Inference Algorithms

Daniel Fried
11-891: Neural Code Generation
https://cmu-codegen.github.io/s2024/

With slides from Graham Neubig
Single-Sample Generation
Autoregressive Code Modeling

Jupyter notebook demo
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

\[
\text{while } x_{j-1} \neq "</s>": \quad x_j = \text{argmax } P(x_j | x_1, ..., x_{j-1})
\]

- 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: 

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

Completion 1: 
```
def count_words(filename: str) -> Counter[str, int]:
    pass
```

Completion 2: 
```
words = Counter()
with open(filename, 'r') as f:
    for line in f.readlines():
        words.update(line.split())
return words
```

Completion 3: 
```
word_counts = Counter()
with open(filename, 'r') as f:
    for line in f.readlines():
        word_counts.update(line.split())
return word_counts
```

Completion 4: 
```
return Counter(tok for line in
    open(filename, 'r').readlines() for
    tok in line.split())
```

Probabilities from DeepSeekCoder-6.7B

- log $p(c_1|\text{prompt}) = -4.69$
- log $p(c_2|\text{prompt}) = -18.13$
- log $p(c_3|\text{prompt}) = -19.19$
- log $p(c_4|\text{prompt}) = -23.31$
Sampling

- Randomly generate words one-by-one. (aka "ancestral sampling")
  
  \[
  \text{while } x_{j-1} \neq "<s>": \\
  x_j \sim P(x_j | x_1, ..., x_{j-1})
  \]

- 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.

\[ P(x_6 \mid \text{“The capital of Pennsylvania is”}) \]

<table>
<thead>
<tr>
<th>City</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harrisburg</td>
<td>34.3%</td>
</tr>
<tr>
<td>Philadelphia</td>
<td>31.1%</td>
</tr>
<tr>
<td>Pittsburgh</td>
<td>12.9%</td>
</tr>
<tr>
<td>Easton</td>
<td>2.2%</td>
</tr>
<tr>
<td>Lancaster</td>
<td>1.8%</td>
</tr>
<tr>
<td>Allentown</td>
<td>1.6%</td>
</tr>
<tr>
<td>Washington</td>
<td>1.5%</td>
</tr>
</tbody>
</table>

- **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

\[ p(y) = \text{softmax} \left( \frac{\text{logit}}{\tau} \right) \propto \exp \left( \frac{\text{logit}}{\tau} \right) \]
Beware of Tokenization!

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

Example from InCoder, Fried et al. 2023
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.

import pandas as pd
import matplotlib.pyplot as plt

# Exercise 1
df = pd.read_csv('scores.csv')

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

# 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']]

# 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 tem-
is used in Reynolds and McDonell [2021] but is not quantitatively evaluated.

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

Sample Ranking Heuristics

- Oracle
- Docstring backtranslation
- Sum logp
- Mean logp
- Random

Pass rate vs. Number of samples (k)

[Codex paper, Chen et al. 2021]
But Beware Model Biases

[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>

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)

\[
\arg\max_y \log \frac{p(y, x)}{p(x)p(y)} \alpha
\]
\[
= \arg\max_y (1 - \alpha) \log p(y|x) + \alpha \log p(x|y)
\]

\[
\frac{\log p(x|y)}{|x|} + \frac{\log p(y|x)}{|y|}
\]
(Normalized Coder-Reviewer Reranking)

[Coder-Reviewer reranking, Zhang et al. 2022]
Mutual Information Helps Avoid Biases

0-shot Codex 002 on MBPP Sanitized

![Graph showing mutual information over # samples]

<table>
<thead>
<tr>
<th></th>
<th>HumanEval</th>
<th>MBPP-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>48.0 −1.2</td>
<td>58.1 −0.7</td>
</tr>
<tr>
<td>Coder</td>
<td>38.1 −7.1</td>
<td>55.3 −4.5</td>
</tr>
<tr>
<td>N.Coder</td>
<td>59.7 −0.5</td>
<td>60.0 −0.5</td>
</tr>
<tr>
<td>Reviewer</td>
<td>57.7 −3.5</td>
<td>55.8 −3.7</td>
</tr>
<tr>
<td>Coder-Reviewer</td>
<td>53.2 −3.5</td>
<td>60.5 −3.9</td>
</tr>
<tr>
<td>Norm. Coder-Reviewer</td>
<td>61.5 −1.0</td>
<td>60.8 −0.7</td>
</tr>
</tbody>
</table>

Mean Reciprocals Rank

![Bar chart showing mean reciprocals rank by adversarial cases]

[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.

\[
\hat{y} = \arg\min_{y' \in \mathcal{Y}} R(y') \\
= \arg\min_{y' \in \mathcal{Y}} \mathbb{E}_{y|x}[L(y, y')] \\
= \arg\min_{y' \in \mathcal{Y}} \sum_{y \in \mathcal{Y}} L(y, y') p(y|x)
\]

\[
R(y') \approx \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} L(y, y') \\
= -\frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} G(y, y')
\]

\[
\hat{y} = \arg\max_{y' \in \mathcal{Y}} \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}_e} G(y, y')
\]

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

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

Test Inputs:

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

<table>
<thead>
<tr>
<th>Method</th>
<th>MBPP</th>
<th>Spider</th>
<th>NL2Bash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy (3-shot)</td>
<td>47.3 ± 2.5</td>
<td>50.8 ± 2.6</td>
<td>52.8 ± 2.9</td>
</tr>
<tr>
<td>Sample (3-shot)</td>
<td>47.7 ± 1.5</td>
<td>48.5 ± 2.6</td>
<td>53.0 ± 2.9</td>
</tr>
<tr>
<td><strong>MBR-EXEC</strong></td>
<td><strong>58.2 ± 0.3</strong></td>
<td><strong>63.6 ± 0.8</strong></td>
<td><strong>58.5 ± 0.3</strong></td>
</tr>
</tbody>
</table>

(a) MBPP  
(b) Spider  
(c) NL2Bash
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 “abcbcd” 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 “abaca” 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 $q$ ($1 \leq q \leq 10^5$) the number of test cases. The first line of each test case contains the string $s$ ($1 \leq |s| \leq 10^5$). Each character of $s$ is a lowercase English letter.
The second line of each test case contains the string $t$ ($1 \leq |t| \leq 10^5$). 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
aaaaaaaa
ababa
```

Example Output

```
YES
NO
NO
YES
```

Explanations
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 “ababa”.

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

![Graph showing the relationship between sample budget and pass@k for different data sizes (300M, 1B, 3B, 9B, 41B). The x-axis represents the sample budget in logarithmic scale from $10^0$ to $10^6$, and the y-axis represents pass@k from 0.0 to 0.4. The graph shows how the pass@k increases with the sample budget for each data size.]
Filtering generated solutions using public test cases is necessary

MBR clustering gives further benefits
Sampling with big models is expensive!
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.
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."""

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

if __name__ == '__main__':
    assert longest([]) == None
    assert longest(['x', 'y', 'z']) == 'z'
    assert longest(['x', 'yyy', 'zzzz', 'www', 'kkkk', 'abc']) == 'kkkk'

[Chen et al. 2022]
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."""

    if len(strings) == 0:
        return None

    return max(strings, key=lambda s: len(s))

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.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Baseline</th>
<th>AlphaCode-C</th>
<th>CODET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k=1 10 100</td>
<td>k=1 2 10</td>
<td>k=1 2 10</td>
</tr>
<tr>
<td><strong>HumanEval</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>code-cushman-001</td>
<td>33.5 54.3</td>
<td>39.6 46.4</td>
<td>44.5 11.0</td>
</tr>
<tr>
<td>code-davinci-001</td>
<td>39.0 60.6</td>
<td>41.6 50.7</td>
<td>50.2 12.2</td>
</tr>
<tr>
<td>code-davinci-002</td>
<td>47.0 74.9</td>
<td>55.1 64.1</td>
<td>65.8 18.8</td>
</tr>
<tr>
<td>INCODER-6B</td>
<td>16.4 15.2</td>
<td>17.7 23.8</td>
<td>20.6 4.2</td>
</tr>
<tr>
<td>CODEGEN-MONO-16B</td>
<td>29.7 29.3</td>
<td>27.3 38.5</td>
<td>36.7 7.0</td>
</tr>
<tr>
<td><strong>MBPP</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>code-cushman-001</td>
<td>45.9 66.9</td>
<td>51.5 59.0</td>
<td>55.4 9.5</td>
</tr>
<tr>
<td>code-davinci-001</td>
<td>51.8 72.8</td>
<td>56.2 64.7</td>
<td>61.9 10.1</td>
</tr>
<tr>
<td>code-davinci-002</td>
<td>58.1 76.7</td>
<td>62.0 70.7</td>
<td>67.7 9.6</td>
</tr>
<tr>
<td>INCODER-6B</td>
<td>21.3 19.4</td>
<td>26.7 35.3</td>
<td>34.4 13.1</td>
</tr>
<tr>
<td>CODEGEN-MONO-16B</td>
<td>42.4 63.8</td>
<td>41.0 55.9</td>
<td>49.5 7.1</td>
</tr>
</tbody>
</table>
HumanEval test case accuracy and “toxicity” (test cases that any generated program can pass but the ground truth cannot).
How high quality are these test cases, really?

<table>
<thead>
<tr>
<th>Methods</th>
<th>Code Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Statement</td>
</tr>
<tr>
<td>code-cushman-001</td>
<td>95.3</td>
</tr>
<tr>
<td>code-davinci-001</td>
<td>94.9</td>
</tr>
<tr>
<td>code-davinci-002</td>
<td>95.7</td>
</tr>
<tr>
<td>INCODER</td>
<td>94.0</td>
</tr>
<tr>
<td>CODEGEN</td>
<td>78.2</td>
</tr>
</tbody>
</table>

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.
How high quality are these test cases, really?

<table>
<thead>
<tr>
<th>Methods</th>
<th>Code Solution Only $f'$</th>
<th>Test Case Only $f''$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k$=1</td>
<td>$k$=2</td>
</tr>
<tr>
<td>code-cushman-001</td>
<td>41.2$^\pm$7.7</td>
<td>49.2$^\pm$0.9</td>
</tr>
<tr>
<td>code-davinci-001</td>
<td>44.4$^\pm$5.4</td>
<td>54.7$^\pm$4.2</td>
</tr>
<tr>
<td>code-davinci-002</td>
<td>55.9$^\pm$9.9</td>
<td>67.0$^\pm$8.1</td>
</tr>
</tbody>
</table>

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

![Diagram showing MBR process]

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Approach</th>
<th>Execution Accuracy</th>
<th>Valid Efficiency Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIRD-SQL</td>
<td>Greedy decoding</td>
<td>42.4</td>
<td>44.4</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>41.9</td>
<td>44.0</td>
</tr>
<tr>
<td></td>
<td>SC-Exec</td>
<td><strong>45.6</strong></td>
<td>48.1</td>
</tr>
<tr>
<td></td>
<td>USC</td>
<td>45.5</td>
<td><strong>48.8</strong></td>
</tr>
<tr>
<td>ARCADE</td>
<td>Greedy decoding</td>
<td>26.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>26.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SC-Exec (strict match)</td>
<td>29.8</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>SC-Exec (fuzzy match)</td>
<td><strong>30.3</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>USC</td>
<td>30.1</td>
<td></td>
</tr>
</tbody>
</table>
Incorporating Syntax
Abstract Syntax Networks

- Approach 1: Constrain the model

```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()))`.
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.
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

<table>
<thead>
<tr>
<th>Model</th>
<th>SQL</th>
<th>Vega-Lite</th>
<th>SMCalFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andreas et al. (2020)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Srinivasan et al. (2021)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Rubin &amp; Berant (2021)</td>
<td>71%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Scholak et al. (2021)</td>
<td>79%</td>
<td>98%</td>
<td>-</td>
</tr>
<tr>
<td>GPT-3 13B</td>
<td>16%</td>
<td>43%</td>
<td>0.42</td>
</tr>
<tr>
<td>” + CSD</td>
<td>20%</td>
<td>66%</td>
<td>0.44</td>
</tr>
<tr>
<td>” + TST</td>
<td>14%</td>
<td>48%</td>
<td>0.42</td>
</tr>
<tr>
<td>” + CSD + TST</td>
<td>19%</td>
<td>72%</td>
<td>0.43</td>
</tr>
<tr>
<td>GPT-3 175B</td>
<td>28%</td>
<td>49%</td>
<td>0.36</td>
</tr>
<tr>
<td>” + CSD</td>
<td>35%</td>
<td>73%</td>
<td>0.36</td>
</tr>
<tr>
<td>” + TST</td>
<td>31%</td>
<td>56%</td>
<td>0.35</td>
</tr>
<tr>
<td>” + CSD + TST</td>
<td>37%</td>
<td>76%</td>
<td>0.34</td>
</tr>
<tr>
<td>Codex 175B</td>
<td>56%</td>
<td>73%</td>
<td>0.25</td>
</tr>
<tr>
<td>” + CSD</td>
<td>61%</td>
<td>85%</td>
<td>0.23</td>
</tr>
<tr>
<td>” + TST</td>
<td>60%</td>
<td>81%</td>
<td>0.23</td>
</tr>
<tr>
<td>” + CSD + TST</td>
<td>64%</td>
<td>85%</td>
<td>0.23</td>
</tr>
</tbody>
</table>

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.
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]

x(1): find the meeting on Wednesday with Bob and Carol

G[y(1)]:
  event ::= "QueryEvent(" constraint ")"
  constraint ::= "(&" constraint constraint ")"
  | "(start_? date ")"
  | "(attendee_? attendee attendee ")"
  date ::= "Wednesday"
  attendee ::= "Bob" | "Carol"

y(1): QueryEvent(& (start_? Wednesday)(attendee_? Bob Carol))

x: Add meeting with Jean's manager on Wednesday at 3PM

---

### LLM Output

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

y: CreateEvent(& (start_? Wednesday NumberPM(3))(attendee_? FindManager(Jean)))
```

Wang et al. 2023