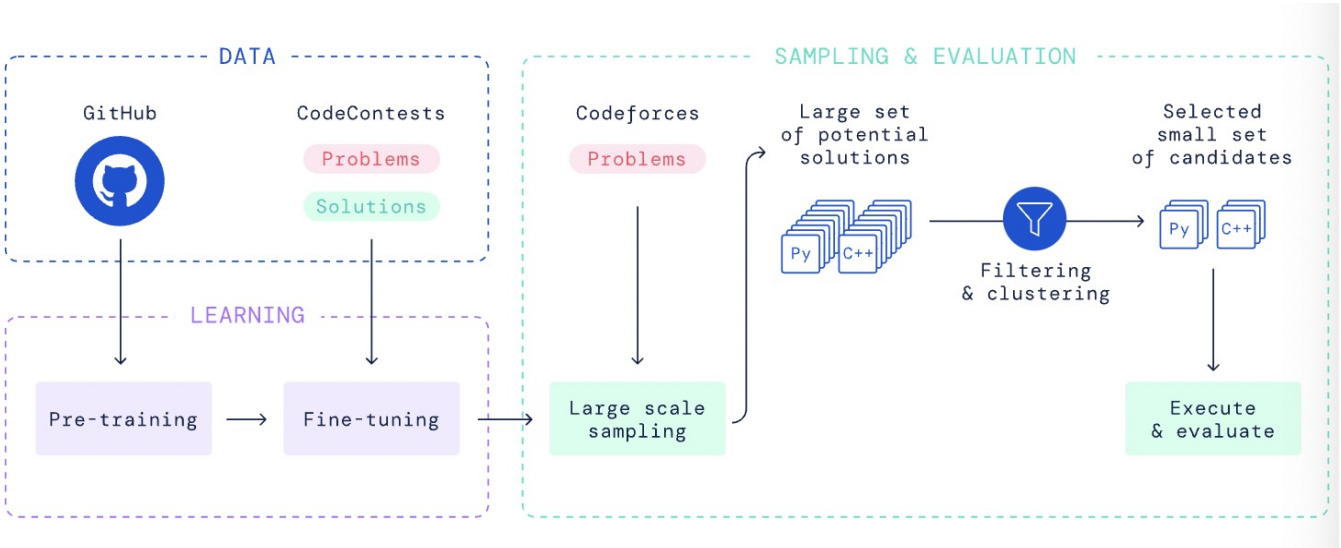
   ```
   Generating code that solves a specific task requires searching in a huge structured space of programs with a
   ```

   1.1 仅仅改变单个字符就有可能改变整个程序的逻辑，即使这没有引起崩溃，所以代码的相关性不能仅仅依靠文本字面上的相关性；
   1.2 一题多解：同一个问题可能有多种编码逻辑可供选择
   1.3 在很多（编程）领域，尤其是编程竞赛，对于每个问题常常只有有限个样本和解题方案可供训练
2. 衡量代码是否有效的测试用例通常是不可见的，需要提供一个有效的测评代码的基准。

现有的方案在生成大型程序代码上还不可靠，加上有效测试用例的缺乏，使得存在较高的假阳性率（false positive rate）。

本文的任务：

1. 提出一个生成代码的新新方案：AlphaCode，使用大型transformer模型，用GitHub上的代码pre-train，然后在精选过的编程竞赛问题上fine-tuning，总体流程如下：



```
1.1 pre-train
1.2 fine tune
1.3 Large scale sampling: Generate a very large number of samples fomr our models for each problem.
1.4 Filter the samples to obtain a small set of candidate submissions (at most 10), to be evaluated on the hidder
```

2. 发布新的关于编程竞赛数据集CodeContests，用于模型训练：
3. 证明本文生成代码的方法并非直接复制训练集的代码片段，而是根据自然语言描述生成的。

本文的创新点在于：便捷、高效的sampling和filtering解题目代码上。

研究方法

数据集构建: CodeContests

1. 数据结构包含：
   1.1 题目难度等级
   1.2 解题方法归类标签，如"greedy"、"dp"
   1.3 编程者的正确、错误的提交，编程语言有C++、Python、Java
   1.4 测试用例，含题目自带的样例测试用例（example test）和评分用的测试用例（hidden test cases）

数据样本如下：

1

Problem (input)

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

Note

Consider the example test from the statement.

In order to obtain "ba" from "ababa", you may press Backspace instead of typing the first and the fourth characters.

There's no way to obtain "bb" while typing "ababa".

There's no way to obtain "aaaa" while typing "aaa".

In order to obtain "ababa" while typing "aababa", you have to press Backspace instead of typing the first character, then type all the remaining characters.

AlphaCode

2

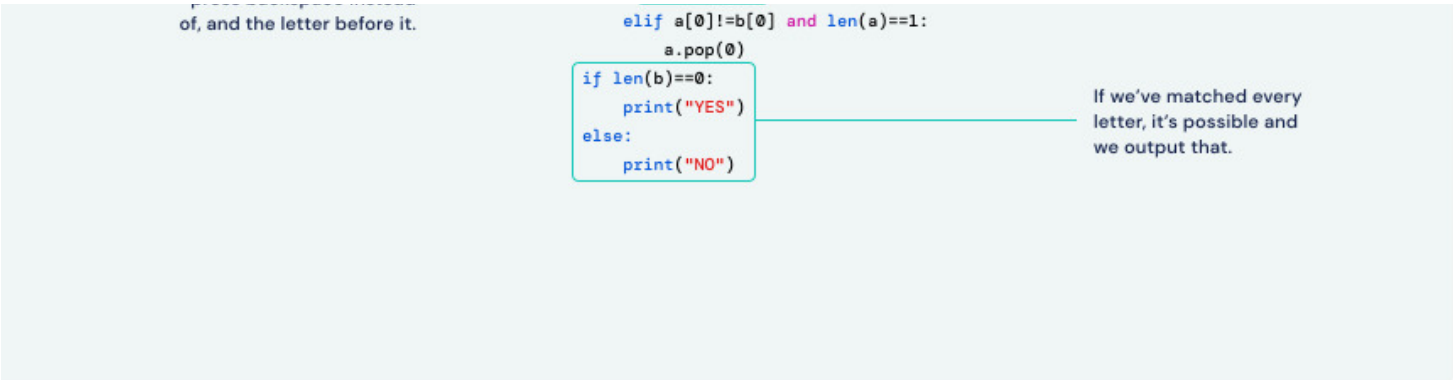
Solution (output)

```
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)
```

First AlphaCode reads the two phrases.

If the letters at the end of both phrases don't match, the last letter must be deleted. If they do match we can move onto the second last letter and repeat.

Backspace deletes two letters. The letter you press backspace instead



1. AlphaCode is presented with a problem, in this case to figure out if it's possible to convert one phrase to another by pressing backspace instead of typing some letters.

2. AlphaCode reads the whole problem statement and produces code, analogous to how a human would approach the problem by reading it, coding a solution, and submitting.

2. 为了防止数据“泄漏”（将训练集用于模型测试），本文对整个数据作了如下划分：所有训练集都在GitHub提交的日期2021/07/14或其之前；验证集的提交在2021/07/15至2021/09/20期间；测试集的提交2021/09/21之后

模型概述

1. 总体基于transformer的seq2seq架构，对条件概率建模$p(Y|X)$，其中X为编程问题的描述（encoder的输入），Y为自回归的输出一个个代码token(decoder的输出)。

Encoder Input X:

```
// RATING: 1200
// TAGS: math
// LANGUAGE IS cpp
// CORRECT SOLUTION
// n towns are arranged in a circle sequentially. The towns are numbered from 1
// to n in clockwise order. In the i-th town, there lives a singer with a
// repertoire of a_i minutes for each i ∈ [1, n].
//
// Each singer visited all n towns in clockwise order, starting with the town he
// lives in, and gave exactly one concert in each town. In addition, in each
// town, the i-th singer got inspired and came up with a song that lasts a_i
// minutes. The song was added to his repertoire so that he could perform it in
// the rest of the cities.
//
// Hence, for the i-th singer, the concert in the i-th town will last a_i
// minutes, in the (i + 1)-th town the concert will last 2 · a_i minutes, ...,
// in the ((i + k) mod n + 1)-th town the duration of the concert will be (k +
// 2) · a_i, ..., in the town ((i + n - 2) mod n + 1) - n · a_i minutes.
//
// You are given an array of b integer numbers, where b_i is the total duration
// of concerts in the i-th town. Reconstruct any correct sequence of positive
// integers a or say that it is impossible.
//
// Input
//
// The first line contains one integer t (1 ≤ t ≤ 10^3) - the number of test
// cases. Then the test cases follow.
//
// Each test case consists of two lines. The first line contains a single
// integer n (1 ≤ n ≤ 4 · 10^4) - the number of cities. The second line contains
// n integers b_1, b_2, ..., b_n (1 ≤ b_i ≤ 10^{9}) - the total duration of
// concerts in i-th city.
//
// The sum of n over all test cases does not exceed 2 · 10^5.
//
// Output
//
// For each test case, print the answer as follows:
//
// If there is no suitable sequence a, print NO. Otherwise, on the first line
// print YES, on the next line print the sequence a_1, a_2, ..., a_n of n
// integers, where a_i (1 ≤ a_i ≤ 10^{9}) is the initial duration of repertoire
// of the i-th singer. If there are multiple answers, print any of them.
//
// Example
//
// Input
//
//
// 4
// 3
// 12 16 14
// 1
// 1
// 3
// 1 2 3
// 6
// 81 75 75 93 93 87
//
//
// Output
//
//
// YES
// 3 1 3
// YES
// 1
// NO
// YES
// 5 5 4 1 4 5
//
// Note
//
// Let's consider the 1-st test case of the example:
//
//   1. the 1-st singer in the 1-st city will give a concert for 3 minutes, in
// the 2-nd - for 6 minutes, in the 3-rd - for 9 minutes;
//   2. the 2-nd singer in the 1-st city will give a concert for 3 minutes, in
// the 2-nd - for 1 minute, in the 3-rd - for 2 minutes;
//   3. the 3-rd singer in the 1-st city will give a concert for 6 minutes, in
// the 2-nd - for 9 minutes, in the 3-rd - for 3 minutes.
```

Decoder Output Y:

```
#include <bits/stdc++.h>
using namespace std;
void solve() {
  long long int n, i, j;
  cin >> n;
  vector<long long int> a(n);
  for (i = 0; i < n; i++) cin >> a[i];
  long long int sum = 0;
  for (i = 0; i < n; i++) sum += a[i];
  if (sum % (n * (n + 1) / 2) != 0) {
    cout << "NO" << '\n';
    return;
  }
  long long int k = sum / (n * (n + 1)
      / 2);
  vector<long long int> ans(n);
  for (i = 0; i < n; i++) {
    long long int t = 0;
    if (i == 0)
      t = a[n - 1];
    else
      t = a[i - 1];
    long long int p = k - (a[i] - t);
    if (p % n != 0) {
      cout << "NO" << '\n';
      return;
    }
    ans[i] = p / n;
    if (ans[i] <= 0) {
      cout << "NO" << '\n';
      return;
    }
  }
  cout << "YES" << '\n';
  for (i = 0; i < n; i++) cout << ans[i
      ] << " ";
  cout << '\n';
}
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  long long int t;
  cin >> t;
  while (t--) {
    solve();
  }
}
```

本文还发现：使用浅层的encoder(层数少)和深层的decoder（层数多）的搭配可以极大改善模型的性能。

各种模型配置如下：

Name	n_{params}	d_{model}	Heads		Blocks		Training		
			Query	KV	Enc	Dec	Batch	Steps	Tokens
AlphaCode 300M	284M	768	6	1	4	24	256	600k	354B
AlphaCode 1B	1.1B	1408	11	1	5	30	256	1000k	590B
AlphaCode 3B	2.8B	2048	16	1	6	36	512	700k	826B
AlphaCode 9B	8.7B	3072	24	4	8	48	1024	530k	1250B
AlphaCode 41B	41.1B	6144	48	16	8	56	2048	205k	967B

2. 使用JAX、Haiku工具建立模型
3. 使用multi-query attention：每个attention block使用全量的query heads，而key和value heads只使用一部分（共享key和value heads），这样可以减少内存和cache的使用，提高sampling的效率。
4. tokenize：使用SentencePiece tokenizer方法，使用GitHub和自身CodeContest数据集一共8000个token，encoder和decoder都使用相同的tokenizer

Pre-training（训练阶段）

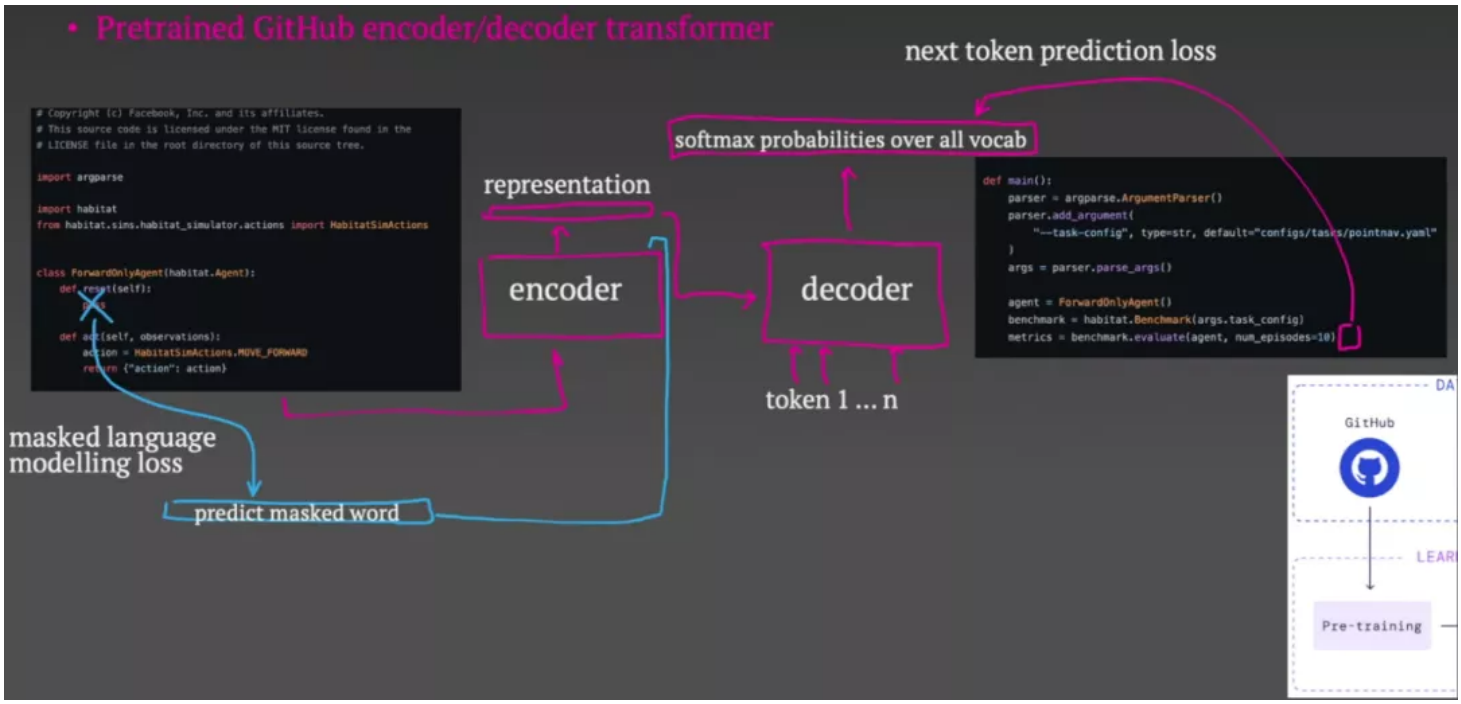
使用GitHub的代码进行预训练。

encoder使用masked language modeling
decoder使用标准的交叉熵损失预测下一个token

从GitHub抓取代码文件，文件中标记一个所谓的pivot point，将代码文件匀切分为两部分，前半部分作为encoder的输入，decoder用于重建后半部分，以自回归方式，一个个代码token预测，直到结束标记。

本文还使用masked launguage modeling技术作为第二个loss，目标是恢复被mask掉的那个token。

架构如下：



(by https://www.youtube.com/watch?v=YjsoN5aJChA)

Fine-tuning（训练阶段）

使用自身CodeContests数据进行模型微调。

同样，encoder使用masked language modeling
decoder使用标准的交叉熵损失预测下一个token。

encoder输入为问题的自然语言描述，而解题的代码用于decoder。

另外，本文还使用了以下一些技巧：

1. Tempering:

```
Tempering, introduced by Dabre and Fujita (2020)([ Softmax tempering for training neural machine translation mode
the output logits of a model by a scalar temperature  before the softmax layer
```

softmax tempering旨在解决NMT模型过拟合问题，因为 softmax分布很快接近黄金标签分布。

旧的计算交叉熵损失的方法：

$P_i = P(Y_i|Y_{<i}, X) = softmax(D_i)$
$loss_i = -\langle log(P_i), L_i \rangle$，其中$\langle ., . \rangle$为点积运算

softmax tempering计算方法：

$P_i^{temp} = softmax(D_i/T)$
$loss_i^{temp} = -\langle log(P_i^{temp}), L_i \rangle \cdot T$

其中D_i为decoder第i个位置的输出，L_i为对应标签的one-hot向量。

当T大于1.0时，softmax输出的概率会更加平滑（smoother probability distribution），分布越平滑越均匀，熵就越高，因此预测时就有更多的不确定性。

```
Because loss is to be minimized, back-propagation will force the model to generate logits to counter the smoothir
softmax distribution compared to those of a model trained without softmax tempering
```

(关于logits的理解：深度学习中经常出现logits，应该是表示进入softmax之前神经层的输出)

2. Value conditioning & prediction

数据集包含一道问题的正确和错误的提交，本文使用Value conditioning & prediction区分这两类的提交。

在Value conditioning阶段，在问题描述中插入这个提交是否正确的信息，如下：

```
RATING: 1200
TAGS: dp,implementation
LANGUAGE IS python3
CORRECT SOLUTION
Polycarp must pay exactly n burles at the checkout ... (rest of the description)
```

然后在采样阶段，solution都填“正确”，模型就会采样到正确的sample了。

而在Value prediction阶段，会加入一个辅助的预测任务（训练中才有），以便在transformer中的最后一层输出也可以用来区分这个代码提交的正确与否：

```
we added an auxiliary value prediction task during training such that the last layer token representations before
```

3. GOLD

每个问题可以有多种解法，因为有不同的算法选择、实现方式等。那么每个问题描述就会对应多个解法。使用标准的最大似然估计去最小化损失函数是通过赋予每个解法不同的权重（类似于recall），而本文希望模型尽可能只生成一个正确解法（类似于precision），于是求loss导数采用了一种offine RL算法（GOLD: Text Generation by Learning from Demonstrations），其求导如下：

$$\nabla \pounds_{GOLD}(\theta) = -\sum_{s\epsilon \quad Solution} \quad \sum_{tokens} P_\theta(s) \nabla log P_\theta(s)$$

其中θ是模型参数，$logP_\theta(s)$是标准的最大似然估计，$P_\theta(s)$赋予一个权重，让模型学习似然值高的tokens，而忽略其他tokens，这样模型就会更加关注与精确率而不是召回率，增加只找到一个精确解的概率。
同时，为了降低训练的稳定性，对$P_\theta(s)$作如下限制：
$(P_\theta(s)^\alpha, \beta), \alpha = \frac{1}{2}, \beta = 0.05$

（这里借鉴了强化学习中策略梯度方法的Loss:
$\nabla J(\theta) = \mathbb{E}[Q(s,a)\nabla \pi_\theta(s,a)]$）

大规模采样（测试阶段）

训练一个transformer（其实就是pre-train和fine-tune那个transformer）负责解题solution的采样。将问题描述与问题的各种元数据（如问题tag、rating等）作为输入，代码sample作为输出。每个问题生成100万个samples。
为使生成的样本多样化，使用如下技巧：
（1）生成Python和C++语言的样本各一半；
（2）随机组合题目的tag和ratings；
（3）使用相对高的softmax tempering参数

过滤（测试阶段）

从大规模采样的samples中，只挑选通过题目中example test的的sample，最终将过滤掉大概99%的samples

聚类（测试阶段）

首先，训练另一个transfomer（跟以上架构相同）用来生成测试输入，使用问题的描述（含其中的example test、hidden test和生成的test）训练数据，输出与问题相关的测试输入（test input）。

然后，对以上过滤剩下的samples（约1000个），分别用transformer生成的测试输入验证测试输出，根据测试输出结果是否相似进行聚类，最后挑选出10个sample。

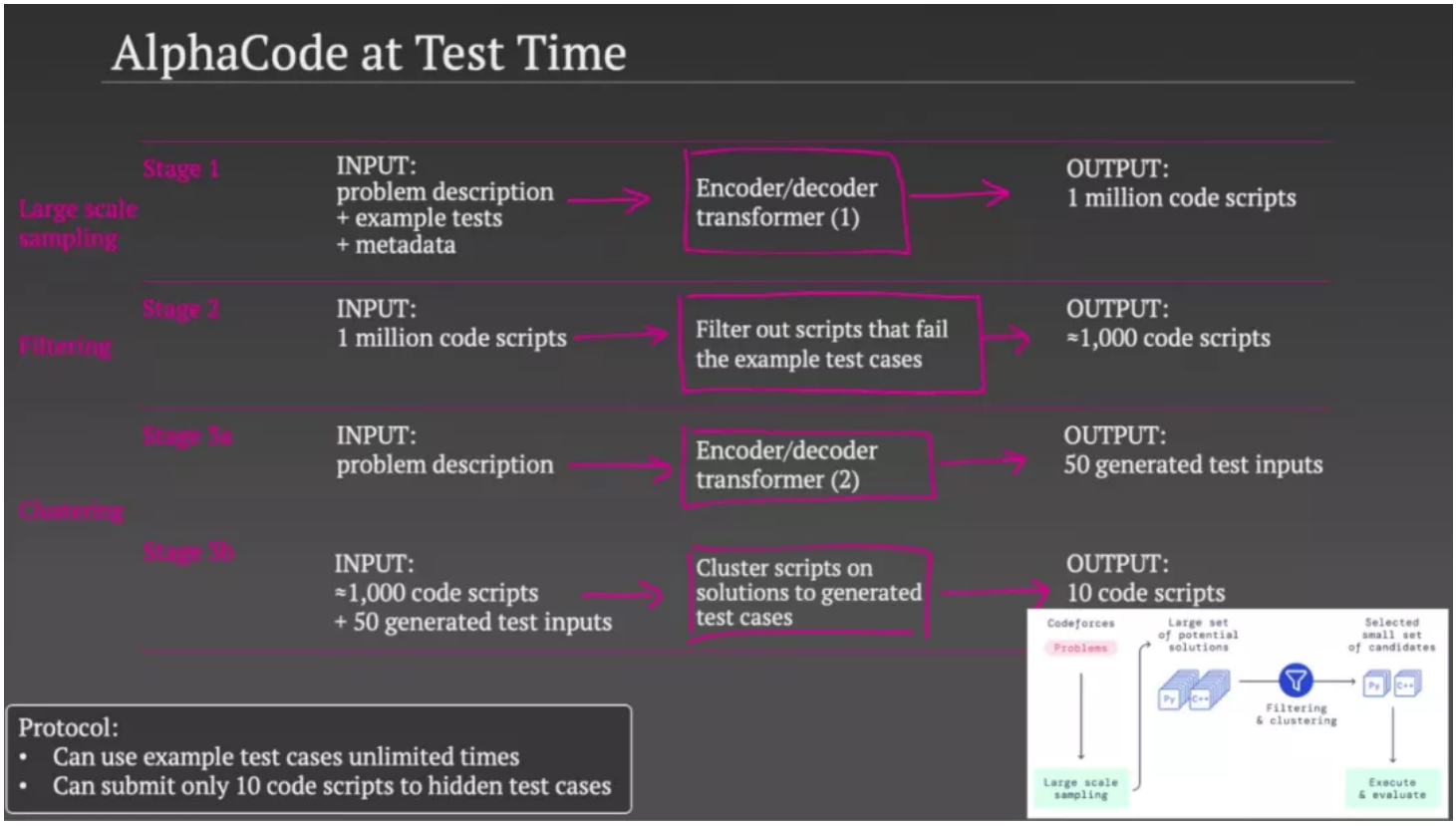
相对随机抽样，这么做的好处是：对于语义相似的代码，通常输出相同的测试结果。按测试结果是否相似对这些语义相似的代码提交进行聚类，就可以避免提交相同解答的代码，可以大大节省提交验证的时间（这一步其实还在进行过滤，最终目标只要模型生成一个提交就足够了）。

```
Semantically equivalent programs could be detected if we had additional test inputs, by executing all remaining p
```

研究发现，以下挑选样本的方法使得研究结果最好：先从最大的聚类开始挑选样本，每个类只选1个样本，直到最小的聚类，然后第二轮开始只从最大（或较大）的聚类选sample，直到sample数量达到10个。

验证时，如果这10个sample中有一个通过了所有的hidden test，那就算成功解决了这个编程问题。

测试总体流程如下图：



(by https://www.youtube.com/watch?v=YjsoN5aJChA)

研究结论

1. 在解决10个编程问题中，本模型平均达到54.3%的排名：

Contest ID	1591	1608	1613	1615	1617	1618	1619	1620	1622	1623	Average
Best	43.5%	43.6%	59.8%	60.5%	65.1%	32.2%	47.1%	54.0%	57.5%	20.6%	**48.4%**
Estimated	44.3%	46.3%	66.1%	62.4%	73.9%	52.2%	47.3%	63.3%	66.2%	20.9%	**54.3%**
Worst	74.5%	95.7%	75.0%	90.4%	82.3%	53.5%	88.1%	75.1%	81.6%	55.3%	**77.2%**

2. 解题率

- pass@k: k个sample都通过题目hidden test的，即表示该题正确解决。这个标准表示每k个sample正确解题比例，可以用来衡量模型寻找可解sample的性能（每k个sample中pass的比率）
- 10@k: 每k个sample中有10个都能正确解题的解题数量占比。

具体计算n@k流程如下：

Algorithm 1 Algorithm for computing n@k with filtering using example tests.

Input n = the number of allowed submissions in $n@k$
Input k = the number of allowed samples in $n@k$
Input e_p = the number of samples which pass the example tests for each problem p
Input s_p = the number of samples which solve the problem (pass all tests) for each problem p
Input K = the number of samples actually taken per problem
Hyperparameter S = the number of subsamples to use for calculation

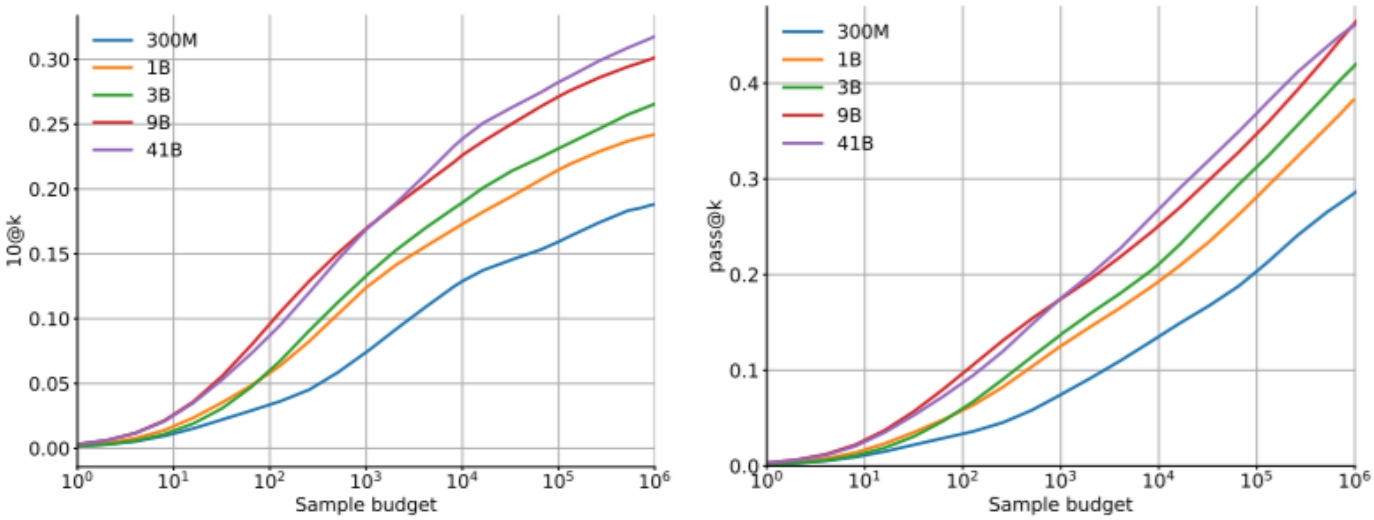
for each problem p in the problem set **do**
  for each of the S subsamples **do**
    Sample e'_p ~Hypergeometric$(e_p, K - e_p, k)$ ▸ # samples out of k which pass examples tests.
    $n' \leftarrow \min(e'_p, n)$ ▸ Only submit samples that pass the example tests.
    Sample s'_p ~Hypergeometric$(s_p, e_p - s_p, n')$ ▸ # correct solutions out of n' submissions.
    solved$_p$ = 1 if $s'_p > 0$ else 0 ▸ Problem is solved if any submission is correct.
  end for
  Compute $n@k$ for this problem as the average of all solved$_p$.
end for
return the average $n@k$ across all problems.

简要说明：
2.1 在全集K个samples中取不重复的S个子集，每个子集含k个samples
2.2 对每个子集S，进行如下操作：
2.2.1 当sample通过题目的example test时就作为有效提交，最多提交n个
2.2.2 提交的samples中，当有一个sample能满足题解，则表示该题得解，统计得解的题目数量 #solved
2.3 求#solved / #S 得对于每个单个题目的平均n@k
2.4 除以所有题目的数量，得总的平均n@k

本模型解题率统计：

Approach	Validation Set				Test Set		
	10@1k	10@10k	10@100k	10@1M	10@1k	10@10k	10@100k
9B	16.9%	22.6%	27.1%	30.1%	14.3%	21.5%	25.8%
41B	16.9%	23.9%	28.2%	31.8%	15.6%	23.2%	27.7%
41B + clustering	21.0%	26.2%	31.8%	34.2%	16.4%	25.4%	29.6%

解题率与总的sample数量呈线性对数关系，而且模型规模越大，曲线斜率越大：



(a) 10 attempts per problem

(b) Unlimited attempts per problem

3. 模型架构的变化的影响

由于解题率取决于sampling数量的大小，所以sampling的速度很重要，以下衡量不同的模型架构对sampling的速度的影响：

Model	Blocks		Seq. length		Hidden Size	Fan-Out Ratio	Params	Samples / TPU sec	10@10K
	Enc.	Dec.	Enc.	Dec.					
AlphaCode model	5	30	1536	768	1408	6	1.15B	4.74	17.3%
Decoder-only	-	40	-	2304	1408	6	1.17B	1.23	18.5%
Std MH attention	5	30	1536	768	1408	4.3	1.16B	0.37	17.0%

其中Std MH attention表示使用标准的多头注意力。

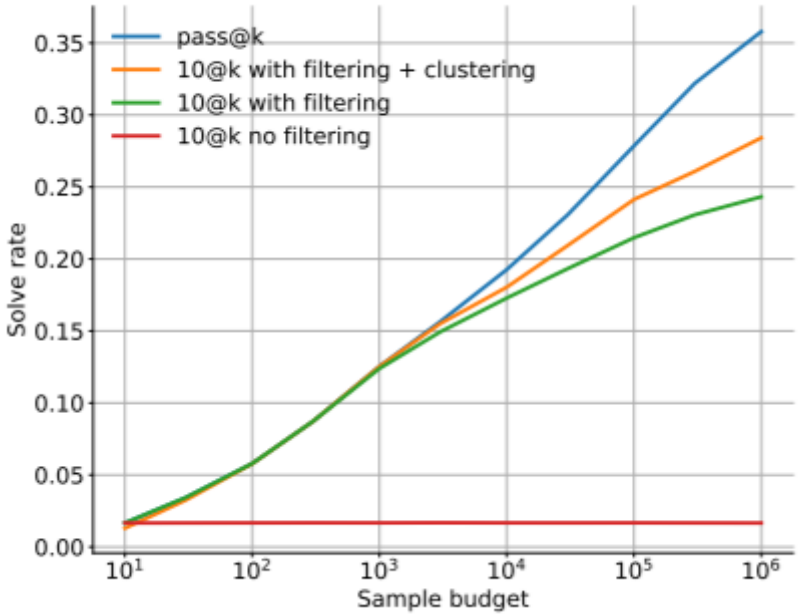
4. 预训练数据的影响

Pre-training dataset	Solve rate 10@1K	10@10K	10@100K
No pre-training	4.5%	7.0%	10.5%
GitHub (Python only)	5.8%	11.1%	15.5%
MassiveText	9.7%	16.1%	21.2%
GitHub (all languages)	12.4%	17.3%	21.5%

5. 模型各种提升性能的技巧影响

Fine-tuning setting	Solve rate 10@1k	10@10k	10@100k	10@1M
No Enhancements	6.7% (6.5-6.8)	10.4% (9.6-11.0)	15.2% (14.3-15.9)	19.6% (18.2-20.4)
+ MLM	6.6% (6.2-7.0)	12.5% (12.1-12.7)	17.0% (16.5-17.2)	20.7% (19.1-21.3)
+ Tempering	7.7% (7.2-8.5)	13.3% (12.5-13.8)	18.7% (18.0-19.2)	21.9% (20.7-22.6)
+ Tags and Ratings	6.8% (6.4-7.0)	13.7% (12.8-14.9)	19.3% (18.1-20.0)	22.4% (21.3-23.0)
+ Value	10.6% (9.8-11.1)	16.6% (16.4-16.9)	20.2% (19.6-20.7)	23.2% (21.7-23.9)
+ GOLD	12.4% (12.0-13.0)	17.3% (16.9-17.6)	21.5% (20.5-22.2)	24.2% (23.1-24.4)
+ Clustering	12.2% (10.8-13.4)	18.0% (17.3-18.8)	24.1% (23.2-25.0)	28.4% (27.5-29.3)

6. 使用sample过滤和聚类的影响：



启示

1. multi-query attention对transformer计算性能的影响
2. 解题率跟样本数呈线性对数关系，生成样本越多，解题率就可以越高，所以sampling的速度很重要（所谓大力出奇迹）。

3. 题目自然语言描述的精简性对性能提升极大：

Problem	Original	Simplified
1554A Cherry	2.98%	15.74%
1559A Mocha and Math	12.25%	55.53%
1569A Balanced Substring	10.61%	31.97%
No consecutive zeros	0.17%	1.25%
Nim	0.95%	85.38%

Appendix Table A9 | **Performance on original vs. simplified problems**. The percentage of correct samples from a total of 100k samples, for original problem wordings and rewordings which make the required algorithm more explicit. This result was obtained using a 1B parameter model.

精简前后的题目对比：

1554A Cherry – Original

```
You are given n integers a_1, a_2, ..., a_n. Find the
 maximum value of max(a_l, a_{l + 1}, ..., a_r) . min
(a_l, a_{l + 1}, ..., a_r) over all pairs (l, r) of
integers for which 1 <= l < r <= n.

Input

The first line contains a single integer t (1 <= t <=
 10 000) - the number of test cases.

The first line of each test case contains a single
integer n (2 <= n <= 10^5).

The second line of each test case contains n integers
 a_1, a_2, ..., a_n (1 <= a_i <= 10^6).

It is guaranteed that the sum of n over all test
cases doesn't exceed 3 . 10^5.

Output

For each test case, print a single integer - the
maximum possible value of the product from the
statement.

Example

Input


4
3
2 4 3
4
3 2 3 1
2
69 69
6
719313 273225 402638 473783 804745 323328


Output


12
6
4761
381274500335

Note

Let f(l, r) = max(a_l, a_{l + 1}, ..., a_r) . min(a_l
, a_{l + 1}, ..., a_r).

In the first test case,

  * f(1, 2) = max(a_1, a_2) . min(a_1, a_2) = max(2,
4) . min(2, 4) = 4 . 2 = 8.
  * f(1, 3) = max(a_1, a_2, a_3) . min(a_1, a_2, a_3)
 = max(2, 4, 3) . min(2, 4, 3) = 4 . 2 = 8.
  * f(2, 3) = max(a_2, a_3) . min(a_2, a_3) = max(4,
3) . min(4, 3) = 4 . 3 = 12.
```

1554A Cherry – Simplified

```
You are given n integers a_1, a_2, ..., a_n. Find the
 maximum value of a_l times a_{l + 1} for an integer
l for which 1 <= l < n.

Input

The first line contains a single integer t (1 <= t <=
 10 000) - the number of test cases.

The first line of each test case contains a single
integer n (2 <= n <= 10^5).

The second line of each test case contains n integers
 a_1, a_2, ..., a_n (1 <= a_i <= 10^6).

It is guaranteed that the sum of n over all test
cases doesn't exceed 3 . 10^5.

Output

For each test case, print a single integer - the
maximum possible value of the product from the
statement.

Example

Input


4
3
2 4 3
4
3 2 3 1
2
69 69
6
719313 273225 402638 473783 804745 323328


Output


12
6
4761
381274500335
```

4. 解题正确率十分依赖于题目的自然语言描述

Rewording	Solve rate
Original (maximum product of two consecutive array elements)	17.1%
Opposite (minimum product of two consecutive array elements)	0.1%
Related (maximum product of two any array elements)	3.2%
Underspecified (maximum function of two consecutive array elements)	0.03%
Verbose	19.4%
Algorithm described in words	19.7%

Appendix Table A10 | **Rewording the *Cherry* problem**. The percentage of solutions in 50000 samples from the 1B parameter model when attempting the simplified version of the *Cherry* problem with different rewordings.

```
the model acutally does better with more language-heavy descriptions
```

5. 模型双loss的应用（标准交叉熵和masked language modeling的loss）
6. 没用按照不同的开源许可证对代码区分对待，用那些明确不允许用于商业用途的代码训练模型，是否涉嫌侵权？模型的输出代码与原训练代码的相似性判断很关键！

附：

- 论文地址：
  https://storage.googleapis.com/deepmind-media/AlphaCode/competition_level_code_generation_with_alphacode.pdf
- 数据集：
  https://github.com/deepmind/code_contests
  https://codeforces.com/
- 应用：
  https://alphacode.deepmind.com/