Inference Algorithms

Daniel Fried

11-891: Neural Code Generation

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

Language
Technologies
Institute

With slides from Graham Neubig

Single-Sample Generation

Autoregressive Code Modeling

Jupyter notebook demo

The Generation Problem

- We have an autoregressive model of P(X), how do we use it to generate an output X?
- Two methods:
  - Argmax (“mode-seeking”): Try to generate the output with the highest probability.
  - Sampling: Try to generate a random output according to the probability distribution.

Argmax type 1: Greedy Search

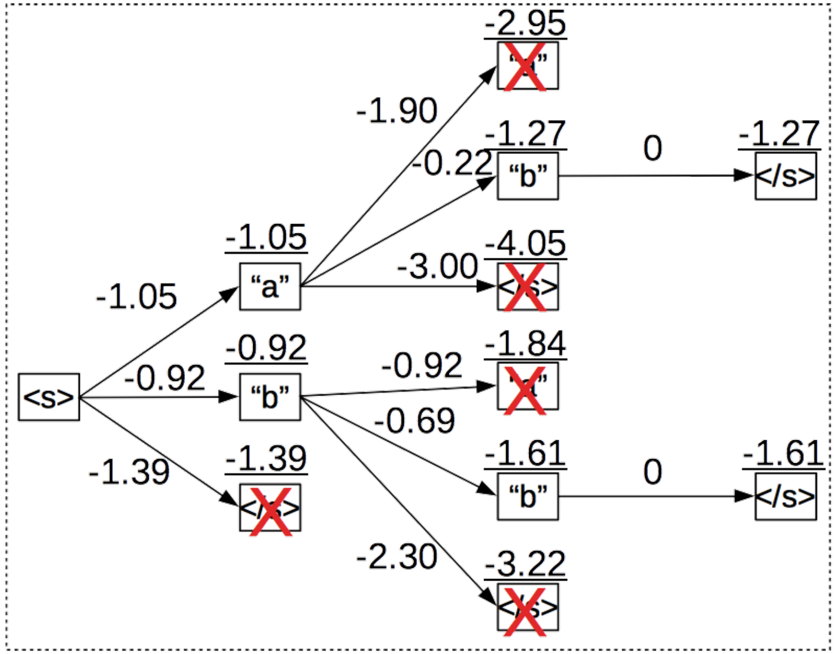
- One by one, pick the single highest-probability token

```
while x_{j-1} != "</s>":
  x_j = argmax P(x_j | x_1, ..., x_{j-1})
```

- Deterministic
- Not exact, real problems:
  - Will often generate the “easy” tokens first
  - Will prefer multiple common tokens to one rare token

Argmax type 2: Beam Search

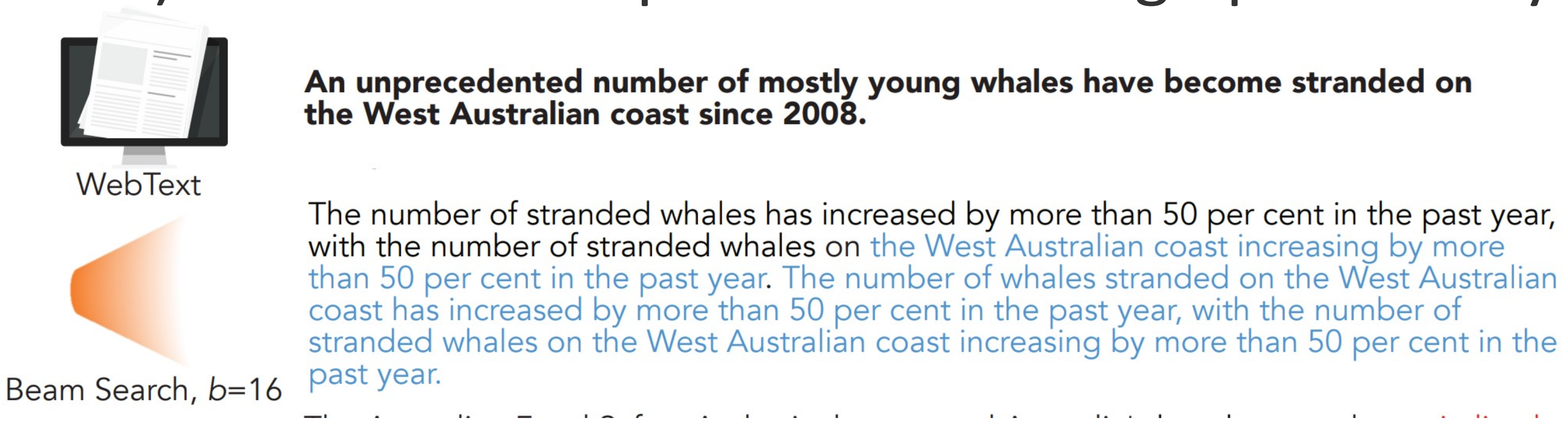
- Instead of picking one high-probability token, maintain several paths



Adjust the "beam size" *b* to spend more time on searching, find a better scoring hypothesis

Limitations of Greedy / Beam Search

- (Transformer) LMs tend to repeat text with high probability



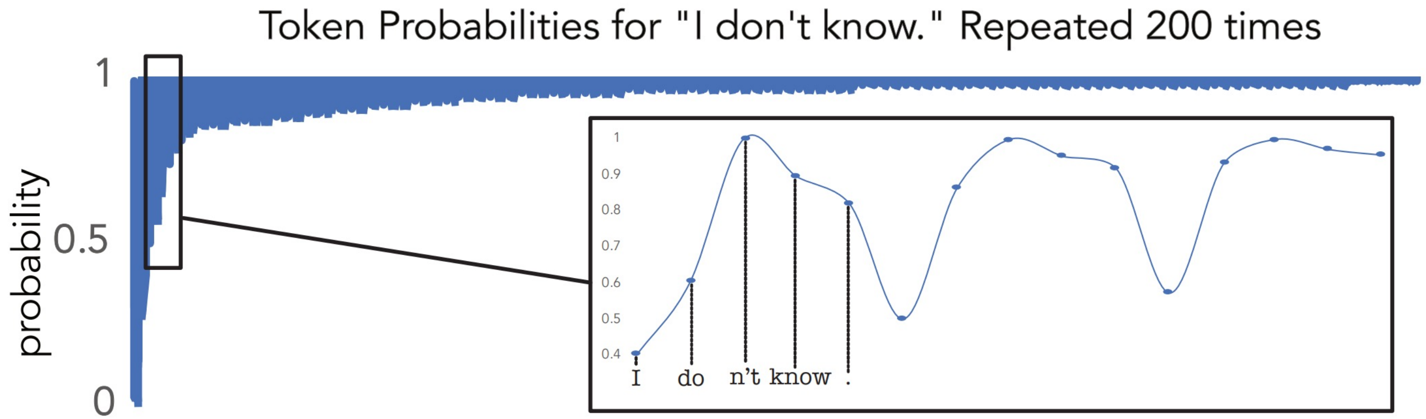


Figure 4: The probability of a repeated phrase increases with each repetition, creating a positive feedback loop. We found this effect to hold for the vast majority of phrases we tested, regardless of phrase length or if the phrases were sampled randomly rather than taken from human text.

Limitations of Greedy / Beam Search

- The most probable output may be uninformative!

Prompt:

```
def count_words(filename: str) -> Counter[str, int]:
```

Completion 1:

```
pass
```

$\log p(c_1 | prompt) = -4.69$

Completion 2:

```
words = Counter()
with open(filename, ‘r’) as f:
    for line in f.readlines():
        words.update(line.split())
return words
```

$\log p(c_2 | prompt) = -18.13$

Completion 3:

```
word_counts = Counter()
with open(filename, ‘r’) as f:
    for line in f.readlines():
        word_counts.update(line.split())
return word_counts
```

$\log p(c_3 | prompt) = -19.19$

Completion 4:

```
return Counter(tok for line in
    open(filename, ‘r’).readlines() for
    tok in line.split())
```

$\log p(c_4 | prompt) = -23.31$

Probabilities from DeepSeekCoder-6.7B

Sampling

- Randomly generate words one-by-one. (aka “ancestral sampling”)

while x_{j-1} != “</s>”:
$\quad x_j \sim P(x_j \mid x_1, \ldots, x_{j-1})$

- Maximum likelihood training assumes samples are sampled from the underlying distribution =>
samples are what your model thinks the training data looks like.

Limitations of Sampling

- Neural LMs that use a softmax assign non-zero probability to every word!
- The tail of the distribution is noisy

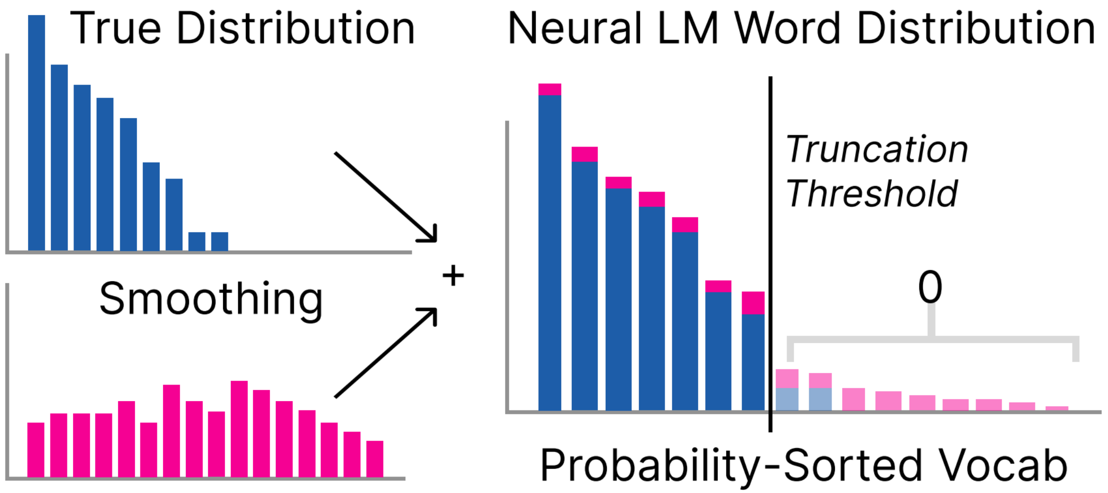


Figure 1: A neural LM as a mixture of the true distribution, and a uniform-like smoothing distribution. Truncation aims to approximate the true distribution support.

Hewitt et al. 2022.
Truncation Sampling as Language Model Desmoothing

Sampling from a Truncated Distribution

- Remove the lowest-probability words at each time step.

$P(x_6 \mid$ "The capital of Pennsylvania is"$)$

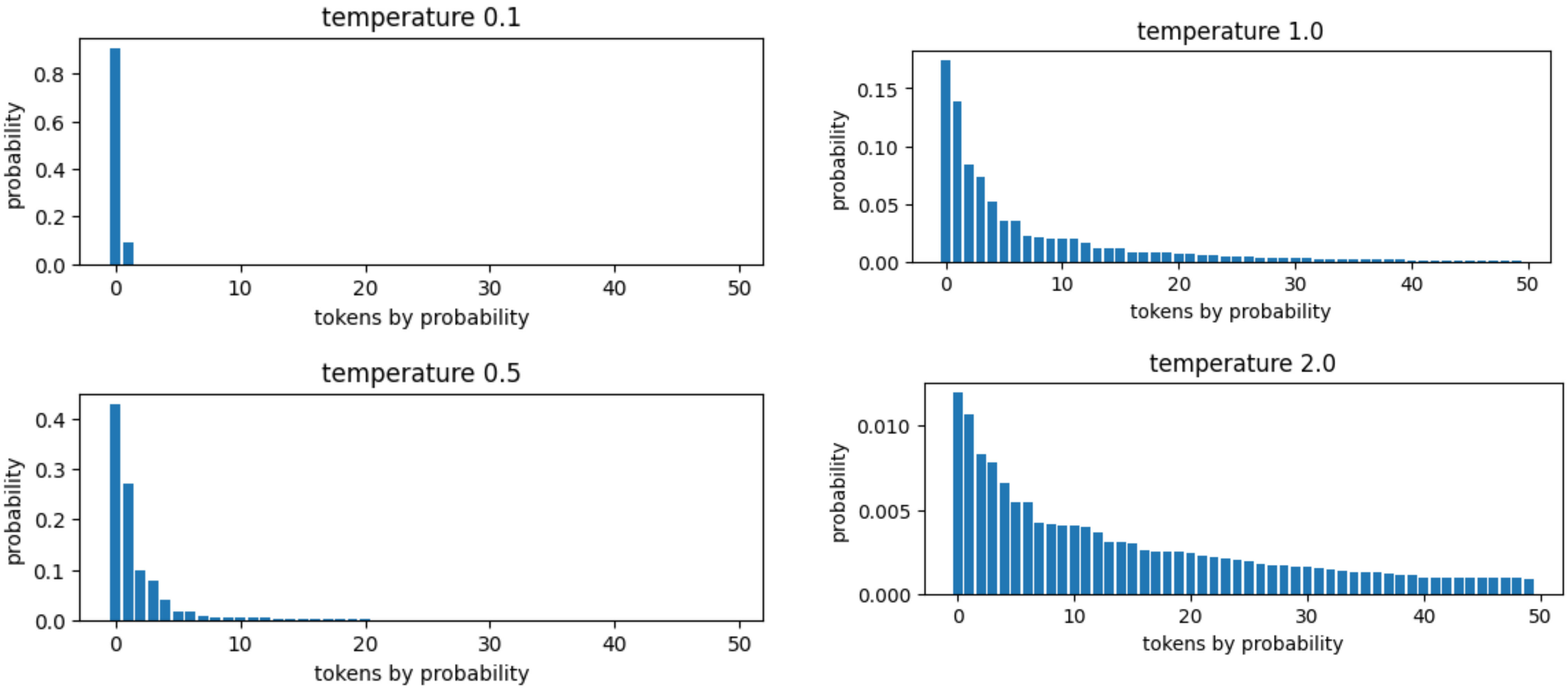
Harrisburg	34.3%
Philadelphia	31.1%
Pittsburgh	12.9%
Easton	2.2%
Lancaster	1.8%
Allentown	1.6%
Washington	1.5%

Top-k Sampling
(e.g. k=5)
Fan et al. 2018

Nucleus (top-p) Sampling
(e.g. p=0.8)
Holtzmann et al. 2019

Temperature Sampling

$$p(y) = softmax(\frac{logit}{\tau}) \propto \exp\left(\frac{logit}{\tau}\right)$$



Beware of Tokenization!

```
len = 18
510     `The`
3048    ` link`
310     ` is`
654     ` <`
66      `a`
3860    ` href`
568     `="`
2413    `http`
1358    `://`
2700    `www`
15      `.`
9906    `google`
15      `.`
681     `com`
16      `/`
8716    `search`
32      `?`
82      `q`
```

```
27      `:`
21610   `:/`
1358    `://`
1450    `::`
16      `/`
```

- What happens if your prompt ends in the middle of a token?
  - `The link is <a href=http:`
  - Last token is 27, but we want it to be 1358
  - Tokenizers are usually greedy: 27 16 16 was probably never seen in the training data, so model is unlikely to generate it
- Code models often have whitespace as part of the vocabulary, so you may get different results if you call `prompt.strip()` to remove trailing whitespace

https://towardsdatascience.com/the-art-of-prompt-design-prompt-boundaries-and-token-healing-3b2448b0be38

Conditioned Generation

- Simple approach: include meta-data as special symbols, or comments

```python
<| file ext=.py |>
# count the words in all files in the current directory
import os
import sys

def main():
  cwd = os.getcwd()

  words = 0

  for filename in os.listdir(cwd):
    if filename.endswith(".in"):
      fname = os.path.join(cwd, filename)
      with open(fname) as infile:
        for line in infile:
          words += len(line.split())

  print(words)

if __name__ == '__main__':
  main()
```

```sh
<| file ext=.sh |>
# count the words in all files in the current directory
find . -type f -name "*.txt" -exec wc -w {} \; | sort -nr | head -n 20
```

Example from InCoder, Fried et al. 2023

Chain-of-Thought Prompting

- Can we access parts of the training distribution where reasoning steps are spelled-out? Also lets the model do more steps of computation per output.

[1]
```
import pandas as pd
import matplotlib.pyplot as plt
```

[2]
```
# Exercise 1
df = pd.read_csv('scores.csv')
```

[3]
```
# Schema of Dataframes:
# Columns in df with example values:
# Stu_Name (Mike), Engineering (90), English (89), Math (92)
```

[4]
```
# Problem: Get the students with an averaged score
above 90 for science subjects.
```
(format of the answer determines the prompting method)

[4a]
```
# Solution:
df['Science_Avg'] = (df['Engineering'] + df['Math']) / 2
df[df['Science_Avg'] > 90][['Stu_Name', 'Science_Avg']]
```

[4b]
```
# Solution:
df['Science_Avg'] = (df['Engineering'] + df['Math']) / 2
df_score_above_90 = df[df['Science_Avg'] > 90]
result = df_score_above_90[['Stu_Name', 'Science_Avg']]
```

[4d]
```
# Solution: Let's solve this problem step-by-step.
# Step 1: Create a new column with the average score of
engineering and math
df['Science_Avg'] = (df['Engineering'] + df['Math']) / 2
# Step 2: Get the rows whose average score is above 90
df_score_above_90 = df[df['Science_Avg'] > 90]
# Step 3: Return the student name and average scores
result = df_score_above_90[['Stu_Name', 'Science_Avg']]
```

CoT: Wei et al. 2022. These examples: Yin et al. 2023

Zero-Shot Chain-of-Thought

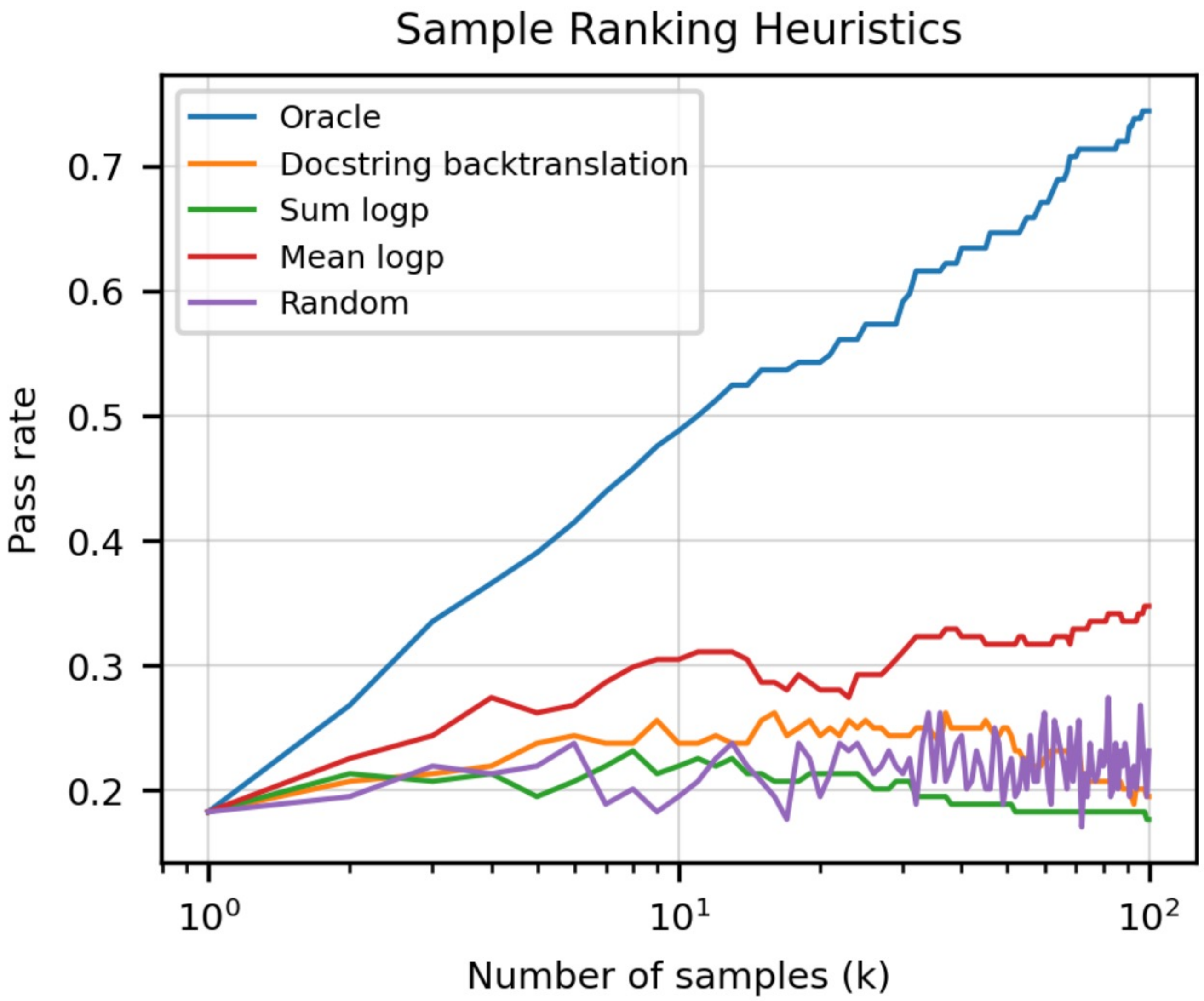
Table 4: Robustness study against template measured on the MultiArith dataset with text-davinci-
(*1) This template is used in Ahn et al. [2022] where a language model is prompted to gene
step-by-step actions given a high-level instruction for controlling robotic actions. (*2) This temp
is used in Reynolds and McDonell [2021] but is not quantitatively evaluated.

No.	Category	Template	Accuracy
1	instructive	Let's think step by step.	**78.7**
2		First, (*1)	77.3
3		Let's think about this logically.	74.5
4		Let's solve this problem by splitting it into steps. (*2)	72.2
5		Let's be realistic and think step by step.	70.8
6		Let's think like a detective step by step.	70.3
7		Let's think	57.5
8		Before we dive into the answer,	55.7
9		The answer is after the proof.	45.7
10	misleading	Don't think. Just feel.	18.8
11		Let's think step by step but reach an incorrect answer.	18.7
12		Let's count the number of "a" in the question.	16.7
13		By using the fact that the earth is round,	9.3
14	irrelevant	By the way, I found a good restaurant nearby.	17.5
15		Abrakadabra!	15.5
16		It's a beautiful day.	13.1
-		(Zero-shot)	17.7

Kojima et al. 2022

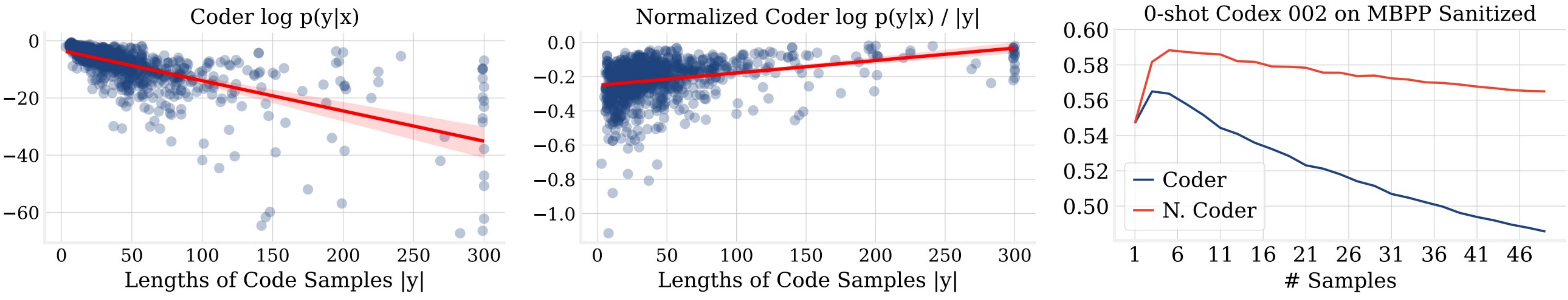
Multi-Candidate Methods

Can Rerank By Probability



[Codex paper, Chen et al. 2021]

But Beware Model Biases



(a) (b) (c)

[Coder-Reviewer reranking, Zhang et al. 2022]

Mutual Information Helps Avoid Biases

3 -shot task-agnostic prompting

Coder Prompt

```
<text>Print info of "bash"</text>
<code>echo $(ls -l /bin/bash)</code>
... 2 more demonstration examples
<text>Change the owner of "dir"
to "nginx"</text>
<code>
```

Reviewer Prompt

```
<code>echo $(ls -l /bin/bash)</code>
<text>Print info of "bash"</text>
... 2 more demonstration examples
<code>chown nginx:nginx dir</code>
<text>Change the owner of "dir"
to "nginx"</text>
```

$$\log p(x|y)p(y|x) = \log p(x|y) + \log p(y|x)$$

(Coder-Reviewer Reranking)

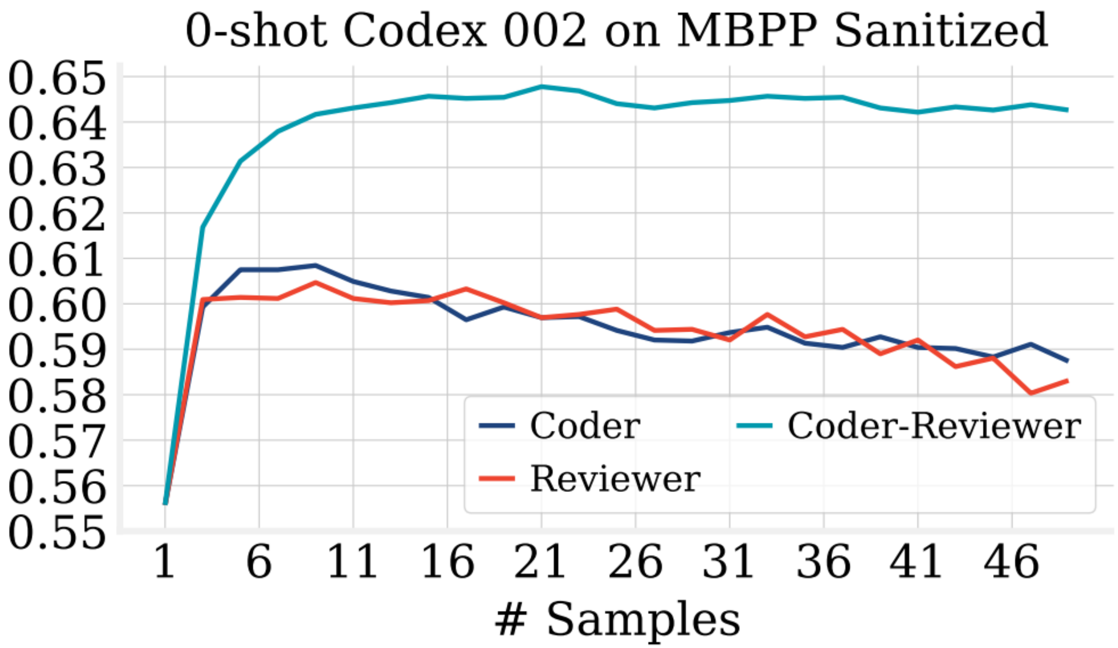
$$\mathrm{argmax}_y \ \log \frac{p(y,x)}{p(x)p(y)^\alpha}$$
$$= \mathrm{argmax}_y \ (1-\alpha)\log p(y|x) + \alpha \log p(x|y)$$

$$\frac{\log p(x|y)}{|x|} + \frac{\log p(y|x)}{|y|}$$

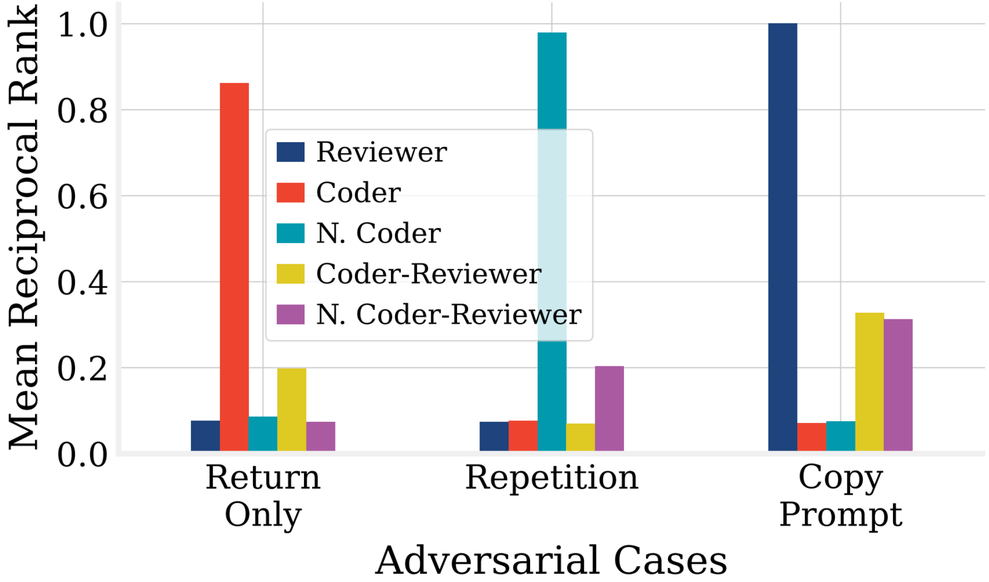
(Normalized Coder-Reviewer Reranking)

[Coder-Reviewer reranking, Zhang et al. 2022]

Mutual Information Helps Avoid Biases



	HumanEval	MBPP-S
Random	$48.0_{-1.2}$	$58.1_{-0.7}$
Coder	$38.1_{-7.1}$	$55.3_{-4.5}$
N.Coder	$59.7_{-0.5}$	$60.0_{-0.5}$
Reviewer	$57.7_{-3.5}$	$55.8_{-3.7}$
Coder-Reviewer	$53.2_{-3.5}$	$60.5_{-3.9}$
Norm. Coder-Reviewer	$\mathbf{61.5}_{-1.0}$	$\mathbf{60.8}_{-0.7}$



[Coder-Reviewer reranking, Zhang et al. 2022]

Mode Splitting

Draw on the board

Minimum Bayes Risk (MBR)

- Assume your model has some error (loss); choose an output that minimizes your expected error (*risk*).
- Or equivalently, assume your model probability is spread over good stuff; choose something close to high probability model outputs.

$$\begin{aligned}\hat{y} &= \underset{y' \in \mathscr{Y}}{\operatorname{argmin}}\, R(y') \\ &= \underset{y' \in \mathscr{Y}}{\operatorname{argmin}}\, \mathbb{E}_{y|x}[L(y, y')] \\ &= \underset{y' \in \mathscr{Y}}{\operatorname{argmin}} \sum_{y \in \mathscr{Y}} L(y, y')p(y|x)\end{aligned}$$

$$\begin{aligned}R(y') &\approx \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} L(y, y') \\ &= -\frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} G(y, y')\end{aligned}$$

$$\hat{y} = \underset{y' \in \mathcal{Y}}{\operatorname{argmax}} \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}_e} G(y, y')$$

It's MBR All The Way Down, Bertsch*, Xie* et al. 2023

MBR with Execution

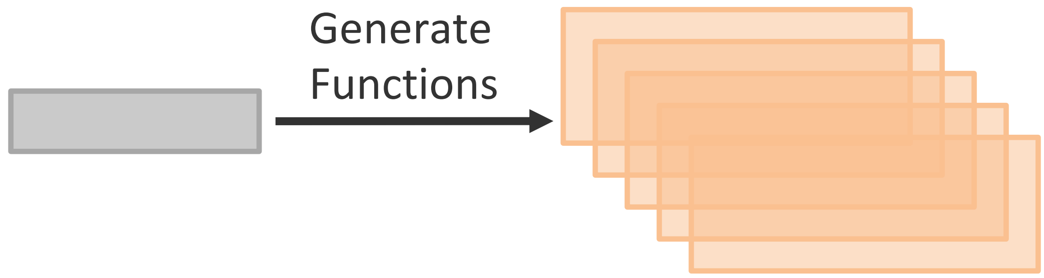
Description:

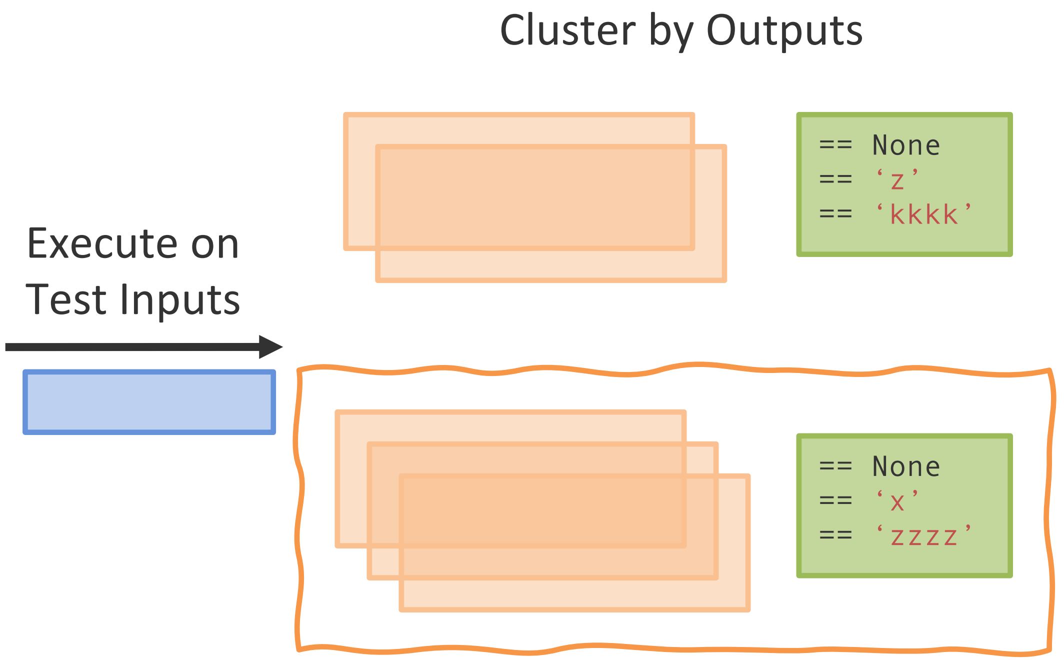
```python
def longest(strings: List[str]) -> Optional[str]:
  """ Out of list of strings, return the longest one.
  Return the first one in case of multiple strings of
  the same length. Return None if the list is empty."""
```

Test Inputs:

```python
longest([]) == ___
longest(['x', 'y', 'z']) == ___
longest(['x', 'yyy', 'zzzz', 'www', 'kkkk', 'abc']) == ___
```

Minimum Bayes Risk with Execution:



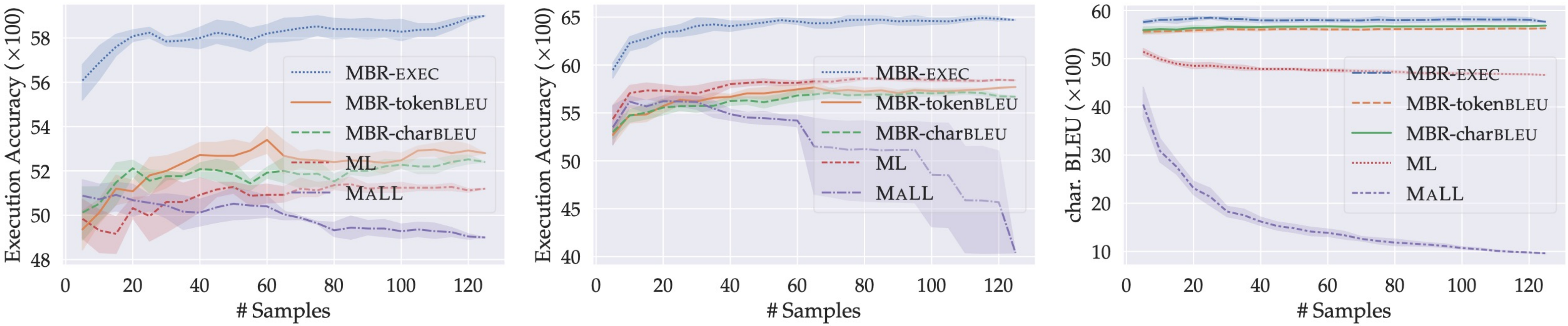


Here, the MBR gain function is 1{functions have the same outputs for these inputs}

Shi et al. 2022

MBR with Execution

Method	MBPP	Spider	NL2Bash
Greedy (3-shot)	47.3 ± 2.5	50.8 ± 2.6	52.8 ± 2.9
Sample (3-shot)	47.7 ± 1.5	48.5 ± 2.6	53.0 ± 2.9
MBR-EXEC	$\mathbf{58.2} \pm 0.3$	$\mathbf{63.6} \pm 0.8$	$\mathbf{58.5} \pm 0.3$



(a) MBPP (b) Spider (c) NL2Bash

Shi et al. 2022

AlphaCode: Setting

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).

Example Input

```
4
ababa
ba
ababa
bb
aaa
aaaa
aababa
ababa
```

Example Output

```
YES
NO
NO
YES
```

Explanation

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.

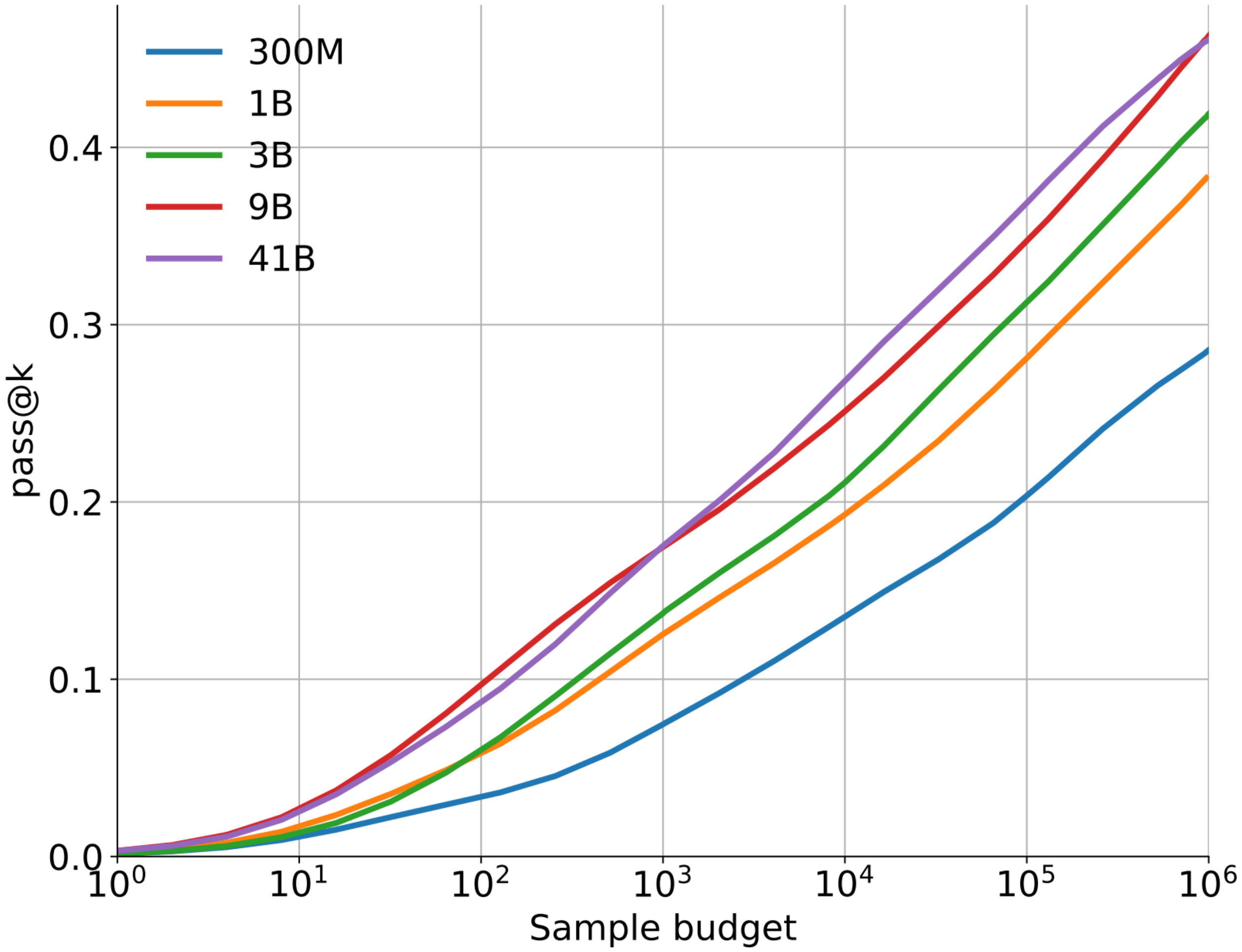
```python
t=int(input())
for i in range(t):
    s=input()
    t=input()
    a=[]
    b=[]
    for j in s:
        a.append(j)
    for j in t:
        b.append(j)
    a.reverse()
    b.reverse()
    c=[]
    while len(b)!=0 and len(a)!=0:
        if a[0]==b[0]:
            c.append(b.pop(0))
        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

AlphaCode: Approach

- Training:
  - Pre-train encoder-decoder LMs (300M – 41B parameters) on GitHub code
  - Fine-tune on 13K problems scraped from Codeforces contest site

- Inference:
  - Sample huge number of candidate solutions (~1M) for each problem
  - Filter the candidates on public test cases, then apply MBR clustering with model-generated test inputs to choose 10 output solutions

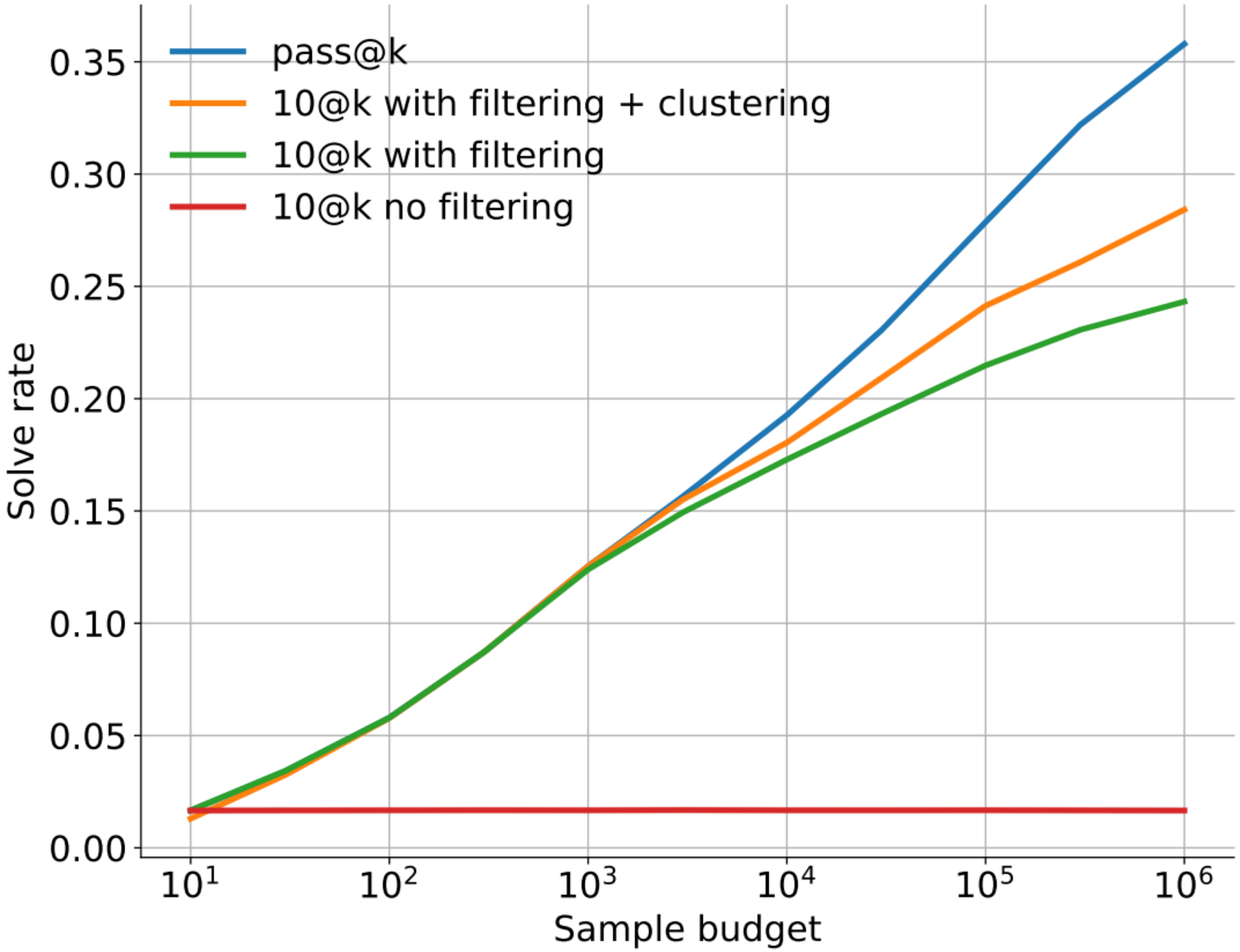
AlphaCode: Google-Scale Sampling



AlphaCode: Results

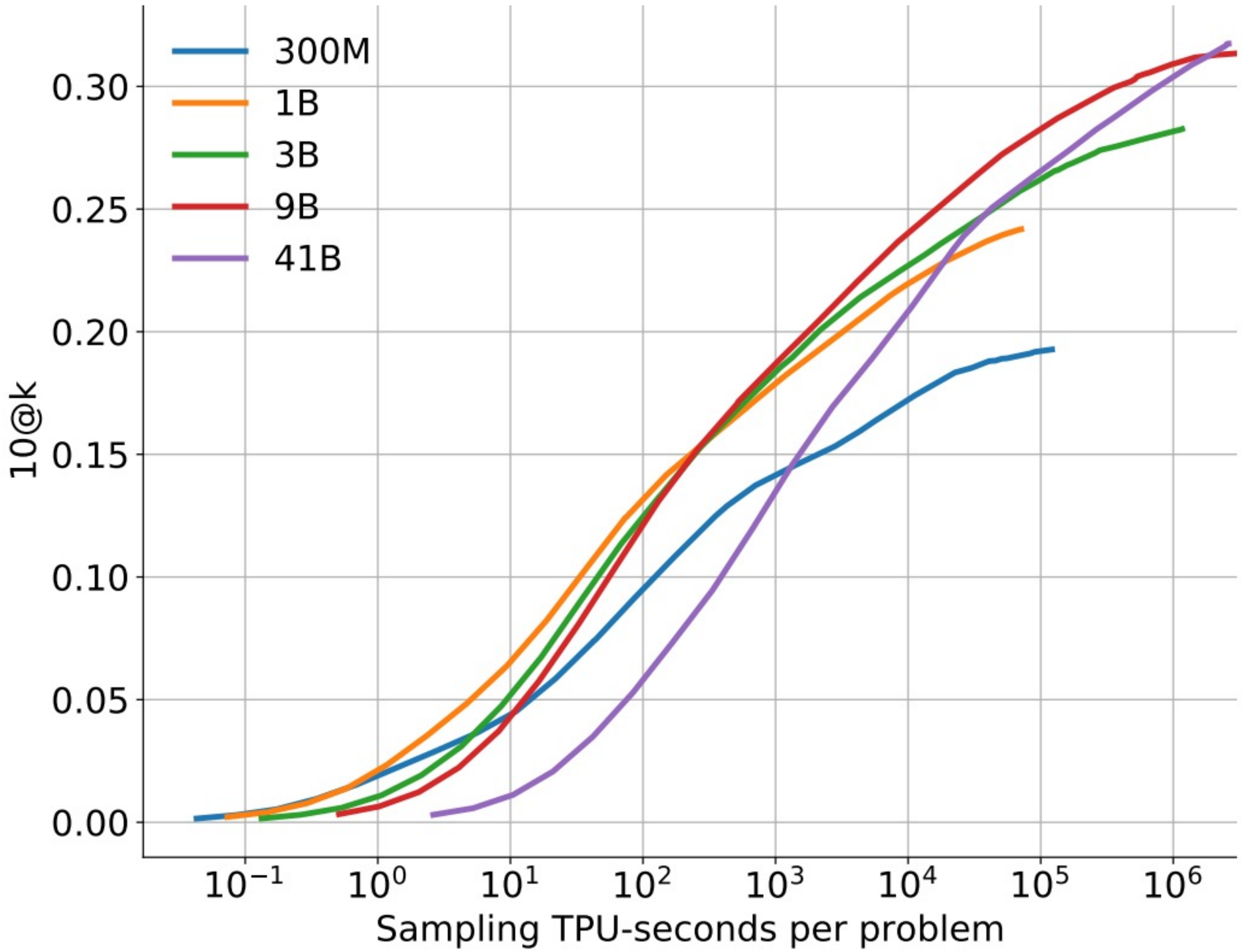
- Filtering generated solutions using public test cases is necessary

- MBR clustering gives further benefits



AlphaCode: Results

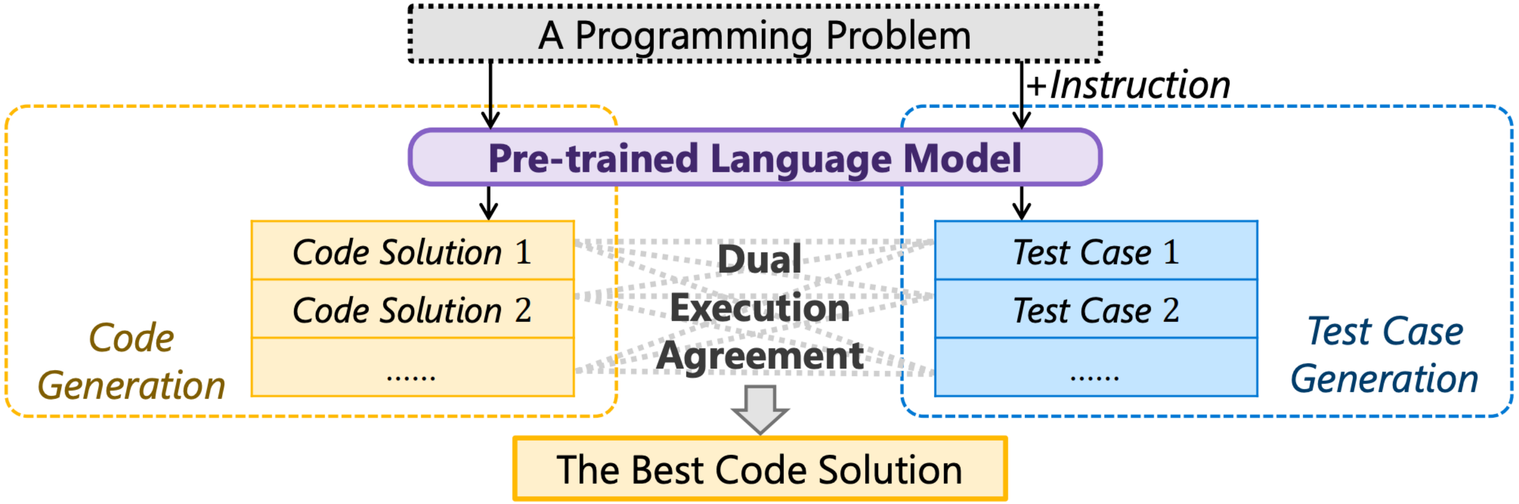
- Sampling with big models is expensive!



(b) Solve Rate vs. Sampling Compute

CodeT: Overview

- **Intuition**: when generating test cases, some test cases may be higher quality than others.
  - We can evaluate test case quality using generated functions, and vice versa.
- Sample many functions, and many test cases, and look for *consensus sets* of (function, test) pairs.



CodeT: Method

```python
def longest(strings: List[str]) -> Optional[str]:
  """ Out of list of strings, return the longest one.
  Return the first one in case of multiple strings of
  the same length. Return None if the list is empty."""
```

```python
    if len(strings) == 0:
        return None
    return max(strings, key=lambda s: len(s))
```

```python
longest([]) == None
longest(['x', 'y', 'z']) == ‘z’
longest(['x', 'yyy', 'zzzz', 'www', 'kkkk', 'abc']) == ‘kkkk’
```

	a	b	c	d	e	f	g	h
A					✓			✓
B				✓		✓	✓	
C	✓		✓	✓				
D				✓		✓	✓	
E				✓		✓	✓	
F					✓			
G	✓		✓	✓				

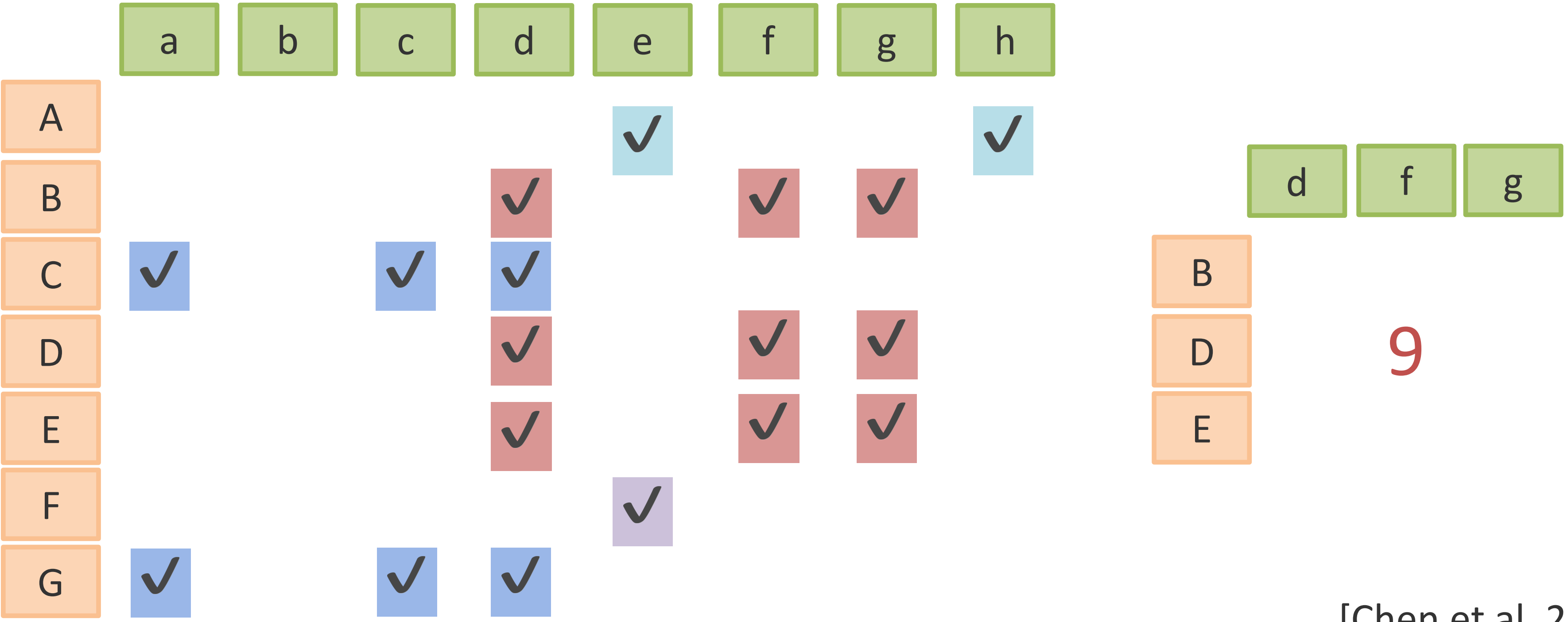
[Chen et al. 2022]

CodeT: Method

```python
def longest(strings: List[str]) -> Optional[str]:
  """ Out of list of strings, return the longest one.
  Return the first one in case of multiple strings of
  the same length. Return None if the list is empty."""
```

```python
    if len(strings) == 0:
        return None
    return max(strings, key=lambda s: len(s))
```

```python
longest([]) == None
longest(['x', 'y', 'z']) == ‘z’
longest(['x', 'yyy', 'zzzz', 'www', 'kkkk', 'abc']) == ‘kkkk’
```



[Chen et al. 2022]

CodeT: Method

- Like MBR-Exec / AlphaCode-C, except…
  - It generates test cases (inputs **and** outputs) too
  - It ranks clustered functions by the number of functions **times** the number of passed test cases
- When tests * solutions is large, use sampling (RANSAC algorithm)
- If *k* solutions are wanted (e.g. pass@k), choose *k* sets with one function from each set

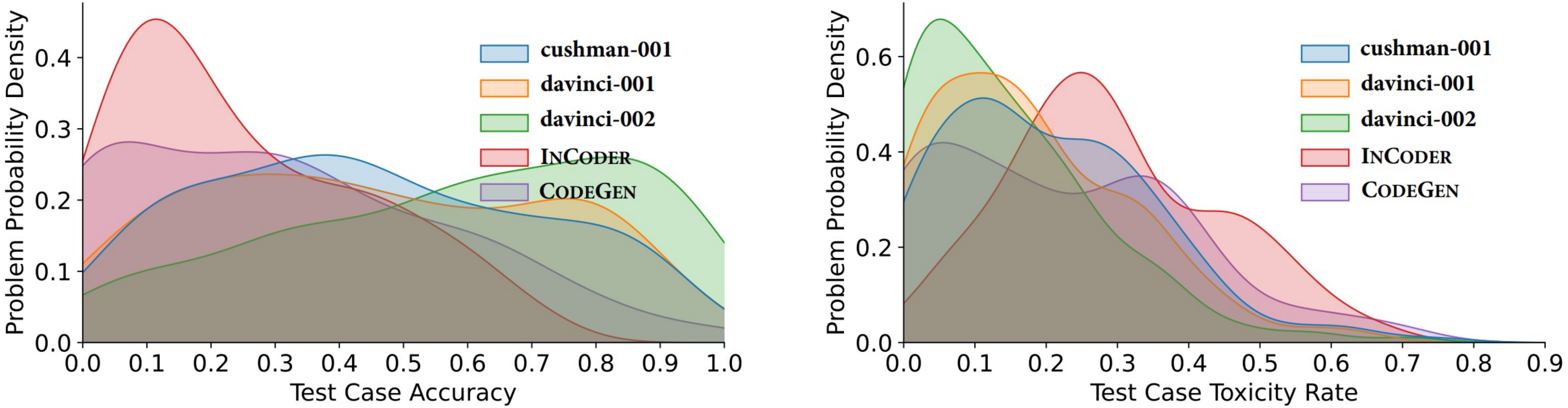
CodeT: Results

- Large improvements in pass@k scores over baseline sampling and MBR / AlphaCode-like clustering.

Methods	Baseline			AlphaCode-C			CodeT		
k	1	10	100	1	2	10	1	2	10
HumanEval									
code-cushman-001	33.5	54.3	77.4	39.6	46.4	63.8	44.5 11.0	50.1	65.7 11.4
code-davinci-001	39.0	60.6	84.1	41.6	50.7	75.6	50.2 11.2	58.9	75.8 15.2
code-davinci-002	47.0	74.9	92.1	55.1	64.1	84.4	65.8 18.8	75.1	86.6 11.7
InCoder-6B	16.4 15.2	28.3 27.8	47.5 47.0	17.7	23.8	34.8	20.6 4.2	27.6	37.1 8.8
CodeGen-Mono-16B	29.7 29.3	50.3 49.9	73.7 75.0	27.3	38.5	64.4	36.7 7.0	44.7	59.3 9.0
MBPP									
code-cushman-001	45.9	66.9	79.9	51.5	59.0	73.3	55.4 9.5	61.7	72.7 5.8
code-davinci-001	51.8	72.8	84.1	56.2	64.7	78.8	61.9 10.1	69.1	79.3 6.5
code-davinci-002	58.1	76.7	84.5	62.0	70.7	79.9	67.7 9.6	74.6	81.5 4.8
InCoder-6B	21.3 19.4	46.5	66.2	26.7	35.3	56.2	34.4 13.1	43.9	58.2 11.7
CodeGen-Mono-16B	42.4	65.8	79.1	41.0	55.9	73.6	49.5 7.1	56.6	68.5 2.7

CodeT: Results

- How high quality are these test cases, really?



HumanEval test case accuracy and “toxicity” (test cases that any generated program can pass but the ground truth cannot).

CodeT: Results

- How high quality are these test cases, really?

Methods	Code Coverage	
	Statement	Branch
code-cushman-001	95.3	98.1
code-davinci-001	94.9	97.6
code-davinci-002	95.7	98.5
INCODER	94.0	96.3
CODEGEN	78.2	78.6

Table 11: The Code Coverage (%) statistics of test cases generated by five models on the HumanEval benchmark.

Coverage scores, test accuracy, and solution accuracy may be very different for a given model.

CodeT: Results

- How high quality are these test cases, really?

Methods	**Code Solution Only** f'			**Test Case Only** f''		
k	1	2	10	1	2	10
code-cushman-001	$41.2^{-3.3}_{+7.7}$	$49.2^{-0.9}$	$61.9^{-3.8}_{+7.6}$	$29.9^{-14.6}_{-3.6}$	$36.6^{-13.5}$	$59.5^{-6.2}_{+5.2}$
code-davinci-001	$44.4^{-5.8}_{+5.4}$	$54.7^{-4.2}$	$69.0^{-6.8}_{+8.4}$	$35.0^{-15.2}_{-4.0}$	$46.0^{-12.9}$	$70.2^{-5.6}_{+9.6}$
code-davinci-002	$55.9^{-9.9}_{+8.9}$	$67.0^{-8.1}$	$82.7^{-3.9}_{+7.8}$	$58.4^{-7.4}_{+11.4}$	$65.1^{-10.0}$	$86.1^{-0.5}_{+11.2}$

Even though tests are noisy, consensus ranking still helps substantially.

MBR also works with a learned similarity function

- Chen et al. 2023 prompt an LLM to choose the consensus output

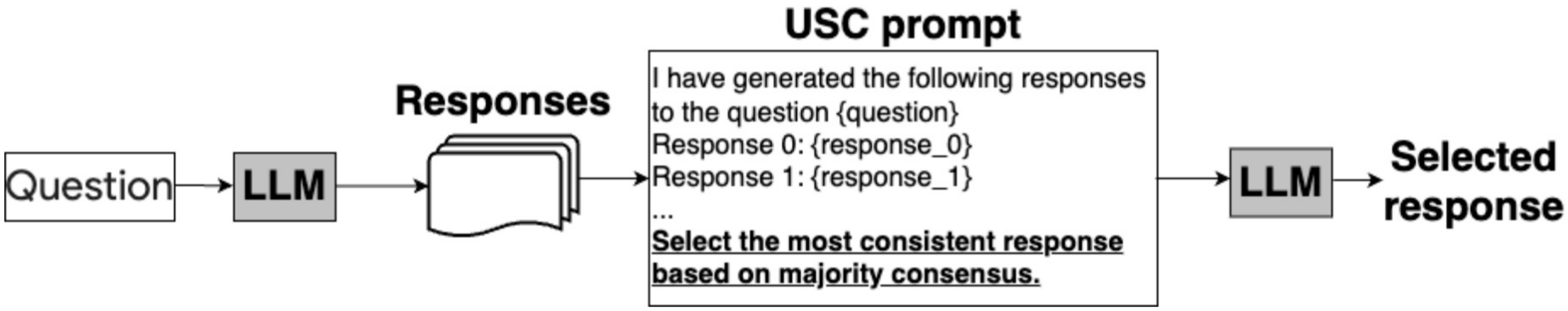


Table 2: Accuracy on code generation benchmarks with `gpt-3.5-turbo`.

Dataset	Approach	Execution Accuracy	Valid Efficiency Score
BIRD-SQL	Greedy decoding	42.4	44.4
	Random	41.9	44.0
	SC-Exec	**45.6**	48.1
	USC	45.5	**48.8**
ARCADE	Greedy decoding	26.0	N/A
	Random	26.8	
	SC-Exec (strict match)	29.8	
	SC-Exec (fuzzy match)	**30.3**	
	USC	30.1	

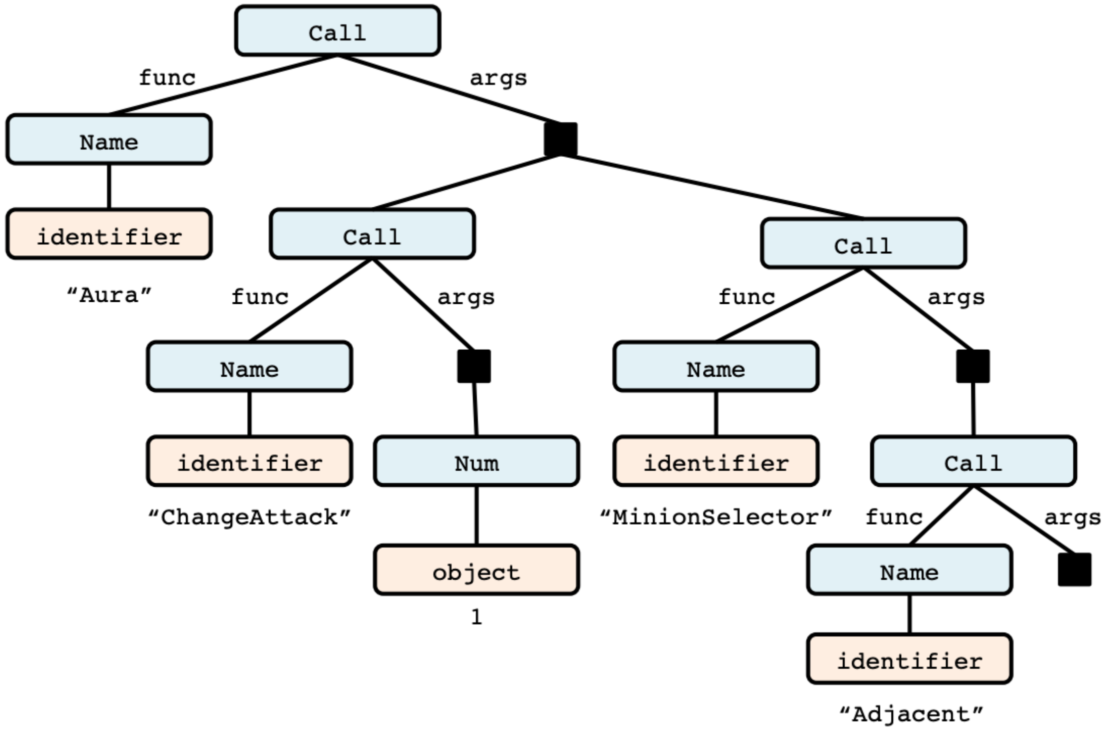
Incorporating Syntax

Abstract Syntax Networks

- Approach 1: Constrain the model


```
name: [
  'D', 'i', 'r', 'e', ' ',
  'W', 'o', 'l', 'f', ' ',
  'A', 'l', 'p', 'h', 'a']
cost: ['2']
type: ['Minion']
rarity: ['Common']
race: ['Beast']
class: ['Neutral']
description: [
  'Adjacent', 'minions', 'have',
  '+', '1', 'Attack', '.']
health: ['2']
attack: ['2']
durability: ['-1']
```

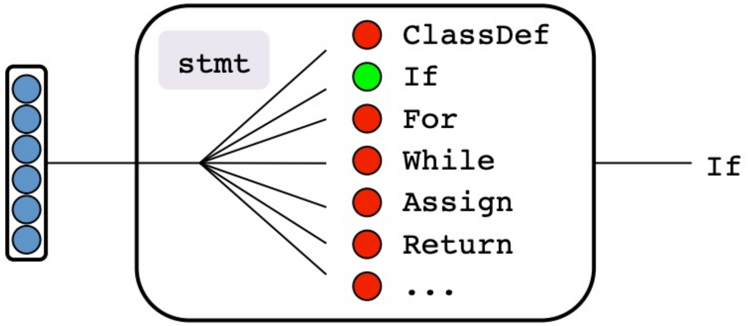
```python
class DireWolfAlpha(MinionCard):
  def __init__(self):
    super().__init__(
      "Dire Wolf Alpha", 2, CHARACTER_CLASS.ALL,
      CARD_RARITY.COMMON, minion_type=MINION_TYPE.BEAST)
  def create_minion(self, player):
    return Minion(2, 2, auras=[
      Aura(ChangeAttack(1), MinionSelector(Adjacent()))
    ])
```



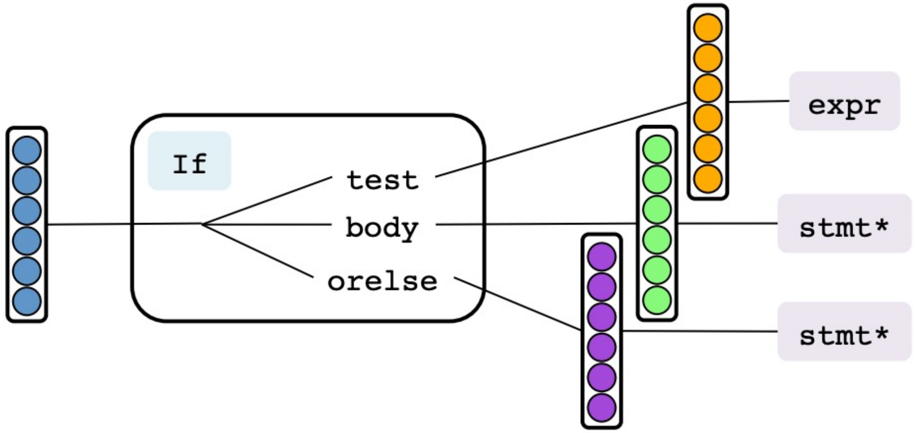
(b) Excerpt from the same AST, corresponding to the code snippet `Aura(ChangeAttack(1),MinionSelector(Adjacent()))`.

Rabinovich, Stern, and Klein 2017

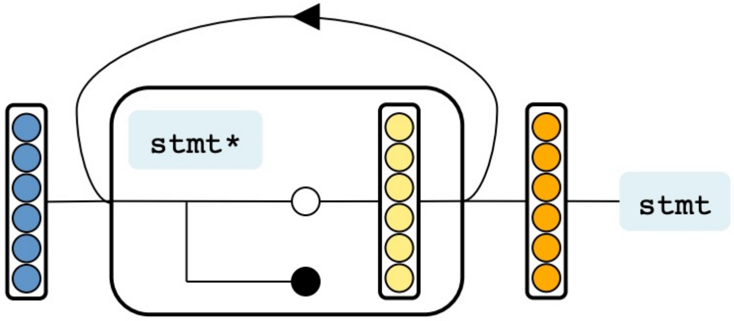
Abstract Syntax Networks



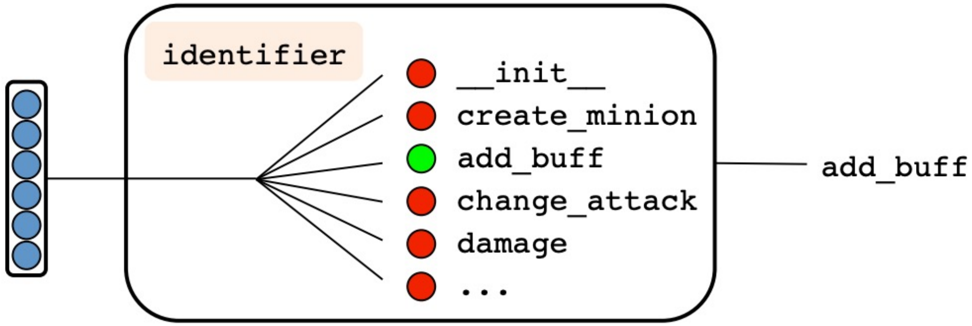
(a) A composite type module choosing a constructor for the corresponding type.



(b) A constructor module computing updated vertical LSTM states.



(c) A constructor field module (`sequential` cardinality) generating children to populate the field. At each step, the module decides whether to generate a child and continue (white circle) or stop (black circle).

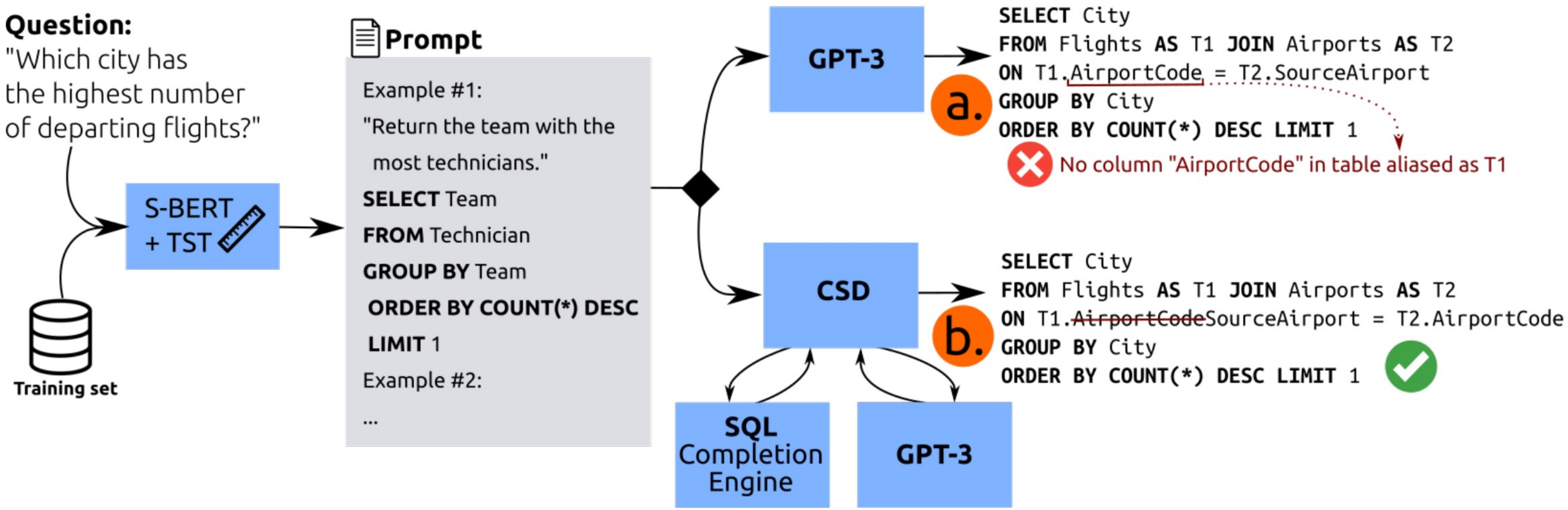


(d) A primitive type module choosing a value from a closed list.

Rabinovich, Stern, and Klein 2017

Constrained Decoding

- Approach 2: Constrain the model's probability distributions
- *Completion engines* give valid completions of any prefixes, using language semantics and user's context (e.g. a database)



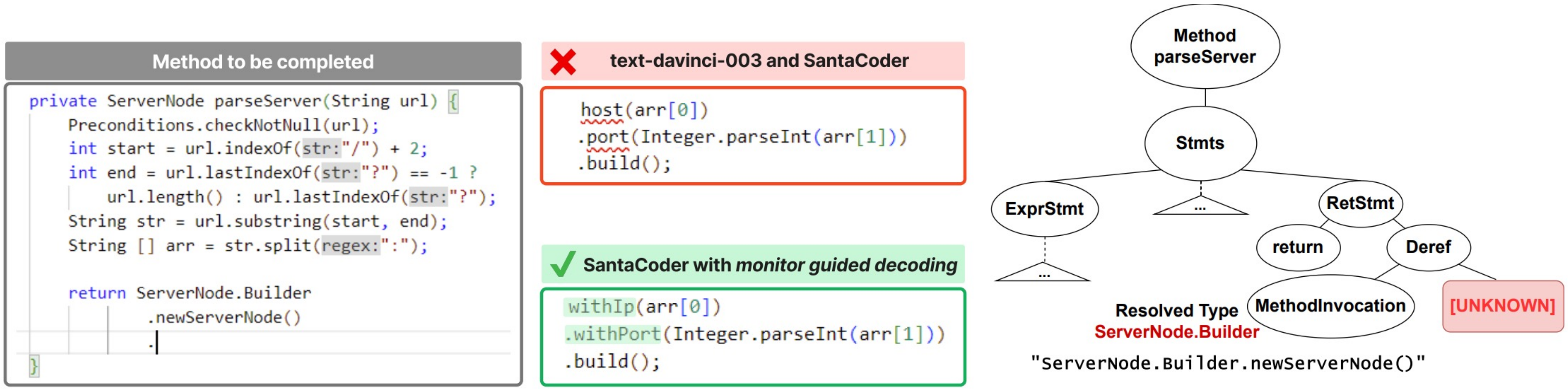
Synchromesh, Poesia et al. 2022.
See also Shin et al. 2021, Shin and Van Durme 2022

Constrained Decoding

Model	SQL			Vega-Lite			SMCalFlow		
	Exec.	**Valid**	**Dist.**	**Acc.**	**Valid**	**Dist.**	**Acc.**	**Valid**	**Dist.**
Andreas et al. (2020)	-	-	-	-	-	-	$72\%^{(S)}$	-	-
Srinivasan et al. (2021)	-	-	-	$64\%^{(S)}$	-	-	-	-	-
Rubin & Berant (2021)	$71\%^{(S)}$	-	-	-	-	-	-	-	-
Scholak et al. (2021)	$79\%^{(S)}$	98%	-	-	-	-	-	-	-
GPT-3 13B	16%	43%	0.42	14%	55%	0.51	38%	76%	0.43
" + CSD	20%	66%	0.44	17%	100%	0.48	40%	95%	0.40
" + TST	14%	48%	0.42	-	-	-	60%	88%	0.22
" + CSD + TST	19%	72%	0.43	-	-	-	63%	98%	0.17
GPT-3 175B	28%	49%	0.36	20%	67%	0.36	44%	77%	0.41
" + CSD	35%	73%	0.36	25%	100%	0.32	45%	97%	0.37
" + TST	31%	56%	0.35	-	-	-	60%	88%	0.24
" + CSD + TST	37%	76%	0.34	-	-	-	66%	97%	0.18
Codex 175B	56%	73%	0.25	39%	87%	0.24	45%	79%	0.37
" + CSD	61%	85%	0.23	40%	99%	0.23	46%	97%	0.33
" + TST	60%	81%	0.23	-	-	-	63%	90%	0.21
" + CSD + TST	64%	85%	0.23	-	-	-	63%	99%	0.19

Synchromesh, Poesia et al. 2022.
See also Shin et al. 2021, Shin and Van Durme 2022

Constrained Decoding



(a) Example where text-davinci-003 and SantaCoder generate wrong identifiers, but SantaCoder with MGD generates correct identifiers.

(b) Annotated partial AST for the code to the left.

Monitor-guided Decoding, Agrawal et al. 2023

Grammar Prompting

LLM Prompt

You are an expert programmer, and you need to write a program for the given natural language query. First, you should write a grammar that contains all the necessary BNF rules. Then, you should write programs that conform to your predicted rules.

(optional) G: [BEGIN RULES] ... [END RULES]

$\boldsymbol{x}^{(1)}$: find the meeting on Wednesday with Bob and Carol

$G[\boldsymbol{y}^{(1)}]$:

```
event       ::= "QueryEvent(" constraint ")"
constraint  ::= "(&" constraint constraint ")"
              | "(start_?" date ")"
              | "(attendee_?" attendee attendee ")"
date        ::= "Wednesday"
attendee    ::= "Bob" | "Carol"
```

$\boldsymbol{y}^{(1)}$: `QueryEvent(& (start_? Wednesday)(attendee_? Bob Carol))`

...

$\boldsymbol{x}$: Add meeting with Jean's manager on Wednesday at 3PM

LLM Output

$\widehat{G}$:

```
event       ::= "CreateEvent(" constraint ")"
constraint  ::= "(&" constraint constraint ")"
              | "(start_?" date time ")"
              | "(attendee_?" attendee ")"
date        ::= "Wednesday"
time        ::= "NumberPM(3)"
attendee    ::= "FindManager(" attendee ")" | "Jean"
```

$\widehat{\boldsymbol{y}}$: `CreateEvent(& (start_? Wednesday NumberPM(3))(attendee_? FindManager(Jean)))`

Wang et al. 2023