Parametric Template Distillation for API Summarization

Naman Jain Aditya Parameswaran Koushik Sen Ion Stoica

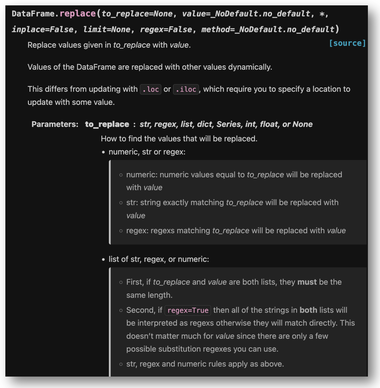
EPIC DATA lab UC Berkeley

Motivation

Data Science APIs are complex

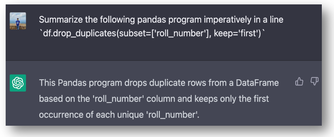
Data Science APIs like Pandas, SkLearn have a huge number of functions and arguments

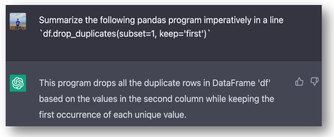
We are looking into using program summarization to aid developers in understanding such programs



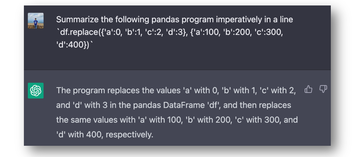
Issues in Language Models for Program Summarization

Robustness to perturbations in programs





Compositional Generation



They can generate incorrect answers

Controllability

These systems provide very little control to end users as to how templates should look

Parametric Templates – Using API knowledge for summarization

Defining Parametric Templates

Given an API function n along with signature $t_1, t_2, \ldots, t_n$

Parametric Templates are defined by a sequence of
- words (w_i)
- holes function (mapping from arguments to sequence of words)

Learning Parametric Templates

Writing parametric templates is not easy!

requires domain expertise	templates are fuzzy and hard to manually annotate	numerous functions and arguments!

We learn templates from corpus of API snippets and summaries
- dynamic programming on spans of texts
- bottom-up program synthesis
- word and phrase-based similarity

Parametric Templates Examples

```
df[df['score'].isin(range(5,10))]
```

Select the rows where value in column score lie in the integers between 5 and 10 (exclusive)

- Template[subscript, [caller - df, arg - expr]]
  - Select the rows VAR1
  - VAR1 = Summary(`df['score'].isin(range(5,10))`)
- Template[isin, [caller - `df['score']`, value - range(5,10)]]
  - where values in column score lie in VAR2, where
  - VAR1 = Summary(`range(5,10)`)
- Template[range, [start - int, end - int]]
  - the integers between start and end(exclusive)

Learned Templates

```
df.replace({'country': {'Germany':'GER', 'France':'FRA'}})
```

Replace the values F^1 in column F^2 with F^3 respectively
F^1(P) = 'Germany' and 'France'
F^2(P) = 'country'
F^3(P) = 'GER' and 'FRA'

```
df.replace({'a':1, 'b':2, 'c':3}, {'a':100, 'b':200, 'c':300}
```

Replace the values F^1 with F^3 respectively
F^1(P) = 1 in 'a', 2 in 'b', 3 in 'c'
F^2(P) = 100, 200, 300

```
df.dropna(subset=['score1', 'score2', 'score3'], thresh=2)
```

Drop the rows in df having atleast F^1 nans in the F^2 columns
F^1(P) = 2
F^2(P) = 'score1', 'score2', and 'score3'