Natural Language to Code Generation

Pengcheng Yin
pcyin@google.com

Google DeepMind
The Problem: Natural Language to Code Generation

Translate a user’s **natural language intents** into machine-executable **programs**

```python
my_list = [3, 5, 1]
sorted(my_list, reverse=True)
```
From Semantic Parsing to General-purpose Code Generation

Semantic Parsing to Domain-specific Formal Meaning Representations

- Show me flights from Pittsburgh to SFO
- $\lambda\ e\ (and\ (flight\ e)\ (from\ e\ pittsburgh:ci)\ (to\ e\ san\_francisco:ci))$

lambda-calculus logical form

Code Generation to General-purpose Programming Languages

- Sort my_list in descending order
- $\text{sorted(my_list, reverse=True)}$

Python code

(Images of logos for Alexa and Hey Siri for context)
Natural Language Intent

Sort my_list in descending order

Abstract Syntax Tree (AST)

Python Source Code

sorted(my_list, reverse=True)

Pre-LLMs: Syntax-driven Generation Methods

- Use Abstract Syntax Trees as general-purpose intermediate meaning representations
- $p_\theta(\text{Expr} \mid \text{Call})$ is a seq-to-tree model using program grammar as prior syntactic knowledge to constrain decoding space
- Deterministic transformation to source code

[Yin and Neubig 2017, 2018]
Factorize the generation process of an AST into sequential applications of tree-constructing actions \( \{a_t\} \)

Derivation Abstract Syntax Tree

Action Sequence

\[ \text{sorted(my\_list, reverse=True)} \]

[Yin and Neubig 2017]
Large Language Models (LLMs) for Code Generation

General Natural Language Pre-training

PaLM (540B Parameters)
- 50% social media conversations
- 30% filtered Web documents
- 5% Github Code (39B tokens)
  (780B tokens in total)

PaLM-Coder (based on PaLM 540B)
- Additional 8B multilingual code tokens
  (including 5B Python tokens)
- Also mix with small % of NL data

2nd-Stage Code-specific Training

Prompt:
```python
def find_k_largest(arr, k):
    # return the k largest elements in the input array
    result = sorted(arr, reverse=True)[:k]
    return result
```

Code Generation as a Prompting Task

Other more-recent Code LLMs:
- Code LLaMA
- DeepSeek Coder


[Chowdhery et al., 2022]
Input Utterance
Show me flights from Pittsburgh to SFO

<table>
<thead>
<tr>
<th>Flight</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FlightNo</td>
<td>Primary Key</td>
</tr>
<tr>
<td>Departure</td>
<td></td>
</tr>
<tr>
<td>Arrival</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>CityName</td>
<td>String</td>
</tr>
<tr>
<td>PublicTransport</td>
<td></td>
</tr>
</tbody>
</table>

SQL Query
```
SELECT Flight.FlightNo
FROM Flight
JOIN Airport as DepAirport
    ON Flight.Departure == DepAirport.Name
JOIN Airport as ArvAirport
    ON Flight.Arrival == ArvAirport.Name
WHERE DepAirport.CityName == Pittsburgh
    AND ArvAirport.CityName == San_Francisco
```
General-purpose Code Generation: Python Algorithmic Problems

HumanEval Doc-string2Code (Chen et al., 2021)

```python
def sum_odd_elements(lst):
    """given 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 list[i] % 2 == 1])
```

MBPP NL description + tests (Austin et al., 2021)

Write a function to find the smallest missing element in a sorted array. Your code should satisfy these tests:

```python
assert smallest_missing([0, 1, 2, 3, 4, 5, 6], 0, 6) == 7
assert smallest_missing([0, 1, 2, 6, 9, 11, 15], 0, 6) == 3
assert smallest_missing([1, 2, 3, 4, 6, 9, 11, 15], 0, 7) == 0
```

```python
def smallest_missing(arr, n, m):
    smallest = min(n, m)
    for i in range(n, m + 1):
        if arr[i] <= smallest:
            smallest += 1
    return smallest
```

[Chen et al., 2021; Austin et al., 2021]
**Competition Level Programming: APPS/CodeContests**

**Problem**

H-Index

Given a list of citations counts, where each citation is a nonnegative integer, write a function h_index that outputs the h-index. The h-index is the largest number h such that h papers have each least h citations.

Example:
Input: [3,0,6,1,4]
Output: 3

**Generated Code**

```python
def h_index(counts):
    n = len(counts)
    if n > 0:
        counts.sort()
        counts.reverse()
        h = 0
        while (h < n and counts[h]-1>=h):
            h += 1
        return h
    else:
        return 0
```

**Test Cases**

Input: [1,4,1,4,2,1,3,5,6]
Generated Code Output: 4

Input: [1000,500,500,250,100,100,100,100,75,50,30,20,15,15,10,5,2,1]
Generated Code Output: 15

An example competition-level coding problem (figure from from Hendrycks et al. 2021)

import json, pathlib, nbconvert, nbformat
import streamlit as st

# Get the path of Jupyter notebooks under '~/datasets/
root_path = pathlib.Path('~/datasets/').expand_user()
notebook_files = [p for p in root_path.glob('*ipynb')]

# Load file contents to a dict with file path as key.
notebook_contents = {fp: nbformat.load(open(fp)) for fp in notebook_files}

# Define a function to render a notebook in HTML.
def convert_notebook_to_html(notebook):
    exporter = nbconvert.HTMLExporter()
    return exporter.from_notebook_node(notebook)[0]

# Build a streamlit app to visualize notebooks in html
nb_to_view = st.selectbox("Choose a notebook to view:", notebook_files)
selected_notebook = notebook_contents[nb_to_view]
st.write(convert_notebook_to_html(selected_notebook))

---

How developers prompt LLMs in AI pair programming

- Succinct or under-specified intents
- Rich programmatic contexts
- Multi-turn NL2Code interaction
- Open-ended tasks

[Barke et al., 2022; Nijkamp et al., 2022]
Challenges in real-world interaction with coding assistants

Real-world Interaction

- Succinct and potentially under-specified intents
- Multi-turn interaction with rich code contexts
- Open-ended tasks

Existing Datasets

- Elaborate specifications and test cases
- No multi-turn problems or rich contexts
- Simple problems using basic data structures

```python
def solution(lst):
    """given 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 list[i] % 2 == 1])
```

HumanEval Doc-string2Code (Chen et al., 2021)
Natural language to Code Generation: Agenda

Instruction Tuning

Modeling Context and Multi-turn Interaction

Decoding and Reasoning Methods
(planning, consistency-based decoding, self-improvement)

Model Evaluation on Open-Domain Tasks
Natural language to Code Generation: Agenda

(Supervised) Instruction Tuning

Modeling Context and Multi-turn Interaction

Decoding and Reasoning Methods
(planning, consistency-based decoding, self-improvement)

Model Evaluation on Open-Domain Tasks
Instruction Tuning: Synthesize NL2Code Examples for Instruction Tuning

**Instruction Generation**
- Prompt LLMs to generate interview-style coding questions
- Focus on sample diversity (high temp)

**Generate Test Cases**
- Define the function signature and input/output specifications
- Focus on solution quality (greedy decoding)

**Generate Code Solutions**
- Generate code based on NL and tests
- Focus on sample diversity (high temp)
- Filtering based on test pass/fail results

**Self-Instruct**

[Wang et al., 2022; Honovich et al., 2022; Rozière et al., 2023]
Instruction Tuning: Synthesize NL2Code Problems for Instruction Tuning

- Other variants:
  - **WizardCoder**: iteratively evolve an instruction-code pair to a more complex version
  - **Textbooks are all you need**: distill textbook-style coding exercise data from LLMs
  - Both methods rely on strong teacher models

**Seed Instructions (with solutions)**

Create a Python program that creates a random password of 8 characters

**More complex instructions (with solutions)**

Create a Python program that generates a random password with 12 characters, including at least one uppercase letter, one special char from !@#$%^&

Example 1: Consider the matrix $A = \text{np.array}([[1, 2], [2, 4]])$. We can check if this matrix is singular or nonsingular using the determinant function. We can define a Python function, `is_singular(A)`, which returns true if the determinant of $A$ is zero, and false otherwise.

```python
import numpy as np
def is_singular(A):
    det = np.linalg.det(A)
    if det == 0: return True
    else: return False
A = np.array([[1, 2], [2, 4]])
print(is_singular(A)) # True
```

**WizardCoder** (Luo et al; 2023)

**Textbooks are all you need** (Gunasekar et al; 2023)
Instruction Tuning: Improve Diversity by Leveraging Code Data in the Wild

- **Idea:** use random Github code snippets to “inspire” an LLM to generate NL2Code problems in similar topics
- **Seed code snippets** ensures broad domain coverage

**Open-source codebase**

- PosNeg.py
- Log.cpp
- GrantInfo.ts
  
**Seed code snippet**

```python
learn_model(
    tf_idfSVM, tf_idfNB, target
)
def get_clean_review(raw_review):
    letters_only = re.sub(
        "[^a-zA-Z]", " ", raw_review)
```

**Prompt (details omitted)**

Please gain inspiration from the code snippet to create a high-quality programming problem...

**Language Model**

**OSS-INSTRUCT**

**Generated solution (details omitted)**

```python
from sklearn.feature_extraction.text import TfidfVectorizer ...
def get_clean_review(raw_review): ...
def train_model(tf_idfSVM, tf_idfNB, reviews, labels): ...
def classify_review(cleaned_review, tf_idfSVM, tf_idfNB): ...
train_model(tf_idfSVM, tf_idfNB, reviews, labels)
cleaned_review = get_clean_review(...)
```

**Generated problem (details omitted)**

You are working on a natural language processing (NLP) project and need to create a program to preprocess and classify movie reviews...

... Your program should be able to preprocess new movie reviews, train the model, and classify new reviews accurately.

**MagicCoder (Wei et al., 2023)**

- **Idea:** use random Github code snippets to “inspire” an LLM to generate NL2Code problems in similar topics
- **Seed code snippets** ensures broad domain coverage
Instruction Tuning: Leverage Noisy NL2Code Data in the Wild

- Mine noisy instruction-tuning like data from Github commits with high-precision heuristics and filters
- Does not need distillation from a teacher model
- Broad domain coverage (compared to interview-style problems)
- Contextualized instructions: Code + NL instruction → Code Solution
- NL instructions based on commit messages are often noisy and under-specified

OctoPack (Muennighoff et al., 2023)
Instruction Tuning: Learning to follow complex instructions with I/O specifications

1. Developer’s intent with I/O specifications

   **Task description + Additional I/O Specification**

   Get average duration of flights between cities for each airline

   **Input dataframe has columns such as**

   - airline
   - source_city

   **destination_city. Output dataframe has columns such as**

   - Airline, Delhi, Mumbai, Chennai

2. Predictions from different code LLMs

   df.groupby([
   'airline',
   'source_city',
   'destination_city']
).duration.mean()

   df.groupby(['airline',
   'source_city',
   'destination_city']).duration.mean().unstack(level=2)

3. Augment intents with I/O specifications derived from execution

   df.groupby(
   ['airline',
   'source_city',
   'destination_city']
).duration.mean().unstack(level=2)

   df.groupby(['airline',
   'source_city',
   'destination_city']).duration.mean().unstack(level=2)

---

1. **Instruct-tuning with synthetic Intents and Code**

   Generate intents with code context

   1. Show the top three countries with the highest GDP
      
      ```
      df.argmax('GDP')['Country'].tolist()
      ```

   2. What are the most populous cities in each country?
      
      ```
      df.groupby('Country').argmax('Population')
      ```

2. **Execute code and collect execution results**

   - Type: List
     - USA, China, Japan

   - Type: pandas.DataFrame
     - Country: USA, City: NYC, Population: 8,622,357

3. **Augment intents with I/O specifications derived from execution**

   - Show the top three countries with the highest GDP
     - Output is a list of string.

   - What are the most populous cities in each country?
     - Output is a dataframe with columns like Country, City, Population.
Natural language to Code Generation: Agenda

(Supervised) Instruction Tuning

Modeling Code Context and Multi-turn Interaction

Decoding and Reasoning Methods (planning, consistency-based decoding, self-improvement)

Model Evaluation on Open-Domain Tasks

```
import json, pathlib, nbconvert, nbformat
import streamlit as st

# Get the path of Jupyter notebooks under '~/datasets/'
root_path = pathlib.Path('~/datasets/').expand_user()
notebook_files = [p for p in root_path.glob('*.ipynb')]

# Load file contents to a dict with file path as key
notebook_contents = {fp: nbformat.load(open(fp)) for fp in notebook_files}

# Define a function to render a notebook in HTML
def convert_notebook_to_html(notebook):
    exporter = nbconvert.HTMLExporter()
    return exporter.from_notebook_node(notebook)[0]

# Build a streamlit app to visualize notebooks
nb_to_view = st.selectbox("Choose a notebook to view:", notebook_files)
selected_notebook = notebook_contents[nb_to_view]
st.write(convert_notebook_to_html(selected_notebook))
```
Modeling Context: In-IDE Multi-turn Code Generation

1. `import pandas as pd
   df = pd.read_csv('dataset/Gamepass_Games_v1.csv')`

2. Extract min and max hours as two columns
   ```python
   def get_hours(x):
       try:
           return float(x[0]), float(x[1])
       except:
           return 0, 0
   df['min'], df['max'] = zip(*df['TIME'].str.replace(' hours', '').str.split('-').apply(get_hours))
   ```

3. In which year was the most played game added?
   ```python
   df['GAMERS'] = df['GAMERS'].str.replace(',', '').astype(int)
   added_year = df['GAMERS'].idxmax()['ADDED'].year
   ```

4. What is the average maximum completion time for all `Fallout` games added that year?
   ```python
   fallout = df['GAME'].str.contains('Fallout')
   fallout.groupby(fallout['ADDED'].dt.year).get_group(added_year)['max'].mean()
   ```
Context-rich, Multi-turn Interaction

- Mix code, natural language, execution results
- Multi-turn (8-10) tasks with dependent context

```
import pandas as pd
df = pd.read_csv('dataset/Gamepass_Games_v1.csv')
```

```
def get_hours(x):
    try: return float(x[0]), float(x[1])
    except: return 0, 0

df['min'], df['max'] = zip(*df['TIME'].str.replace(' hours','').str.split('-').apply(get_hours))
```

Extract min and max hours as two columns

```
df['GAMERS']=df['GAMERS'].str.replace(',', '').astype(int)
added_year = df[df['GAMERS'].idxmax()]['ADDED'].year
```

In which year was the most played game added?

```
fal = df[df['GAME'].str.contains('Fallout')]
fal.groupby(fal['ADDED'].dt.year).get_group(added_year)['max'].mean()
```

What is the average maximum completion time for all fallout games added that year?

```
```
[5]

What is the amount of games added in each year for each month? (show a table with index as years, columns as months and fill null values with 0)

```
pd.pivot_table(df, index=df['ADDED'].dt.year, ..., aggfunc=np.count_nonzero,
                  fill_value='0').rename_axis(
                  index='Year', columns='Month')
```

Arcade: Answer Repository for Computational Analysis and Data Engineering.

Multi-turn code generation with succinct intents.

- **Context-rich, Multi-turn Interaction**
  - Mix code, natural language, execution results
  - Multi-turn (8-10) tasks with dependent context

- **Grounded Natural Language Understanding**
  - Requires understanding of variable contents (e.g., dataframe contents) and NL concepts.

- **Succinct and More Realistic Intents**
  - Intents often lack detailed specifications

[Yin et al., 2023]
import pandas as pd

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

# Schema of Dataframes:
# Columns in df with example values:
# name (Mike), subject (math), score (90), date (2021-05-01)

How many students took math courses this year?

df[(df['subject'] == 'math' &
    df['date'].dt.year == datetime.now().year)].count()

Is there a correlation between math and physics grade?

df[df['subject'].in(['math', 'physics'])].corr()

Plot the number of students in each letter grade range (A: >=90, B: 70-90, C: <70).

> LLM Completion:

df.score.apply(
    lambda x: 'A' if x >= 90 else
    ('B' if 70 <= x < 90 else 'C')
).value_counts().plot(kind='bar')
Arcade is a challenging benchmark Code LLMs on Arcade

Base PaLM model trained on 1.3T tokens

Fine-tune on Python code (64B tokens)

Fine-tune on Jupyter notebooks (10B tokens)

pass@5

30.6

21.4

12.5

12.0

8.9

4.6

3.0

7.0

3.8

0.0%
10.0%
20.0%
30.0%
40.0%

Arcade is a challenging benchmark Code LLMs on Arcade

[Yin et al., 2023]
Hi! I am working on this notebook:

```python
import pandas as pd
df = pd.read_csv('dataset/Gamepass_Games_v1.csv')
```

Can you extract min and max hours as two columns?

```python
def get_hours(x):
    ...
df['min'], df['max'] = df['hours'].str.split('-').apply(get_hours)
```

In which year was the most played game added?

```python
You can use `idxmax` to find the index of rows with max `GAMERS` and get the value of the `ADDED` column

max_year=df[df['GAMERS'].idxmax()]['ADDED'].year
```

Great! What is the average maximum completion time for all fallout games added that year?

```python
Just get the rows for the `max_year` above and ...

fallout = df[df['GAME'].str.contains('Fallout')]
fallback['ADDED'] == max_year['max'].mean()
```
Arcade for multi-turn evaluation of chat LLMs

[Yin et al., 2023]
Modeling Context: Cross-context Repository-level Code Generation

Task: given a GitHub issue description, generate a patch (multi-file code changes in a PR) that fixes the issue.

A two-stage baseline approach:
- Retrieval: Given NL issue description, retrieve relevant source code files that need to be edited.
- Code Generation: Generate a patch given issue description and retrieved relevant code files.

SWE-bench (Jimenez et al., 2023)
- Tasks are quite challenging for SoTA LLMs even given ground-truth source files to edit.
- Models struggle with understanding long contexts (worse performance with increased prompt length)

[Jimenez et al., 2023]
Natural language to Code Generation: Agenda

1. import json, pathlib, nbconvert, nbformat
2. import streamlit as st
3. 
4. # Get the path of Jupyter notebooks under '~/datasets/'
5. root_path = pathlib.Path('~/datasets/').expand_user()
6. notebook_files = [p for p in root_path.glob('*.ipynb')]
7. 
8. # Load file contents to a dict with file path as key
9. notebook_contents = {
10.     fp: nbformat.load(open(fp)) for fp in notebook_files
11. }
12. 
13. # Define a function to render a notebook in HTML
14. def convert_notebook_to_html(notebook):
15.     exporter = nbconvert.HTMLExporter()
16.     return exporter.from_notebook_node(notebook)[0]
17. 
18. # Build a streamlit app to visualize notebooks
19. nb_to_view = st.selectbox("Choose a notebook to view:", notebook_files)
20. selected_notebook = notebook_contents[nb_to_view]
21. st.write(convert_notebook_to_html(selected_notebook))

(Supervised) Instruction Tuning

Modeling Code Context and Multi-turn Interaction

Decoding and Reasoning Methods
(planning, consistency-based decoding, self-improvement)

Model Evaluation on Open-Domain Tasks
Decoding Methods: solving problems with step-by-step prompting

A Problem that requires a multi-step solution:

```python
import pandas as pd
df = pd.read_csv('scores.csv')

Plot the number of students in each letter grade range (A: >=90, B: 70-90, C: <70).
>_ LLM Completion...

df.score.apply(
    lambda x: 'A' if x >= 90 else
    ('B' if 70 <= x < 90 else 'C')
).value_counts().plot(kind='bar')
```

A vanilla solution:

Decompositional step-by-step decoding

[Yin et al., 2023; Jiang et al., 2023; Zelikman et al., 2023]
Decoding Methods: solving problems with step-by-step prompting

A Problem that requires a multi-step solution:

1. Import pandas as pd
   ```python
df = pd.read_csv('scores.csv')
```
2. Plot the number of students in each letter grade range (A: \(\geq 90\), B: 70-90, C: <70).
   ```python
   count_df = df['grade'].value_counts()
   count_df.plot(kind='bar')
   ```

Step-by-Step Planning in NL (Jiang et al., 2023):
Let's solve this problem step-by-step.
- Step 1: Define a function to convert scores to letter grades.
- Step 2: Convert scores to letter grades.
- Step 3: Count the number of students by grade.
- Step 4: Visualize in a bar chart.

Decompositional step-by-step decoding

Step-by-step Prediction with Explanations: (Yin et al., 2023)

```python
# Let's solve this problem step-by-step.
# Step 1: Define a function to convert scores to letter grades.
def get_grade(score):
    if score \(\geq 90\):
        return 'A'
    elif 70 \(\leq\) score < 90:
        return 'B'
    else:
        return 'C'

# Step 2: Convert scores to letter grades.
df['grade'] = df.score.apply(get_grade)

# Step 3: Count the number of students by grade.
count_df = df['grade'].value_counts()

# Step 4: Visualize in a bar chart.
count_df.plot(kind='bar')
```
A Problem that requires a multi-step solution:

[1] import pandas as pd
def df = pd.read_csv(scores.csv')

[2] Plot the number of students in each letter grade range (A: >=90, B: 70-90, C: <70).
>_ LLM Completion...

Decompositional step-by-step decoding

Step-by-step Prediction with Explanations: (Yin et al., 2023)

# Let's solve this problem step-by-step.
# Step 1: Define a function to convert scores to letter grades.
def get_grade(score):
    if score >= 90:
        return 'A'
    elif 70 <= score < 90:
        return 'B'
    else:
        return 'C'
# Step 2: Convert scores to letter grades.
df['grade'] = df.score.apply(get_grade)
# Step 3: Count the number of students by grade.
count_df = df['grade'].value_counts()
# Step 4: Visualize in a bar chart.
count_df.plot(kind='bar')

[Yin et al., 2023; Jiang et al., 2023; Zelikman et al., 2023]
Step-by-Step Decoding Improves Accuracy and Code Style
(prompting using notebook context and additional exemplars with step-by-step code solutions)

Larger gap for harder problems

Let's solve the problem step-by-step!
**Intent:** What is the distribution of student adaptivity level across each age group?

```python
# Step 1: Get the unique values of age
age_groups = students['Age'].unique()
# Step 2: Group students by their age groups and adaptivity level and
# get the number of students in each group
distribution = students.groupby(['Age', 'Adaptivity Level']).size()
# Step 3: Return the age groups as an index and adaptivity levels as columns
result = distribution.unstack(level=1, fill_value=0)
```

**Intent:** What are the five most common genres for shows released during the year having the highest percent increase in votes?

```python
# Step 1: Get the year that has the highest percent increase in votes.
max_year_pct_change = pct_change_votes.idxmax()
# Step 2: Get the genres for each shows in that year.
genres = drama[drama['Year of Release'] == max_year_pct_change]['Genre']
# Step 3: Count each genre
genre_counts = {}
for genre in genres:
    for genre in genre.split(', '):
        if genre not in genre_counts:
            genre_counts[genre] = 0
        genre_counts[genre] += 1
# Step 4: Return the most common genres
most_common_genres = pd.Series(genre_counts).sort_values(ascending=False)
most_common_genres[:5]
```
**Intent:** How many people have credit cards expiring in 2025?

### Step-by-step Prompting

**Step 1:** Create a new column with the last two digits of the year in expiration date column

```python
ecom['CC Exp Date_last'] = ecom['CC Exp Date'].str[-2:]
```

**Step 2:** Get the rows that expire in 2025

```python
ecom_2025 = ecom[ecom['CC Exp Date_last'] == '25']
```

**Step 3:** Return the count of rows that expire in 2025

```python
len(ecom_2025.index)
```

### Vanilla Prompting

**Step 1:** Get all the year in the expiration date

```python
exp_date_year = ecom['CC Exp Date'].apply(lambda x: x[3:])
```

**Step 2:** Get all the year in the expiration date as integer

```python
exp_date_year = exp_date_year.astype(int)
```

**Step 3:** Get all the year in the expiration date that are equal to 25

```python
exp_date_year = exp_date_year[exp_date_year == 25]
```

**Step 4:** Count the occurrences

```python
exp_date_year.value_counts()
```

---

[Step-by-Step Decoding Improves Solution Diversity] [Yin et al., 2023; Jiang et al., 2023]
Decoding Methods: Find consistency among candidate solutions

Write a function to count the lowercase letters in a string

```python
def count(string):
    cnt = 0
    for ch in string:
        cnt += ch.islower()
    return cnt
```

```python
def count(s):
    return len([c for c in s if c.islower()])
```

Sample multiple candidate code solutions

Synthesize input cases and collect execution results

Cluster solutions based on execution outputs

Return a solution whose execution results agree with most candidate solutions

Define a distribution over their “semantics” (execution behavior):

\[
p(>_{\{2,3\}} | \text{Output}) = p(\text{Solution}_1 | \text{Output}) + p(\text{Solution}_3 | \text{Output})
\]

\[
p(>_{\{3,4\}} | \text{Output}) = p(\text{Solution}_2 | \text{Output})
\]

Find whose execution behavior is the highest under

\[
p(>_{\text{Output}})
\]

Particularly effective with step-by-step decoding since it improves sample diversity (Yin et al., 2023)

[Shi et al., 2023; Wang et al., 2023]

(Example credit: talk by Freda Shi)
Decoding Methods: Reason with semantic consistency *without* execution

What is the most expensive phone model in each brand?

1. Sample multiple candidate code solutions

2. Use an LLM to select a most consistent answer

- Does not require execution
- Useful for open-ended problems where exact match of execution results is difficult
- Model can also leverage partially correct answers during reasoning

Based on the three code solutions, the most consistent answer is:

```python
idx = df.groupby('brand')['price'].idxmax()
expensive_models_df = df.loc[idx]
expensive_models_df[['brand', 'model', 'price']]
```

[Chen et al., 2023. *Universal Self-Consistency for Large Language Model Generation*]
Decoding Methods: Self-improvement by reasoning with/without execution

Write a function to find all words which are at least 4 characters long and end with “e”.

```
import re
def find_words(text):
    return re.findall(r"\w{3,}e", text)
```

Model first generates an initial code solution

1. If test fails, pass the test result to model as feedback

2. Model reasons with the execution of the program to identify potential bugs and suggest a fix.

- > _ execution feedback could be optional and LLMs could self-debug by reasoning about execution
- Self-repair performance is largely bottlenecked by the model’s reasoning ability to find the bug and propose a strategy to fix its prediction (Olausson et al., 2023).

To fix this, we can change the regex to only match full words instead of arbitrary prefixes. Here’s a fix of the code:

```
import re
def find_words(text):
    return re.findall(r"\w{3,}e\b", text)
```

[Chen et al., 2023. Teaching Large Language Models to Self-Debug]
Natural language to Code Generation: Agenda

(Supervised) Instruction Tuning

Modeling Code Context and Multi-turn Interaction

Decoding and Reasoning Methods (planning, consistency-based decoding, self-improvement)

Model Evaluation on Open-Domain Tasks
Write a function to find the k-th largest item in an array

Test Cases:
assert k_largest(arr=[5,7,3],k=2) == 5
assert k_largest(arr=[4,2,3,1],k=3) == 2
assert k_largest(arr=[15, 8],k=1) == 15

I have a 1d numpy array a = np.array([1,0,3]). Encode this as a 2D one-hot array
np.array([[0,1,0,0],[1,0,0,0],[0,0,0,1]])

Unit Tests:
import numpy as np
da = np.array([1,2,0])
assert answer(a) == np.array([[0,1,0],[0,0,1],[1,0,0]])

Show the time of the day and the price for each airline

Acceptable Answers:

<table>
<thead>
<tr>
<th>Airline</th>
<th>Time</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>United</td>
<td>Noon</td>
<td>$450</td>
</tr>
</tbody>
</table>

- Evaluating LLMs requires high-quality annotated natural language problems with test cases or reference answers
- Creating annotated NL2Code problems costs $ and person
- Therefore, datasets are limited in domain coverage and size

Can we leverage high-quality code with tests in the wild to evaluate the natural language to code skills of LLMs?
def unique_in_window(iterable, n):
    """Yield the items from iterable that haven't been seen recently. n is the size of the lookback window.""
    window = deque(maxlen=n)
    counts = defaultdict(int)
    use_key = key is not None
    for item in iterable:
        if len(window) == n:
            to_discard = window[0]
            if counts[to_discard] == 1:
                del counts[to_discard]
            else:
                counts[to_discard] -= 1
        if item not in counts:
            yield item
        counts[item] += 1
        window.append(item)

---

def unique_in_window(iterable, n):
    """Yield the items from iterable that haven't been seen recently. n is the size of the lookback window."
    window = deque(maxlen=n)
    counts = defaultdict(int)
    use_key = key
    is not None
    for item in iterable:
        if len(window) == n:
            to_discard = window[0]
            if counts[to_discard] == 1:
                del counts[to_discard]
            else:
                counts[to_discard] -= 1
            if item not in counts:
                yield item
        counts[item] += 1
        window.append(item)
        if len(window) == n:
            to_discard = window.popleft()
            if counts[to_discard] == 1:
                del counts[to_discard]
            else:
                counts[to_discard] -= 1
    Test Pass

"""to discard" takes the first value in "window". If the count for that value in "counts" is "1", we remove the value, otherwise we decrement the count.

if the window is at capacity, discard the oldest element, and update counts so that this element is only considered if it is seen again.

"to discard" equals "window[(len(window) - 1) % len(window)]"

Round Trip Correctness Score $RTC_{pass} = 1/3$

[Allamanis*, Panthaplackel*, Yin*, 2024]
Round-trip correctness scores correlate well with official benchmark metrics without using annotated natural language instructions [Allamanis*, Panthaplackel*, Yin*, 2024]
Round-trip correctness (RTC) scores vary widely across projects/domains, suggesting that narrow-domain benchmarks cannot capture the LLM’s skills across multiple domains.

[Allamanis*, Panthaplackel*, Yin*, 2024]
Natural Language to Code Generation: Summary

(Supervised) Instruction Tuning

Modeling Code Context and Multi-turn Interaction

Decoding and Reasoning Methods (planning, consistency-based decoding, self-improvement)

Model Evaluation on Open-Domain Tasks