Evaluation: Metrics and Benchmarks

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

With slides from Zora Wang and Nikitha Rao
The NL2Code Task

- Given a natural language instruction $Q$, generate code implementation $C$

```
Calculate sum over all rows of 2D numpy array
```
```
a.sum(axis=1)
```
The Landscape for NL2Code Generation

- Transition of Evaluation Metrics:
  - Lexical
  - Neural based metrics
  - Test case execution

- Domain Coverage
  - Built-in grammar: \( \text{sum([1, 2, 4])} \)
  - Domain-specific: data science
  - Open domain: diverse Python libraries

- Functional Complexity
  - Simple (toy) functions: e.g., LeetCode
  - Class level
  - Repository level

- Test Automation
  - Human-written tests
  - Fuzzing methods
  - Integrating LLMs
Most code snippets were short, and evaluated using BLEU or exact match.
Datasets were fairly large, with dedicated training sets.

<table>
<thead>
<tr>
<th>Natural Language</th>
<th>Bash Command(s)</th>
</tr>
</thead>
</table>
| find .java files in the current directory tree that contain the pattern ‘TODO’ and print their names | grep -l "TODO" *.java
find . -name "*.java" -exec grep -il "TODO" {} \;
find . -name "*.java" | xargs -I {} grep -l "TODO" {} |
| display the 5 largest files in the current directory and its sub-directories   | find . -type f | sort -nk 5,5 | tail -5
du -a . | sort -rh | head -n5
find . -type f -printf ’%s %p
’ | sort -rn | head -n5 |
| search for all jpg images on the system and archive them to tar ball “images.tar” | tar -cvf images.tar $(find / -type f -name *.jpg)
tar -rvf images.tar $(find / -type f -name *.jpg)
find / -type f -name "*.jpg" -exec tar -cvf images.tar {} \;  

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td># pairs</td>
<td>8,090</td>
<td>609</td>
<td>606</td>
</tr>
<tr>
<td># unique nls</td>
<td>7,340</td>
<td>549</td>
<td>547</td>
</tr>
</tbody>
</table>
Most code snippets were short, and evaluated using BLEU or exact match.
Datasets were fairly large, with dedicated training sets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PL</th>
<th>Num. pairs</th>
<th>Num. words</th>
<th>Num. tokens</th>
<th>Avg. # w. in NL</th>
<th>Avg. # t. in code</th>
<th>NL collection</th>
<th>Code collection</th>
<th>Semantic alignment</th>
<th>Introduced by</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFTTT</td>
<td>DSL</td>
<td>86,960</td>
<td>24,857</td>
<td>91,156</td>
<td>7.0</td>
<td>21.8</td>
<td>scraped</td>
<td>scraped</td>
<td>Noisy</td>
<td>(Quirk et al., 2015)</td>
</tr>
<tr>
<td>C#2NL*</td>
<td>C#</td>
<td>66,015</td>
<td>24,857</td>
<td>91,156</td>
<td>12.0</td>
<td>38</td>
<td>scraped</td>
<td>scraped</td>
<td>Noisy</td>
<td>(Iyer et al., 2016)</td>
</tr>
<tr>
<td>SQL2NL*</td>
<td>SQL</td>
<td>32,337</td>
<td>10,086</td>
<td>1,287</td>
<td>9.0</td>
<td>46</td>
<td>scraped</td>
<td>scraped</td>
<td>Good*</td>
<td>(Zhong et al., 2018)</td>
</tr>
<tr>
<td>RegexLib</td>
<td>Regex</td>
<td>3,619</td>
<td>13,491</td>
<td>179*</td>
<td>36.4</td>
<td>58.8*</td>
<td>scraped</td>
<td>scraped</td>
<td>Good*</td>
<td>(Ling et al., 2016)</td>
</tr>
<tr>
<td>HeartStone</td>
<td>Python</td>
<td>665</td>
<td>–</td>
<td>–</td>
<td>7.0</td>
<td>352*</td>
<td>game card description</td>
<td>game card source code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTG</td>
<td>Java</td>
<td>13,297</td>
<td>–</td>
<td>21</td>
<td>1,080*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StaQC</td>
<td>Python</td>
<td>147,546</td>
<td>17,635</td>
<td>137,123</td>
<td>9.0</td>
<td>86</td>
<td>extracted using ML</td>
<td>extracted using ML</td>
<td>Noisy</td>
<td>(Yao et al., 2018)</td>
</tr>
<tr>
<td></td>
<td>SQL</td>
<td>119,519</td>
<td>9,920</td>
<td>21,413</td>
<td>9.0</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NL2RX</td>
<td>Regex</td>
<td>10,000</td>
<td>560</td>
<td>45*</td>
<td>10.6</td>
<td>26*</td>
<td>synthesized &amp; paraphrased</td>
<td>synthesized written</td>
<td>Very Good</td>
<td>(Locascio et al., 2016)</td>
</tr>
<tr>
<td>WikiSQL</td>
<td>SQL</td>
<td>80,654</td>
<td>–</td>
<td>–</td>
<td>10.9</td>
<td>16.0</td>
<td></td>
<td></td>
<td></td>
<td>(Zhong et al., 2017)</td>
</tr>
<tr>
<td>NLMAPS</td>
<td>DSL</td>
<td>2,380</td>
<td>1,014</td>
<td>–</td>
<td>10.9</td>
<td>16.0</td>
<td>synthesized given code</td>
<td>expert written</td>
<td>Very Good</td>
<td>(Haas and Riezler, 2016)</td>
</tr>
<tr>
<td>Jobs640*</td>
<td>DSL</td>
<td>640</td>
<td>391</td>
<td>58†</td>
<td>9.8</td>
<td>22.9</td>
<td>user written</td>
<td>expert written</td>
<td>Very Good</td>
<td>(Tang and Mooney, 2001)</td>
</tr>
<tr>
<td>GEO880</td>
<td>DSL</td>
<td>880</td>
<td>284</td>
<td>60†</td>
<td>7.6</td>
<td>19.1</td>
<td></td>
<td></td>
<td></td>
<td>(Zelle and Mooney, 1996)</td>
</tr>
<tr>
<td>Freebase917</td>
<td>DSL</td>
<td>917</td>
<td>–</td>
<td>–</td>
<td>11.1</td>
<td>28.1</td>
<td></td>
<td></td>
<td></td>
<td>(Cai and Yates, 2013)</td>
</tr>
<tr>
<td>ATIS*</td>
<td>DSL</td>
<td>5,410</td>
<td>936</td>
<td>176†</td>
<td>11.1</td>
<td>28.1</td>
<td></td>
<td></td>
<td></td>
<td>(Dahl et al., 1994)</td>
</tr>
<tr>
<td>WebQSP</td>
<td>DSL</td>
<td>4,737</td>
<td>–</td>
<td>–</td>
<td>14.3</td>
<td>–</td>
<td>search log</td>
<td></td>
<td></td>
<td>(Yih et al., 2016)</td>
</tr>
<tr>
<td>NL2RX-KB13</td>
<td>Regex</td>
<td>824</td>
<td>715</td>
<td>85*</td>
<td>7.1</td>
<td>19.0*</td>
<td>turker written</td>
<td></td>
<td></td>
<td>(Kushman and Barzilay, 2013)</td>
</tr>
<tr>
<td>Django*</td>
<td>Python</td>
<td>18,805</td>
<td>–</td>
<td>14.3</td>
<td>7.7</td>
<td>–</td>
<td>expert written</td>
<td></td>
<td></td>
<td>(Oda et al., 2015)</td>
</tr>
<tr>
<td>NL2Bash</td>
<td>Bash</td>
<td>9,305</td>
<td>7,790</td>
<td>6,234</td>
<td>11.7</td>
<td>7.7</td>
<td>scraped</td>
<td>scraped</td>
<td>Ours</td>
<td></td>
</tr>
</tbody>
</table>

Evaluation Metrics
Reference Matching: BLEU

- Developed for machine translation (Papineni et al. 2002)
- Compares n-gram overlap between predicted and reference
- Typically, uses n-grams up to 4 (BLEU-4)

<table>
<thead>
<tr>
<th>Reference: Taro visited Hanako</th>
</tr>
</thead>
<tbody>
<tr>
<td>System: the Taro visited the Hanako</td>
</tr>
<tr>
<td>1-gram: 3/5</td>
</tr>
<tr>
<td>2-gram: 1/4</td>
</tr>
<tr>
<td>Brevity: min(1,</td>
</tr>
</tbody>
</table>

\[
\text{BLEU-2} = (\frac{3}{5} \times \frac{1}{4})^{1/2} \times 1.0 \\
= 0.387
\]
Reference Matching: CodeBLEU

Higher weight for keywords

Match syntactic subtrees

Match data dependency graphs

Machine translation:
public static int Sign ( double d )
{
    return ((int)(((d == 0) ? 0 : (d < 0)))?
    -1 : 1);
}

1.0 1.0 0.7 0.5:

Reference (human) translation:
public static int Sign ( double d )
{
    return ((int)(((d == 0) ? 0 : (d < 0)))?
    -1 : 1):
}

Weighted N-Gram Match

Syntactic AST Match

Semantic Data-flow Match

CodeBLEU = α · N-Gram Match (BLEU) + β · Weighted N-Gram Match + γ · Syntactic AST Match + δ · Semantic Data-flow Match

Ren et al. 2020
When evaluating evaluation metrics, check correlation with human judgements.

In CodeBLEU: rate code outputs on a Likert scale of general quality (1=very bad; 5=very good)

Figure 5: BLEU and CodeBLEU predict human evaluation scores. (a) Text-to-code; (b) Code translation.
Issues: Evaluations Are Not Rigorous

Return list with elements incremented by 1

Output #1: 
\[ [i - 1 \text{ for } i \text{ in } l] \]
High lexical overlap with the solution

Test: 
\[ l = [1, 2, 3] \]
Output: 
\[ [0, 1, 2] \]
Mismatch

Solution:

\[
\text{ret_list} = [] \\
\text{for } e \text{ in } l: \\
\quad \text{ret_list}.append(e + 1) \\
\text{return } \text{ret_list}
\]

Output #2:
\[ [i + 1 \text{ for } i \text{ in } l] \]
Low lexical overlap with the canonical solution, But actually correct

Output: 
\[ [2, 3, 4] \]
Match
HumanEval Benchmark

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

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

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

- Similar to HumanEval, but a bit easier
- 974 short Python problems, written by crowdworkers
  - 58% mathematical, 43% list processing, 19% string processing

Austin et al. 2021
Model performance is sensitive to sampling temperature and number of candidates (similar findings in HumanEval/Codex paper)

Figure 9: Higher temperatures achieve better scaling with more samples, but perform worse with a smaller budget.
BLEU against a reference solution is uncorrelated with whether samples pass execution tests (similar findings in Codex paper).

Figure 10: Comparison of BLEU score and synthesis performance for the 137B parameter model. No strong correlation is observed.
Model evaluated is a large Google LLM, LaMDA, trained mostly on natural language, which has some interaction ability.

Figure 12: An overview of the “flow” of the human-model collaboration experiments. The human gives a description of the desired program and then guides the model toward the correct solution via dialog.

Figure 13: Percent of problems solved as the number of human dialog interventions increases. With 4 interventions, the solve rate increases from 30% to over 65%. Except for the purple horizontal baseline (which corresponds to 5 samples from the model), all pass-rates in this figure were computed using a single sample from the model.
Automated & Improved Testing

- EvalPlus: use LLMs and **fuzzing** (type-aware mutation) to create test cases
- Prompt ChatGPT with the GT solution, some inputs, and instructions to generate more
EvalPlus: use LLMs and *fuzzing* (type-aware mutation) to create test cases.

Fuzzing: mutate inputs to the functions, apply the groundtruth function, and use the input-output pair to make a new test case.

<table>
<thead>
<tr>
<th>Type</th>
<th>Mutation</th>
<th>Type</th>
<th>Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>float</td>
<td>Returns (x \pm 1)</td>
<td>List</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Insert/replace (x[i]) with \text{Mutate}(x[i])</td>
</tr>
<tr>
<td>bool</td>
<td>Returns a random boolean</td>
<td>Tuple</td>
<td>Returns \text{Tuple(Mutate(List(x)))}</td>
</tr>
<tr>
<td>NoneType</td>
<td>Returns None</td>
<td>Set</td>
<td>Returns \text{Set(Mutate(List(x)))}</td>
</tr>
<tr>
<td>str</td>
<td>{ Remove a sub-string (s)</td>
<td>Dict</td>
<td>Remove a key-value pair (k \rightarrow v)</td>
</tr>
<tr>
<td></td>
<td>Repeat a sub-string (s)</td>
<td></td>
<td>Update (k \rightarrow v) to (k \rightarrow \text{Mutate}(v))</td>
</tr>
<tr>
<td></td>
<td>Replace (s) with \text{Mutate}(s)</td>
<td></td>
<td>Insert \text{Mutate}(k) \rightarrow \text{Mutate}(v)</td>
</tr>
</tbody>
</table>
Automated & Improved Testing

- EvalPlus: use LLMs and *fuzzing* (type-aware mutation) to create test cases
- Optionally, minify the test sets while preserving code coverage and edge case detection.

### Table 2: Overview of EvalPlus-improved benchmarks.

<table>
<thead>
<tr>
<th></th>
<th>#Tests</th>
<th>#Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>HUMAN_EVAL</strong></td>
<td>9.6</td>
<td>7.0</td>
</tr>
<tr>
<td><strong>HUMAN_EVAL⁺</strong></td>
<td>764.1</td>
<td>982.5</td>
</tr>
<tr>
<td><strong>HUMAN_EVAL⁺-MINI</strong></td>
<td>16.1</td>
<td>13.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
<th>pass@k</th>
<th>k=1⁺</th>
<th>k=1</th>
<th>k=10</th>
<th>k=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-4 [49]</td>
<td>N/A</td>
<td>base</td>
<td>88.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+extra</td>
<td>76.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phind-CodeLlama [52]</td>
<td>34B</td>
<td>base</td>
<td>71.3</td>
<td>90.5</td>
<td>96.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+extra</td>
<td>67.1</td>
<td>85.0</td>
<td>92.5</td>
<td></td>
</tr>
<tr>
<td>WizardCoder-CodeLlama [38]</td>
<td>34B</td>
<td>base</td>
<td>73.2</td>
<td>85.2</td>
<td>94.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+extra</td>
<td>64.6</td>
<td>78.6</td>
<td>88.9</td>
<td></td>
</tr>
<tr>
<td>ChatGPT [48]</td>
<td>N/A</td>
<td>base</td>
<td>73.2</td>
<td>88.6</td>
<td>94.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+extra</td>
<td>63.4</td>
<td>82.1</td>
<td>91.1</td>
<td></td>
</tr>
</tbody>
</table>

Liu et al. 2023
Automated & Improved Testing

- EvalPlus: use LLMs and *fuzzing* (type-aware mutation) to create test cases
- These extra tests substantially reduce the pass@k of many models!

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>pass@k</th>
<th>k=1*</th>
<th>k=1</th>
<th>k=10</th>
<th>k=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-4 [49]</td>
<td>N/A</td>
<td>base</td>
<td>88.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>+extra</td>
<td>76.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phind-CodeLlama [52]</td>
<td>34B</td>
<td>base</td>
<td>71.3</td>
<td>71.6</td>
<td>90.5</td>
<td>96.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+extra</td>
<td>67.1</td>
<td>67.0</td>
<td>85.0</td>
<td>92.5</td>
</tr>
<tr>
<td>WizardCoder-CodeLlama [38]</td>
<td>34B</td>
<td>base</td>
<td>73.2</td>
<td>61.6</td>
<td>85.2</td>
<td>94.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+extra</td>
<td>64.6</td>
<td>54.5</td>
<td>78.6</td>
<td>88.9</td>
</tr>
<tr>
<td>ChatGPT [48]</td>
<td>N/A</td>
<td>base</td>
<td>73.2</td>
<td>69.4</td>
<td>88.6</td>
<td>94.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+extra</td>
<td>63.4</td>
<td>62.5</td>
<td>82.1</td>
<td>91.1</td>
</tr>
</tbody>
</table>
Key idea: it’s relatively easy to translate test cases on simple types (e.g. no matrices or functions) from Python to other languages.

This allows porting HumanEval & MBPP to 18 other languages.

(a) Original Python assertion.
```python
assert lsi([0]) == (None, None)
```

(b) Equivalent R.
```r
if(!identical(lsi(c(0)), c(NULL, NULL))){
  quit('no', 1)}
```

(c) Equivalent JavaScript.
```javascript
assert.deepEqual(lsi([0]), [void 0, void 0]);
```

Figure 4: Example of a translated assertion.

(a) Original Python docstring from HumanEval #95.
```
Given a dictionary, return True if all keys are strings in lower case or all keys are strings in upper case, else return False. The function should return False is the given dictionary is empty.
```

(b) Terminology translated to Perl.
```
Given a hash, return 1 if all keys are strings in lower case or all keys are strings in upper case, else return "". The function should return "" is the given hash is empty.
```

Figure 5: A Python docstring and its Perl translation. Errors (e.g., “is” for “if”) are from the original benchmark.
MultiPL-E

- Models are generally better on “high-resource” languages with more code on GitHub.
- More analysis of this in the Data lecture, with Starcoder.

Cassano et al. 2022
Incorrect Code Can Be Valuable Too!

- Code might be easily editable to achieve a good solution.

Levenshtein distance: number of character edits required to transform.

\[
\text{EDIT-SIM} = 1 - \frac{\text{lev}(\text{gen}, \text{ref})}{\max(\text{len}(\text{gen}), \text{len}(\text{ref}))}
\]

Reference Code Snippet
```python
def even_odd_count(num):
    even_count = 0
    odd_count = 0
    for i in str(abs(num)):
        if int(i)%2==0:
            even_count +=1
        else:
            odd_count +=1
    return (even_count, odd_count)
```

Generated Code Snippet
```python
def even_odd_count(num):
    even_count = 0
    odd_count = 0
    for i in str(num):
        if int(i) % 2 == 0:
            even_count += 1
        else:
            odd_count += 1
    return even_count, odd_count
```

Functional Metric: \(\text{pass} = 0\)

Similarity Metric: edit similarity = 0.93

Human preference: preference = 0.9

Dibia et al. 2022
Incorrect Code Can Be Valuable Too!

- Dibia et al. compare metrics to evaluate 5 model outputs on HumanEval.
  - EditDistance, BLEU, Pass@1

- Professional programmers with Python experience rate on:
  - **Accuracy**: judge if the snippets are functionally equivalent (judging is easier than writing!)
  - **Value**: How useful is the snippet as a starting point?
  - **Effort**: how much effort to modify the solution into a correct one?
Incorrect Code Can Be Valuable Too!

- Value is nearly perfectly correlated with effort (accuracy less so).
- Of all metrics, Pass@1 is most correlated with Value.
- But, Edit sim > BLEU and a combination is best (as dissimilar, incorrect code is bad).

\[
\text{COMBINED} = \min(1.0, \text{Pass} + \text{Edit-Sim})
\]

<table>
<thead>
<tr>
<th>Human Judgement</th>
<th>Offline Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>Value</td>
<td>1.00</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.87</td>
</tr>
<tr>
<td>Effort</td>
<td>0.94</td>
</tr>
<tr>
<td>Pass</td>
<td>0.61</td>
</tr>
<tr>
<td>Edit Sim</td>
<td>0.48</td>
</tr>
<tr>
<td>bleu</td>
<td>0.36</td>
</tr>
<tr>
<td>Combined</td>
<td>0.70</td>
</tr>
</tbody>
</table>
Captures some intuitions about incorrect code being useful
- BLEU and edit distance only give points for exactly matching code
- Takes NL code descriptions into account

Use vector similarity from CodeBERT representations
- Recall: every reference vector has $\geq 1$ candidate vector with high similarity
- Precision: every candidate vector has $\geq 1$ reference vector with high similarity

Zhou et al. 2023
Domains of Code
HumanEval Looks Like Toy Examples?

- HumanEval Examples

```python
def incr_list(lst):
    """Return list with elements incremented by 1."
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in lst]

def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements that are in even positions."
    Examples
    solution([5, 8, 7, 1]) => 12
    solution([3, 3, 3, 3, 3]) => 9
    solution([30, 13, 24, 321]) => 0
    return sum(lst[i] for i in range(0, len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

- Real-World Development

```
# I am writing a program that accepts user input.

750
# Note: Python 2.7 users should use 'raw_input', the equivalent of 3.x's 'input'
# age = int(input("Please enter your age: "))
if age >= 18:
    print("You are able to vote in the United States!")
else:
    print("You are not able to vote in the United States.")
```
Broadening Domains

- Leetcode Style: HumanEval, APPS, MBPP
  - Manually written or collected from code contest websites
  - Only uses Python built-in grammar
- Limited Domains: e.g., Data Science
  - DS-1000: StackOverflow questions
  - ARCADE: Interactive Jupyter Notebooks
  - ...
- Open Domain: ODEX
  - 79 Python libraries
  - Four natural languages
APPS

- 10,000 problems taken from contest and exercise sites
- 3 difficulty levels
- Has test cases

Problem
You are given two integers $n$ and $m$. Calculate the number of pairs of arrays $(a, b)$ such that: the length of both arrays is equal to $m$; each element of each array is an integer between 1 and $n$ (inclusive); $a_i \leq b_i$ for any index $i$ from 1 to $m$; array $a$ is sorted in non-descending order; array $b$ is sorted in non-ascending order. As the result can be very large, you should print it modulo $10^9 + 7$.

Input: The only line contains two integers $n$ and $m$ ($1 \leq n \leq 1000$, $1 \leq m \leq 10$). Output: Print one integer – the number of arrays $a$ and $b$ satisfying the conditions described above modulo $10^9 + 7$.

Model Output

```python
n, m = map(int, input().split())
mod = 10**9+7

fact = [1, 1]
factinv = [1, 1]
inv = [0, 1]

for i in range(2, n+1):
    fact.append((fact[-1] * i) % mod)
    inv.append((-inv[mod % i] * (mod//i)) % mod)
    factinv.append((factinv[-1] * inv[-1]) % mod)

def cmb(n, r, mod):
    if (r < 0) or (n < r):
        return 0
    r = min(r, n - r)
    return fact[n] * factinv[r] * factinv[n-r] % mod

print((cmb(n, m, mod) * cmb(m-1, n-1, mod)) % mod)
```
CodeContests (AlphaCode)

- Similar to APPS but focuses on improving test coverage, via input mutation (like EvalPlus).

<table>
<thead>
<tr>
<th>Split</th>
<th>Problems</th>
<th>Tests per problem</th>
<th>Solutions per problem (% correct)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Example</td>
<td>Hidden</td>
</tr>
<tr>
<td>Train</td>
<td>13328</td>
<td>2.0</td>
<td>14.8</td>
</tr>
<tr>
<td>Valid</td>
<td>117</td>
<td>1.5</td>
<td>12.9</td>
</tr>
<tr>
<td>Test</td>
<td>165</td>
<td>1.7</td>
<td>9.4</td>
</tr>
</tbody>
</table>

- Manual inspection shows high false-positive rate of model-produced solutions.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Tests / problem</th>
<th>False Positive (FP) Rate</th>
<th>FP or Slow Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPS</td>
<td>20.99</td>
<td>60%</td>
<td>70%</td>
</tr>
<tr>
<td>HumanEval</td>
<td>7.77</td>
<td>30%</td>
<td>N/A</td>
</tr>
<tr>
<td>CodeContests raw</td>
<td>12.4</td>
<td>62%</td>
<td>88%</td>
</tr>
<tr>
<td>CodeContests</td>
<td><strong>203.7</strong></td>
<td><strong>4%</strong></td>
<td><strong>46%</strong></td>
</tr>
</tbody>
</table>
DS-1000

- 1,000 data science problems, based on StackOverflow questions
- Domain-specific test cases, e.g. matplotlib plots have their elements programatically extracted

1. Manually Selecting and Modifying StackOverflow Problems
   
   Here is a sample dataframe:

   ```python
   df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
   
   I'd like to add inverses of each existing column to the dataframe and ... [omitted for brevity]
   
   try:
   inv_df = df.join(df.apply(lambda x: 1/x).add_prefix('inv_')))```

2. Adding Code Context
   
   ```python
   import pandas as pd
   df = pd.DataFrame({'A': [1, 2, 3],
                     'B': [4, 5, 6]})
   
   ### BEGIN SOLUTION
   [insert]
   ### END SOLUTION
   print(result)
   
3. Implementing Automatic Tests
   
   Test cases
   ...
   [omit for brevity]
   
   ```python
   pd.testing.assert_frame_equal(result, ans)
   
   Surface-form constraints
   for and while should not appear in Syntax Tree```

4. Perturbing Original Problem
   
   ... I'd like to apply the exponential function to each existing column ... The resulting dataframe should look like so:

   ```python
   result = pd.DataFrame({'A': [1, 2, 3],
                         'B': [4, 5, 6],
                         'exp_A': [e^1, e^2, e^3],
                         'exp_B': [e^4, e^5, e^6]})
   
   ... [omitted for brevity]```

5. Red Teaming
   
   ```python
   df = pd.DataFrame({'A': [1, 2, 3],
                     'B': [4, 5, 6]})
   
   ### BEGIN SOLUTION
   # A known WRONG SOLUTION
   result = df.join(df.apply(lambda x: math.e**x).add_prefix('exp_'))
   ### END SOLUTION
   print(result)```
Perturb the problems to reduce chances of memorization, since models may have been trained on StackOverflow

”Surface” perturbations: don’t change solution. “Semantic”: do, but try to keep difficulty the same (e.g. max -> min)

<table>
<thead>
<tr>
<th></th>
<th>Pandas</th>
<th>NumPy</th>
<th>Scikit-learn</th>
<th>SciPy</th>
<th>TensorFlow</th>
<th>PyTorch</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin_{surface}</td>
<td>37.3</td>
<td>61.2</td>
<td>52.6</td>
<td>33.0</td>
<td>64.9</td>
<td>64.8</td>
<td>53.2</td>
</tr>
<tr>
<td>Surface</td>
<td>31.9</td>
<td>−5.4</td>
<td>58.4</td>
<td>−2.8</td>
<td>55.7</td>
<td>32.1</td>
<td>−0.9</td>
</tr>
<tr>
<td>Origin_{semantic}</td>
<td>36.8</td>
<td>56.7</td>
<td>60.6*</td>
<td>40.3</td>
<td>71.3</td>
<td>65.1</td>
<td>47.2</td>
</tr>
<tr>
<td>Semantic</td>
<td>33.2</td>
<td>−3.6</td>
<td>49.0</td>
<td>−7.7</td>
<td>38.9−21.7</td>
<td>34.3</td>
<td>−6.0</td>
</tr>
<tr>
<td>Origin_{difficult}</td>
<td>39.9</td>
<td>52.7</td>
<td>5.0*</td>
<td>58.1</td>
<td>73.0*</td>
<td>53.8*</td>
<td>46.8</td>
</tr>
<tr>
<td>Difficult Rewrite</td>
<td>17.7</td>
<td>−22.2</td>
<td>27.1</td>
<td>−25.6</td>
<td>0.0−5.0</td>
<td>13.8</td>
<td>−44.3</td>
</tr>
</tbody>
</table>
executable problems from Jupyter notebooks

Which countries host at least two Olympic games?

```python
# Solution: Let's solve this problem step-by-step.
import pandas as pd

df = pd.read_csv('stores.csv')

# Prompt
Which countries host at least two Olympic games?

# Step 1: Get the counts each country hosted Olympics
count_df = df['Country'].value_counts()

# Step 2: Get the rows whose average score is above 90
filtered_df = count_df[count_df >= 2]

# Step 3: Get the country names as a list
filtered_df.index.tolist()
```

### Models

<table>
<thead>
<tr>
<th>Models</th>
<th>pass@30</th>
<th># API</th>
<th>Lines of Code (LoC)</th>
<th>Comment Lines</th>
<th>Tokens / Line</th>
<th>API / Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (Tab. 2)</td>
<td>47.7</td>
<td>4.9</td>
<td>2.3</td>
<td>0.1</td>
<td>21.1</td>
<td>3.2</td>
</tr>
<tr>
<td>+ More Context</td>
<td>49.3</td>
<td>4.9</td>
<td>2.3</td>
<td>0</td>
<td>21.1</td>
<td>3.1</td>
</tr>
</tbody>
</table>

#### Prompting with Additional Few-shot Exemplars

| Vanilla Code            | 49.9    | 5.3  | 2.4                 | 0.1           | 20.8          | 3.1        |
| Step-by-Step Code       | 51.9    | 5.6  | 3.2                 | 0.1           | 17.8          | 2.7        |
| + Preamble              | 51.9    | 5.9  | 3.5                 | 0.2           | 16.9          | 2.5        |
| + Pre. + Explanation    | 52.5    | 6.8  | 4.2                 | 3.3           | 14.9          | 2.2        |

#### Step-by-Step Prompting (with examples):

```python
# Solution: Let's solve this problem step-by-step.
import pandas as pd

df = pd.read_csv('stores.csv')

# Prompt
Which countries host at least two Olympic games?

# 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']]```
ODEX: Open-Domain, with Evaluation

- Larger Domain Coverage
- Test execution on real-world coding queries
  - Collected from StackOverflow questions
- Support four natural languages as input
  - English, Spanish, Japanese, Russian
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Samples</th>
<th>Domain</th>
<th>Executable?</th>
<th>Avg. Test Cases</th>
<th>Data Source</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>JuICe (Agashe et al., 2019)</td>
<td>1,981</td>
<td>open</td>
<td>✗</td>
<td></td>
<td>GitHub Notebooks</td>
<td>en</td>
</tr>
<tr>
<td>HumanEval (Chen et al., 2021)</td>
<td>164</td>
<td>4</td>
<td>✓</td>
<td>7.7</td>
<td>Hand-written</td>
<td>en</td>
</tr>
<tr>
<td>MBPP (Austin et al., 2021)</td>
<td>974</td>
<td>8</td>
<td>✓</td>
<td>3.0</td>
<td>Hand-written</td>
<td>en</td>
</tr>
<tr>
<td>APPS (Hendrycks et al., 2021)</td>
<td>10,000</td>
<td>0</td>
<td>✓</td>
<td>13.2</td>
<td>Competitions</td>
<td>en</td>
</tr>
<tr>
<td>DSP (Chandel et al., 2022)</td>
<td>1,119</td>
<td>16</td>
<td>✓</td>
<td>2.1</td>
<td>Github Notebooks</td>
<td>en</td>
</tr>
<tr>
<td>MTPB (Nijkamp et al., 2022)</td>
<td>115</td>
<td>8</td>
<td>✓</td>
<td>5.0</td>
<td>Hand-written</td>
<td>en</td>
</tr>
<tr>
<td>Exe-DS (Huang et al., 2022)</td>
<td>534</td>
<td>28</td>
<td>✓</td>
<td></td>
<td>GitHub Notebooks</td>
<td>en</td>
</tr>
<tr>
<td>DS-1000 (Lai et al., 2022)</td>
<td>1,000</td>
<td>7</td>
<td>✓</td>
<td>1.6</td>
<td>StackOverflow</td>
<td>en</td>
</tr>
<tr>
<td>CoNaLa (Yin et al., 2018)</td>
<td>2,879</td>
<td>open</td>
<td>✗</td>
<td></td>
<td>StackOverflow</td>
<td>en</td>
</tr>
<tr>
<td>MCoNaLa (Wang et al., 2022)</td>
<td>896</td>
<td>open</td>
<td>✗</td>
<td></td>
<td>StackOverflow</td>
<td>es, ja, ru</td>
</tr>
<tr>
<td>ODEX</td>
<td>945</td>
<td>79</td>
<td>✓</td>
<td>1.8</td>
<td>Hand-Written</td>
<td>en, es, ja, ru</td>
</tr>
</tbody>
</table>

**Calculate the improper integral given by the function f from the number `n` to infinity**

```python
import sympy

def function(f, n)
    return sympy.integrate(f, (sympy.Symbol('x'), n, sympy.oo))
```

# test case
x = sympy.Symbol('x')
f = (x * x)
n = 1
assert str(function(f, n)) == 'oo'
ODEX: Unique Challenges for Execution

Closed-domain code: easy to execute and verify

Open-domain code:
- Random outputs
- Specialized verification
- (Potentially) not reproducible queries
  - HTTP requests, e.g., `requests.post("https://def.xyz", data={'key': 'value'})`

```
assert func([1, 2, 10]) == [2, 3, 11]
```

```
In [1]: import numpy as np
In [2]: a = np.array([1, 2, 3])
In [3]: b = np.array([1, 2, 3])

ValueError: The truth value of an array with more than one element is ambiguous.
```

```
In [4]: assert (a == b)
```

```
Out[5]: True
```
Significant Performance Gaps: Open vs. Closed

- Although Codex performs better overall
- CodeGen has smaller domain gaps

Figure 7: CODEX (left) and CODEGEN (right) pass@1 on open- and closed-domain problems in each language.
Code Complexity
### Functional Complexity

- **Function Level:** HumanEval, MBPP
- **Class Level:** ClassEval

---

**HumanEval Function Test**

```python
METADATA = { 'author': 'JH', 'dataset': 'test' }

def check(candidate):
    assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) == True
    assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) == False
```

**MBPP Function Test**

```python
["assert get_ludic(10) == [1, 2, 3, 5, 7]",
 "assert get_ludic(25) == [1, 2, 3, 5, 7, 11, 13, 17, 23, 25]", ...]
```

---

**ClassEval Class Test**

```python
def tearDown(self):
    self.assertEqual(self.vm.inventory, {"Coke": ('price': 1.25, 'quantity': 10)})
    self.assertEqual(self.vm.balance, 0)
```

---

**Figure 1:** Examples in Existing Benchmarks

- Write a python function to find the first repeated character in a given string.

**Figure 2:** An Example of Class Skeleton in ClassEval

```python
>>> vendingMachine.inventory = {'Coke': ('price': 1.25, 'quantity': 10})
>>> vendingMachine.restock_item('Coke', 10)
True
>>> vendingMachine.inventory['Coke'] = ('price': 1.25, 'quantity': 20)
```

**Figure 4:** Test Cases in Existing Benchmarks and ClassEval

- This is a class to simulate a vending machine, including adding products, inserting coins, purchasing products, viewing balance, replenishing product inventory, and displaying product information.

```python
from datetime import datetime

class VendingMachine:
    def __init__(self):
        initializes the vending machine's inventory and balance.
        self.inventory = {}
        self.balance = 0

def purchase_item(self, item_name):
    purchases a product from the vending machine and returns the balance after the purchase.
```
Functional Complexity

- **Function Level:** HumanEval, MBPP
- **Class Level:** ClassEval
- **Repository Level:**
  - RepoCoder
    - Retrieval-augmented generation
    - Multiple iterations
  - RepoEval
    - Collected 14 Github Repositories
- **Metrics:**
  - exact match
  - exact similarity
  - execution

![Visual example](image)

**Figure 3:** A visual example demonstrating the format of the RepoCoder prompt, which combines the retrieved code snippets from the repository with the unfinished code present in the target file.