StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow

Ziyu Yao

The Ohio State University

The Web Conference 2018, May 25th

In collaboration with Prof. Daniel S. Weld (UW), Dr. Wei-Peng Chen (Fujitsu Lab), Prof. Huan Sun (OSU).

THE OHIO STATE UNIVERSITY

UNIVERSITY of WASHINGTON

FUJITSU

shaping tomorrow with you

Mapping between natural language and programming language

Question

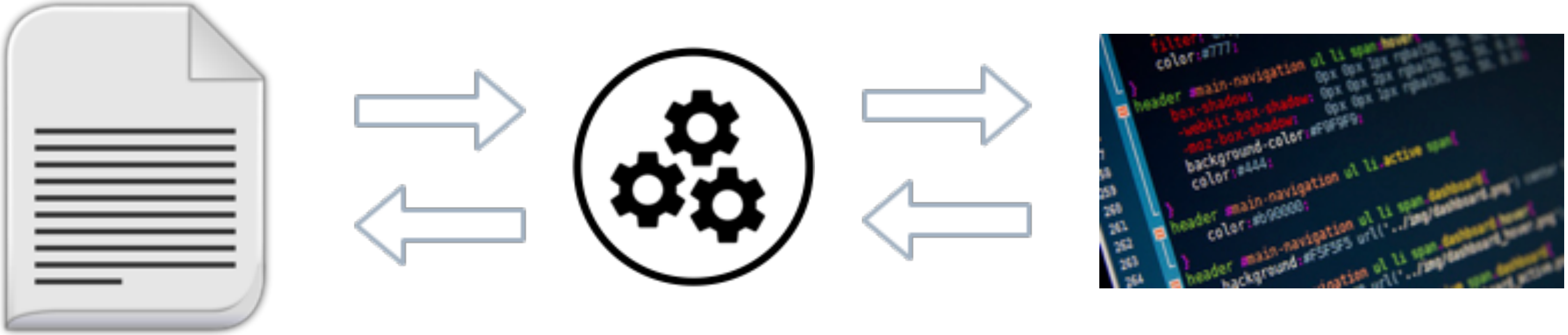
"how to clone or copy a python list?"

↔

Code

```
“new_list =
copy.copy(old_list)”
```

e.g., automated code search/annotation/generation.



Challenges

- Lack of large-scale datasets for model development.
  i.e., pairs of <natural language question, code snippet>

Challenges

- Lack of large-scale datasets for model development.

  i.e., pairs of <natural language question, code snippet>

- And datasets are important:

IMAGENET
SQuAD
The Stanford Question Answering Dataset
Microsoft COCO
Common Objects in Context

This work: StaQC

StaQC: the largest dataset to date of ~148K Python and ~120K SQL “how-to-do-it”* Question-Code pairs!

Example:

Question

“how to clone or copy a python list?”

↔

Code

```
“new_list =
copy.copy(old_list)”
```

“how-to-do-it” questions [Souza et al., 2014; Defim et al., 2016]: *the questioner provides a scenario and asks how to implement it.*

This work: StaQC

StaQC: the largest dataset to date of ~148K Python and ~120K SQL "how-to-do-it"* Question-Code pairs!

Continuously growing in *size* and *diversity*

stackoverflow

Diversity of StaQC

- Containing multiple code solutions to the same question.
  - Question “How to limit a number to be within a specified range?”
  - **4** code solutions in StaQC:

```
def clamp(n, minn, maxn):
  return max(min(maxn, n),
minn)
```

(1)

```
clamp = lambda n, minn,
maxn: max(min(maxn, n),
minn)
```

(2)

```
def clamp(n, minn, maxn):
  if n < minn:
    return minn
  elif n > maxn:
    return maxn
  else:
    return n
```

(3)

```
n = minn if n < minn else maxn if n > maxn else n
```

(4)

Diversity of StaQC

- Containing different questions asking for semantically similar code solutions.

Question A: "*How to find a gap in range in SQL*"

```
SELECT id + 1
FROM test mo
WHERE NOT EXISTS
(
  SELECT NULL
  FROM test mi
  WHERE mi.id = mo.id + 1
  ) and mo.id> 100
ORDER BY
        id
LIMIT 1
```

Question B: "*How do I find a “gap” in running counter with SQL?* "

```
SELECT id + 1
FROM mytable mo
WHERE NOT EXISTS
(
  SELECT NULL
  FROM mytable mi
  WHERE mi.id = mo.id + 1
  )
ORDER BY
        id
LIMIT 1
```

Diversity of StaQC

- Containing different questions asking for semantically sim

Critical for Model Robustness:

1. Natural language variation.
2. Different implementations to do the same thing in programming language.

```
   ) and mo.id> 100
ORDER BY
        id
LIMIT 1
```

```
   )
ORDER BY
        id
LIMIT 1
```

This work: StaQC

StaQC: the largest dataset to date of ~148K Python and ~120K SQL "how-to-do-it"* Question-Code pairs!

Continuously growing in *size* and *diversity*

stackoverflow

A better source for constructing models mapping between NL and PL

This work: StaQC

StaQC: the largest dataset to date of ~148K Python and ~120K SQL “how-to-do-it”* Question-Code pairs!

Continuously growing in *size* and *diversity*

stackoverflow

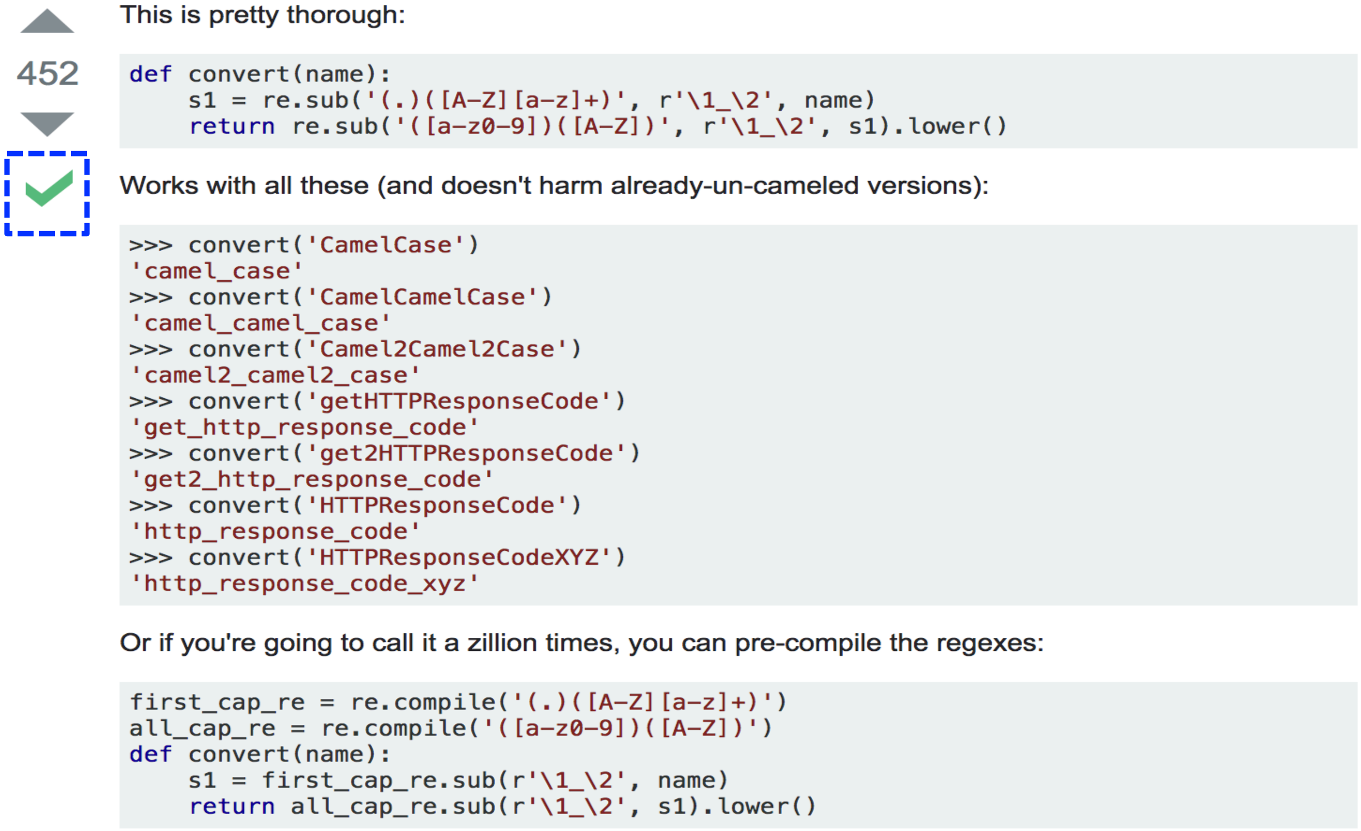
A better source for constructing models mapping between NL and PL

Example

stackoverflow

Question Elegant Python function to convert CamelCase to snake_case?

Accepted answer post



This is pretty thorough:

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Works with all these (and doesn't harm already-un-cameled versions):

```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Or if you're going to call it a zillion times, you can pre-compile the regexes:

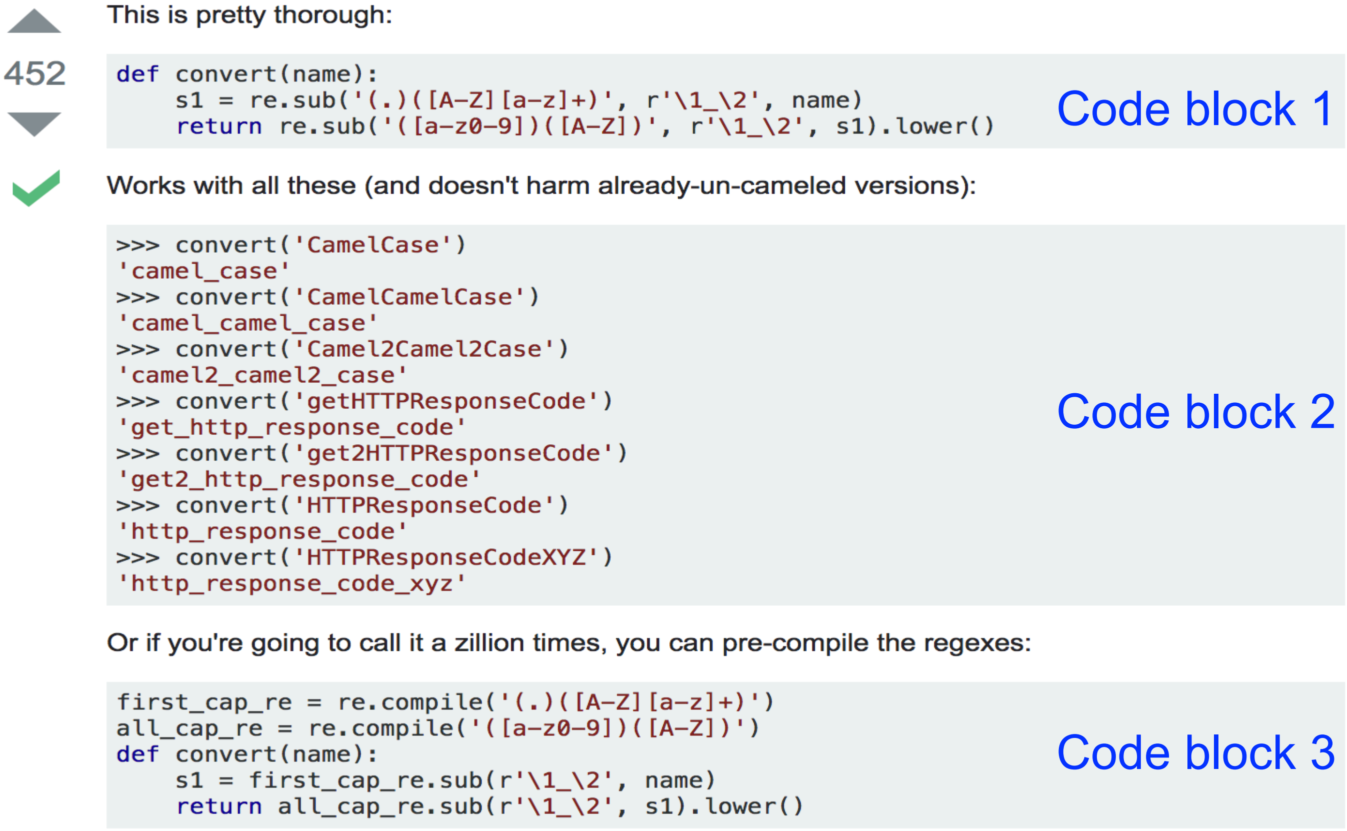
```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

Example

stackoverflow

Question Elegant Python function to convert CamelCase to snake_case?

Accepted answer post



452

This is pretty thorough:

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Code block 1

Works with all these (and doesn't harm already-un-cameled versions):

```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Code block 2

Or if you're going to call it a zillion times, you can pre-compile the regexes:

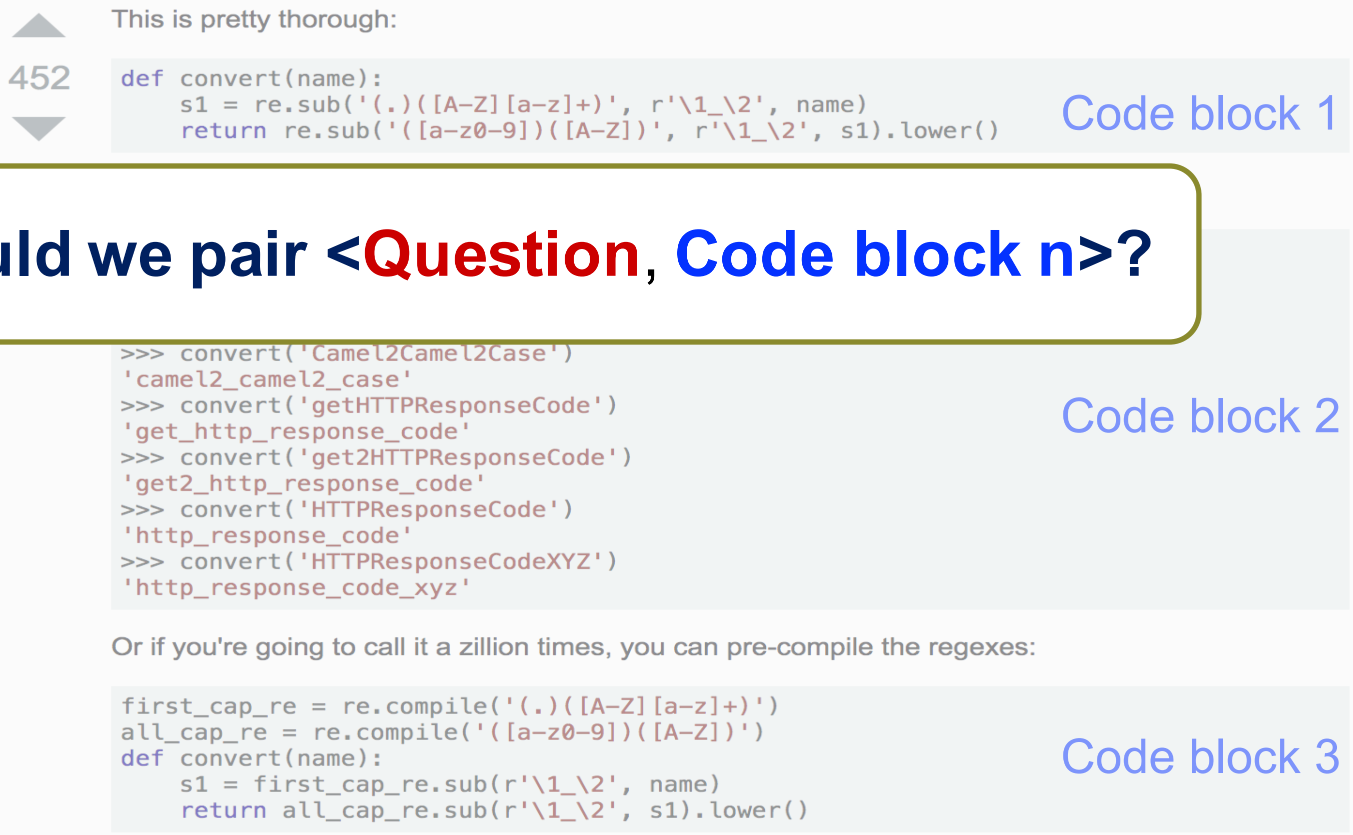
```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

Code block 3

Example

Question Elegant Python function to convert CamelCase to snake_case?

Accepted answer post



452

This is pretty thorough:

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Code block 1

Should we pair <Question, Code block n>?

```
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Code block 2

Or if you're going to call it a zillion times, you can pre-compile the regexes:

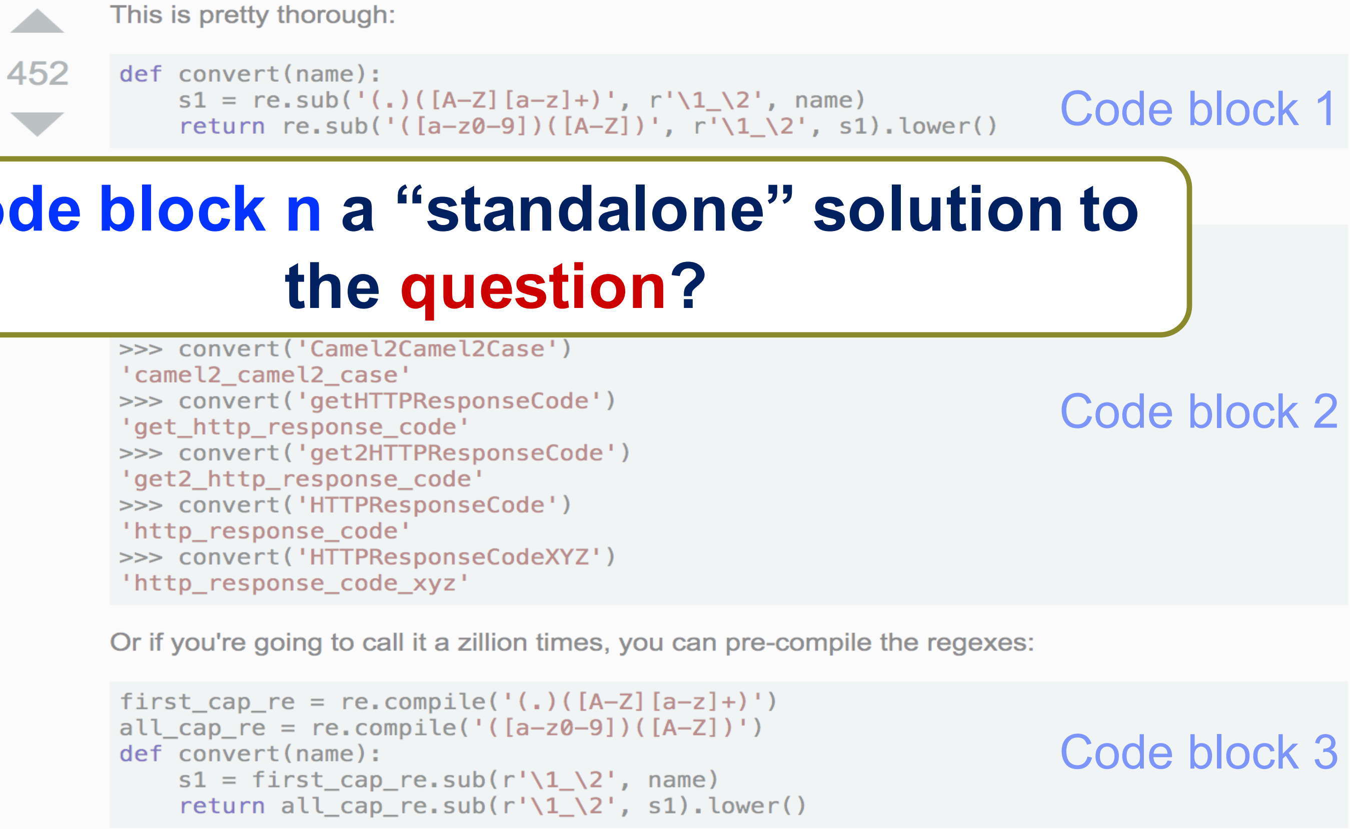
```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

Code block 3

Example

Question Elegant Python function to convert CamelCase to snake_case?

Accepted answer post



This is pretty thorough:

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Code block 1

Is Code block n a “standalone” solution to the question?

```
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Code block 2

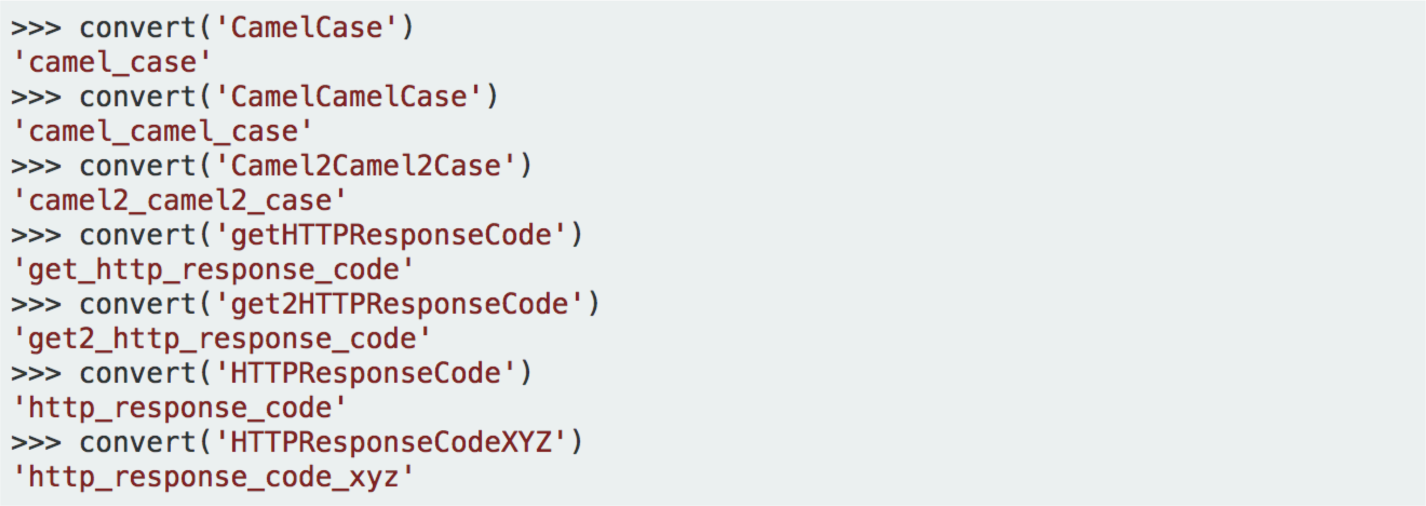
Or if you're going to call it a zillion times, you can pre-compile the regexes:

```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

Code block 3

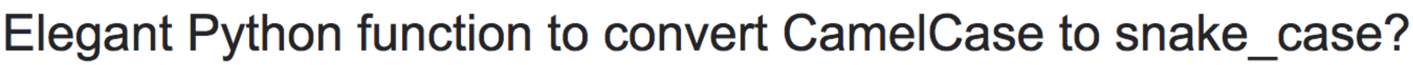
“Standalone” code solution

By looking at Code block n,



```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

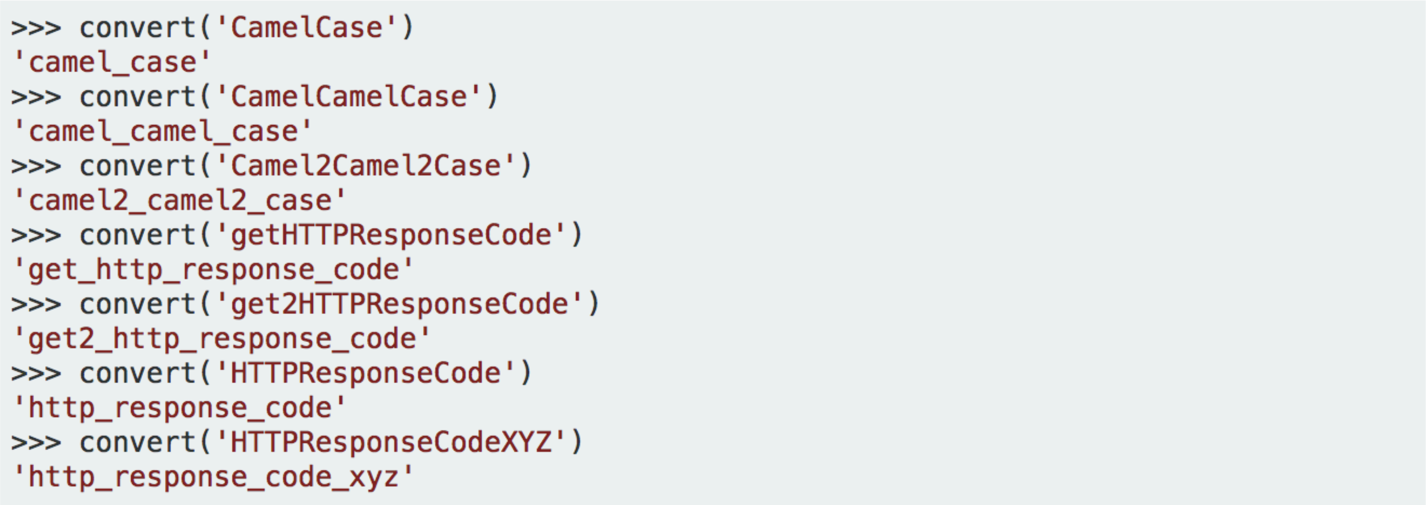
can you solve the problem:



Elegant Python function to convert CamelCase to snake_case?

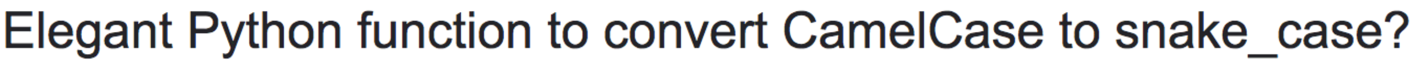
“Standalone” code solution

By looking at Code block n,



```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

can you solve the problem:



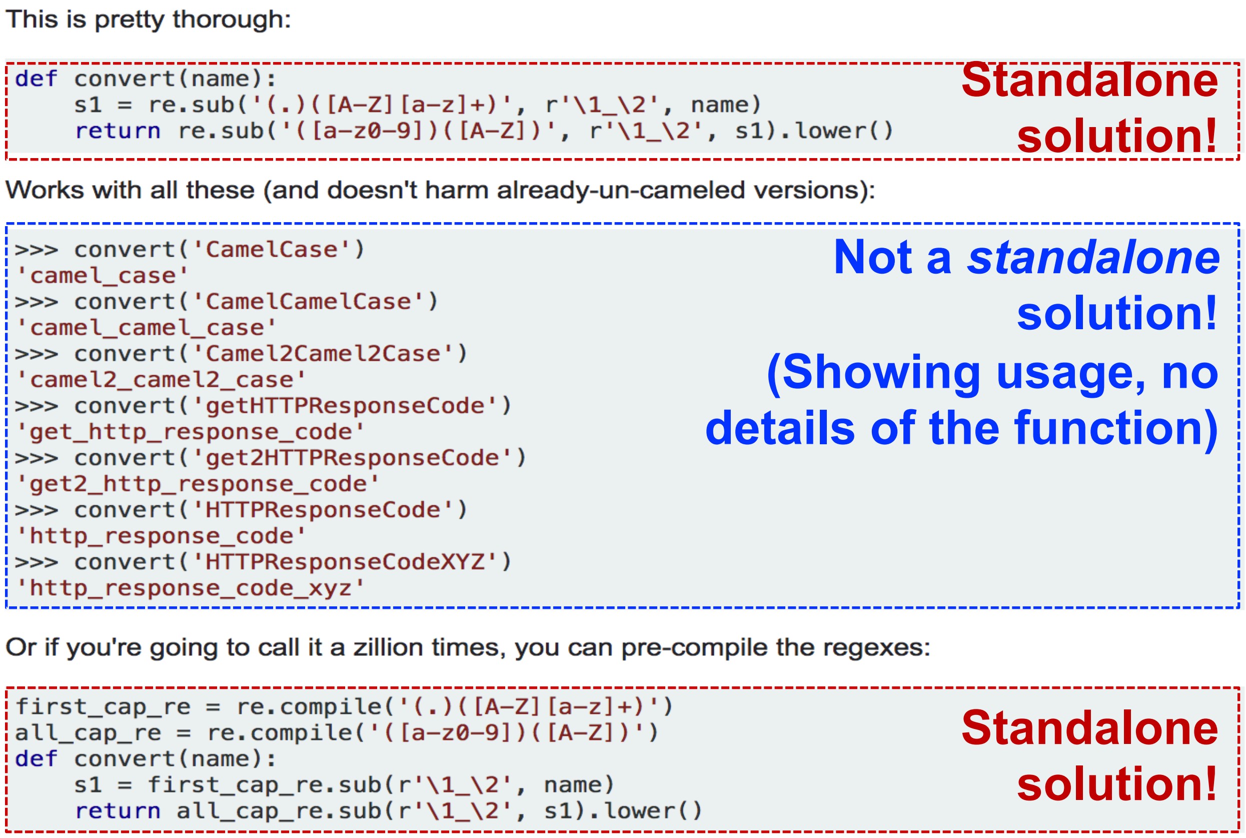
Elegant Python function to convert CamelCase to snake_case?

No! (it shows the usage, but with no details of the function)

Example: Question-Code pairs

Question Elegant Python function to convert CamelCase to snake_case?

Accepted answer post



This is pretty thorough:

☑

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Standalone solution!

Works with all these (and doesn't harm already-un-cameled versions):

✖

```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Not a *standalone* solution!

(Showing usage, no details of the function)

Or if you're going to call it a zillion times, you can pre-compile the regexes:

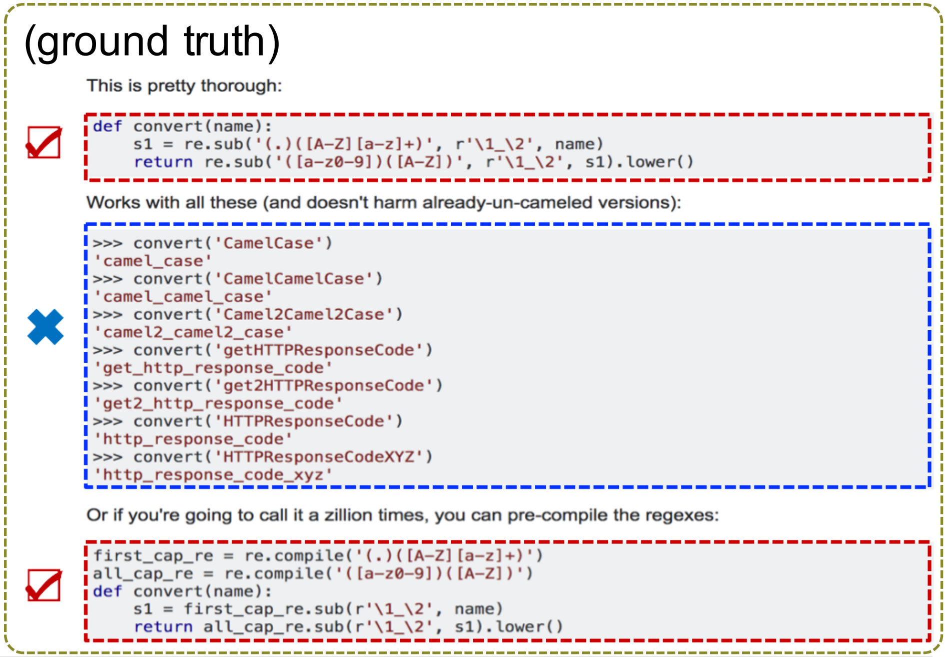
☑

```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

Standalone solution!

Previous methods: heuristics based

- “**Select All**”: Taking *all* code snippets in the answer post as code solutions. [Allamanis et al., 2015][Zilberstein and Yahav, 2016]
  - Low precision



(ground truth)

This is pretty thorough:

☑

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Works with all these (and doesn't harm already-un-cameled versions):

✖

```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Or if you're going to call it a zillion times, you can pre-compile the regexes:

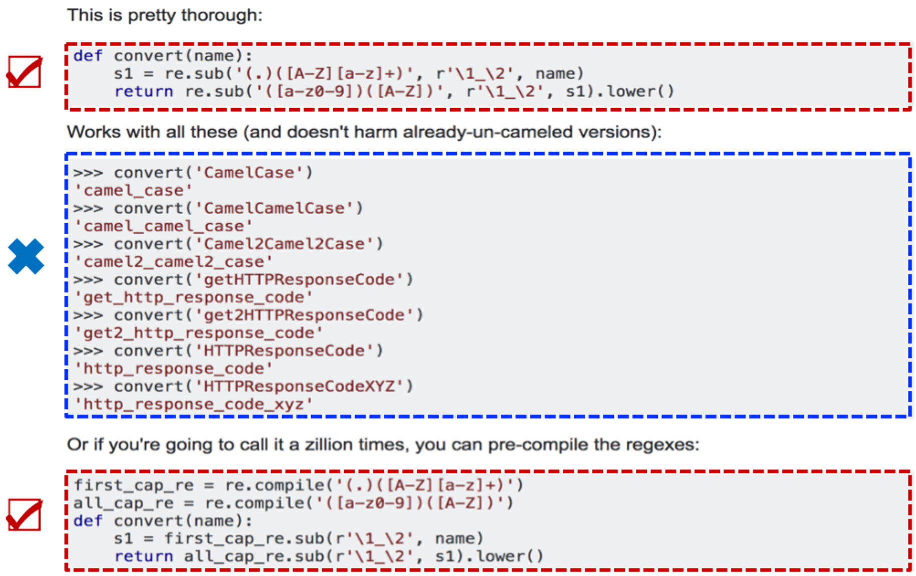
☑

```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

Previous methods: heuristics based

- “**Select First**”: Taking only the *<u>first</u>* code snippet in the answer post as a code solution, or considering only answer posts containing exactly one code snippet.
  [Iyer et al., 2016]
  - Low recall

(ground truth)



This is pretty thorough:

☑

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Works with all these (and doesn't harm already-un-cameled versions):

✖

```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Or if you're going to call it a zillion times, you can pre-compile the regexes:

☑

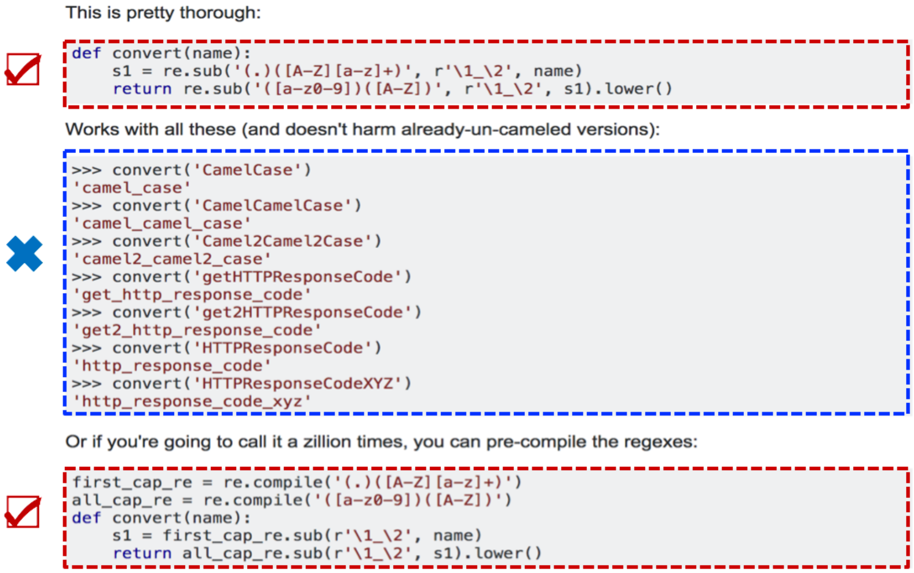
```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

Our solution: A systematic framework

- Binary classification formulation:

  Input: A question on Stack Overflow and its accepted answer post with multiple code snippets

  Output: A binary label for each code snippet on whether it is a standalone solution to the question



This is pretty thorough:

☑

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Works with all these (and doesn't harm already-un-cameled versions):

✖

```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

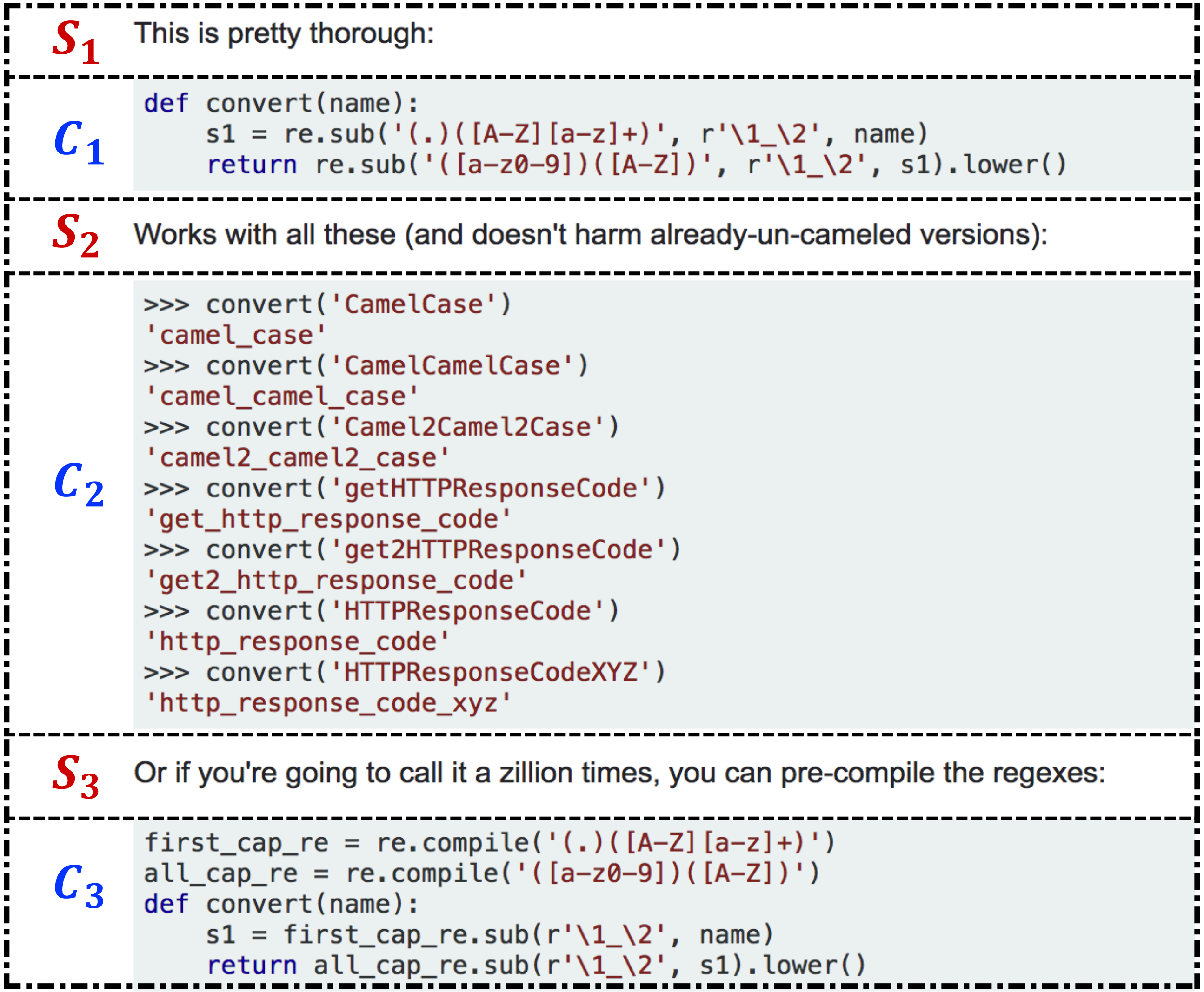
Or if you're going to call it a zillion times, you can pre-compile the regexes:

☑

```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

A bi-view formulation

Interleaving text and code blocks



S_1 This is pretty thorough:

C_1

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

S_2 Works with all these (and doesn't harm already-un-cameled versions):

C_2

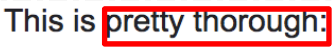
```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

S_3 Or if you're going to call it a zillion times, you can pre-compile the regexes:

C_3

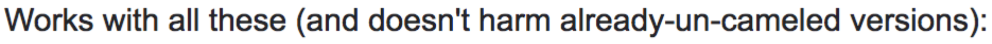
```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

Text-based view: contextual hints

S_1 

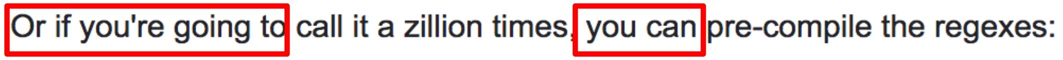
This is pretty thorough:

C_1 → more likely to be a code solution

S_2 

Works with all these (and doesn't harm already-un-cameled versions):

C_2

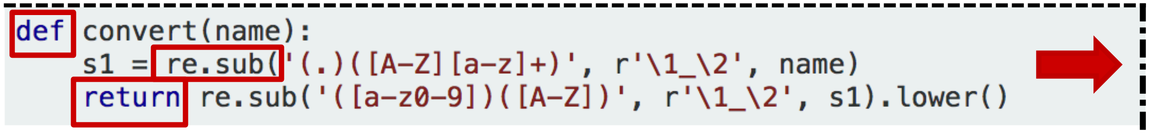
S_3 

Or if you're going to call it a zillion times, you can pre-compile the regexes:

C_3 → more likely to be a code solution

Code-based view: semantics of code content

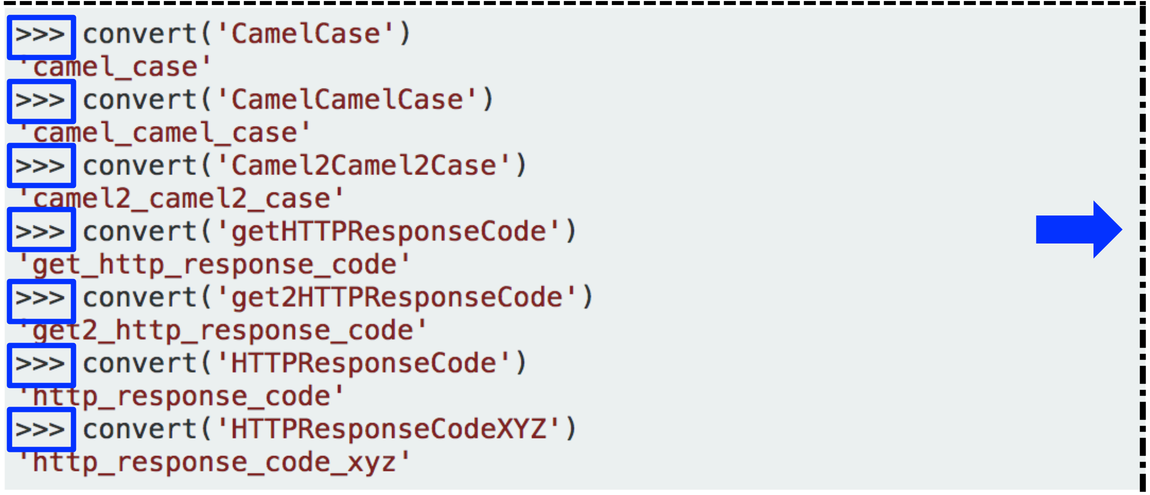
C_1



```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

more likely to be a solution

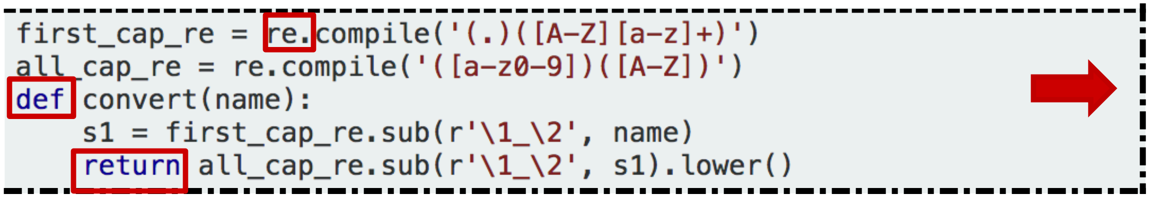
C_2



```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

possibly a usage demo

C_3



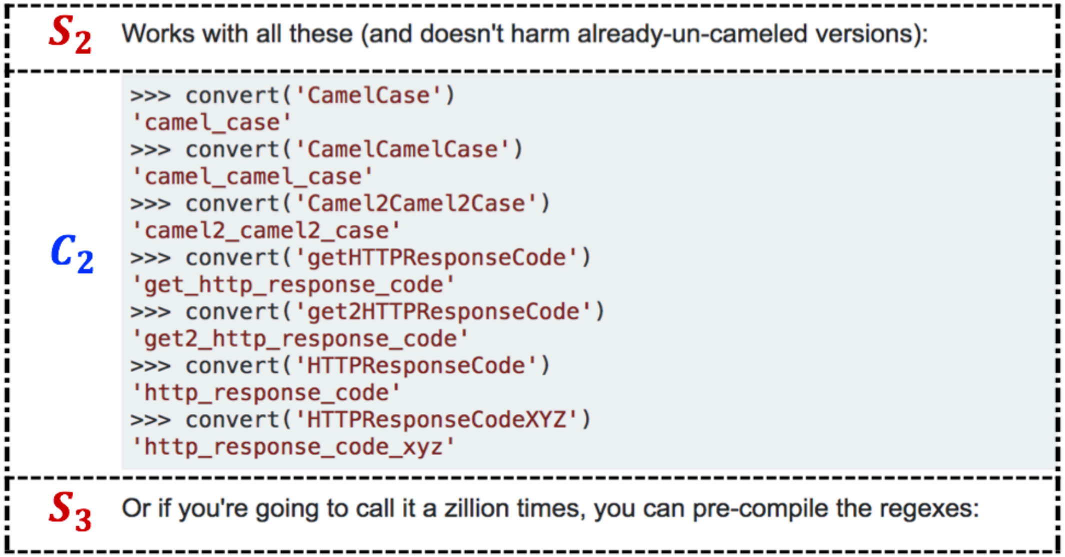
```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

more likely to be a solution

Formulation for each code snippet

Predict a “solution or not” label for a code snippet (here C_2) based on:

1. **Textual context** (text view): $\boldsymbol{S_2, S_3}$.
2. **Code content** (code view): $\boldsymbol{C_2}$.



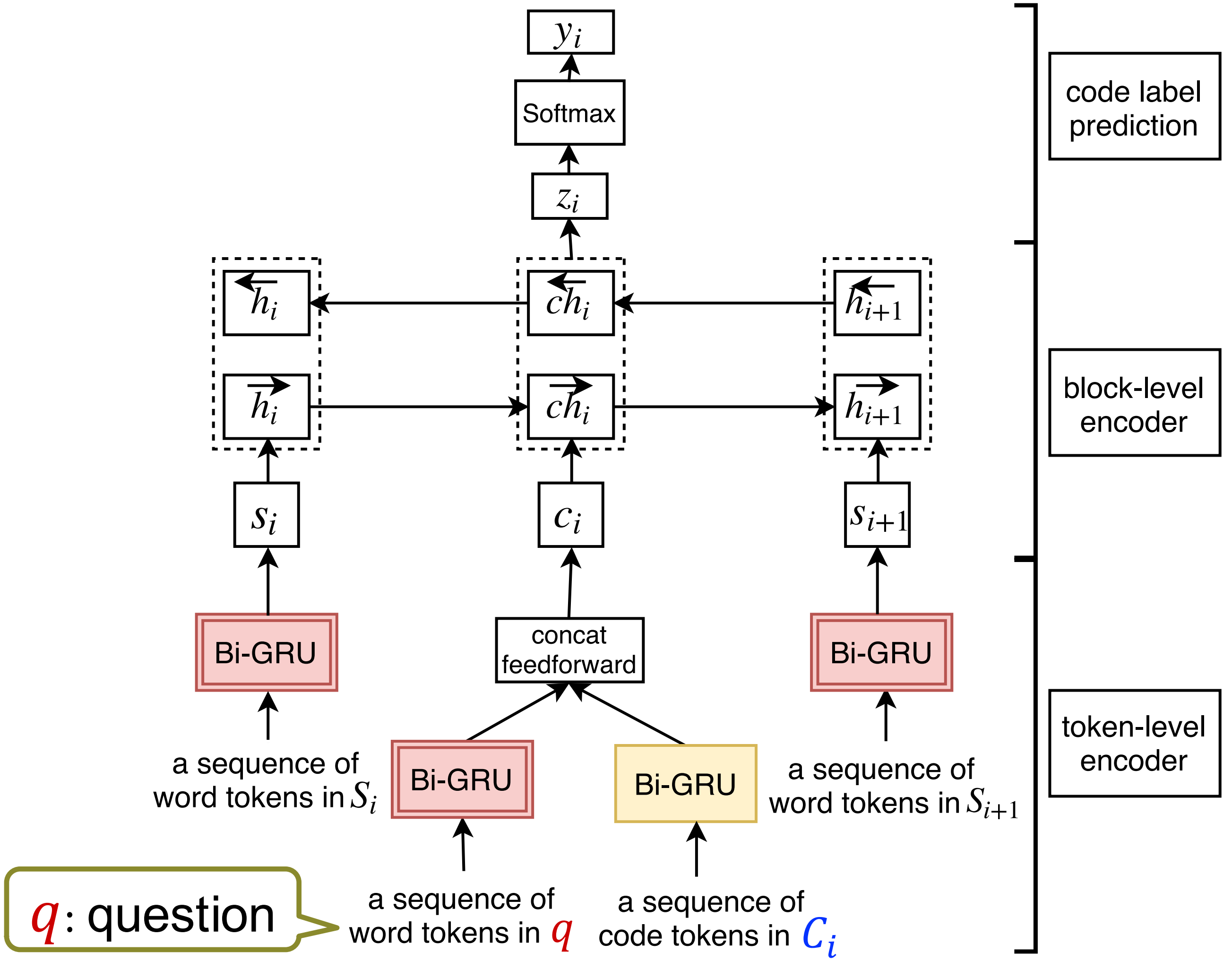
$\boldsymbol{S_2}$ Works with all these (and doesn't harm already-un-cameled versions):

$\boldsymbol{C_2}$

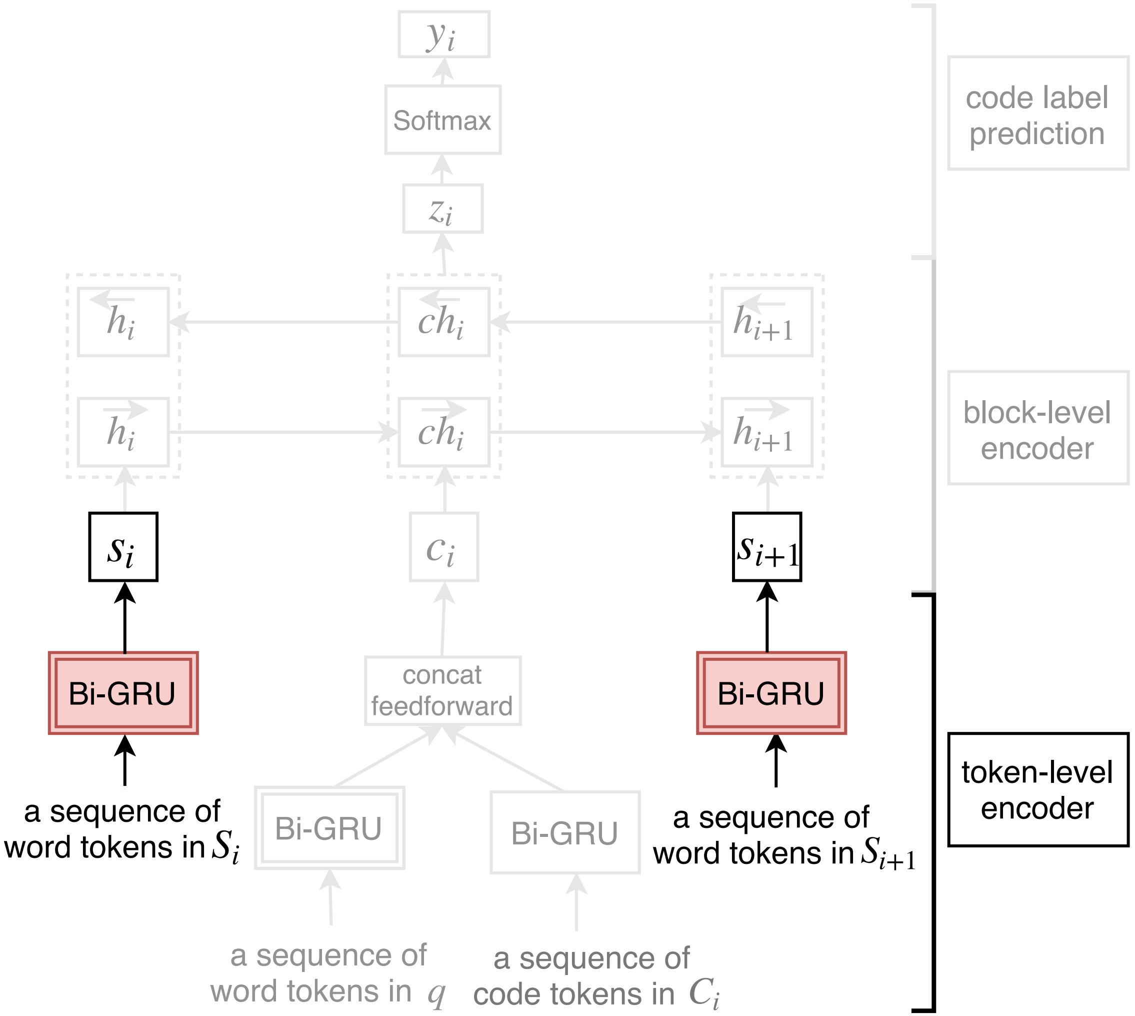
```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

$\boldsymbol{S_3}$ Or if you're going to call it a zillion times, you can pre-compile the regexes:

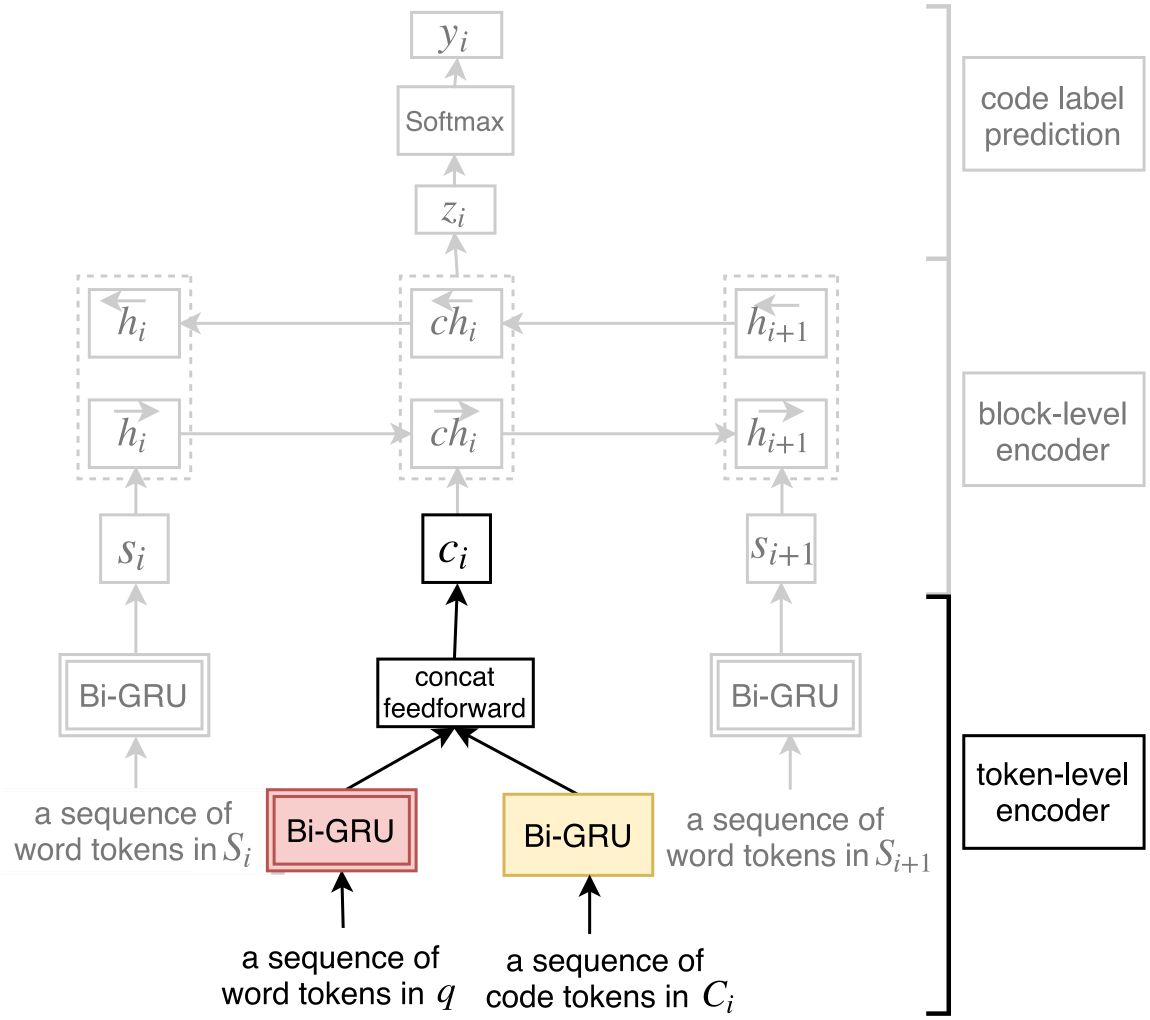
Bi-View Hierarchical Neural Network (BiV-HNN)



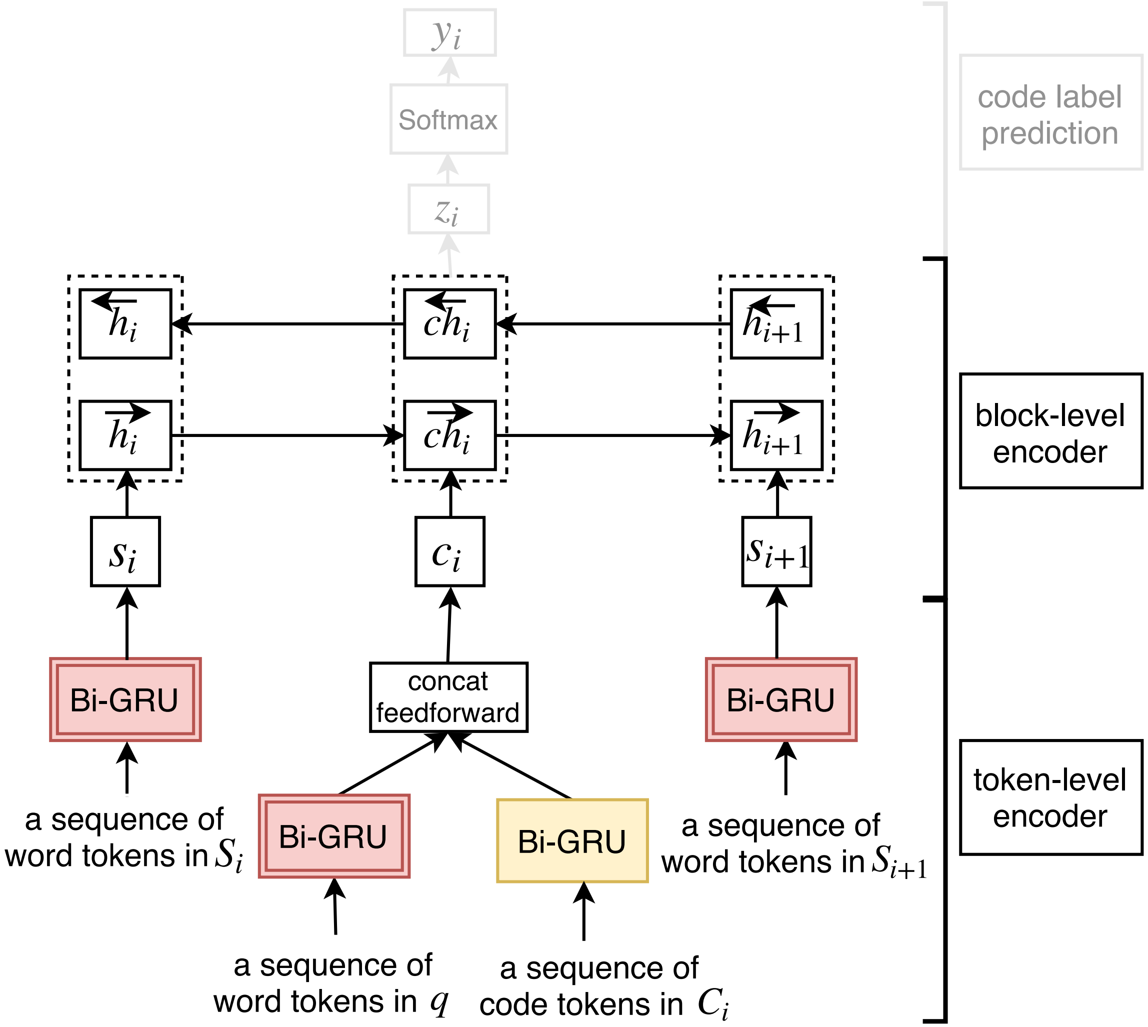
Token-level encoder for text blocks



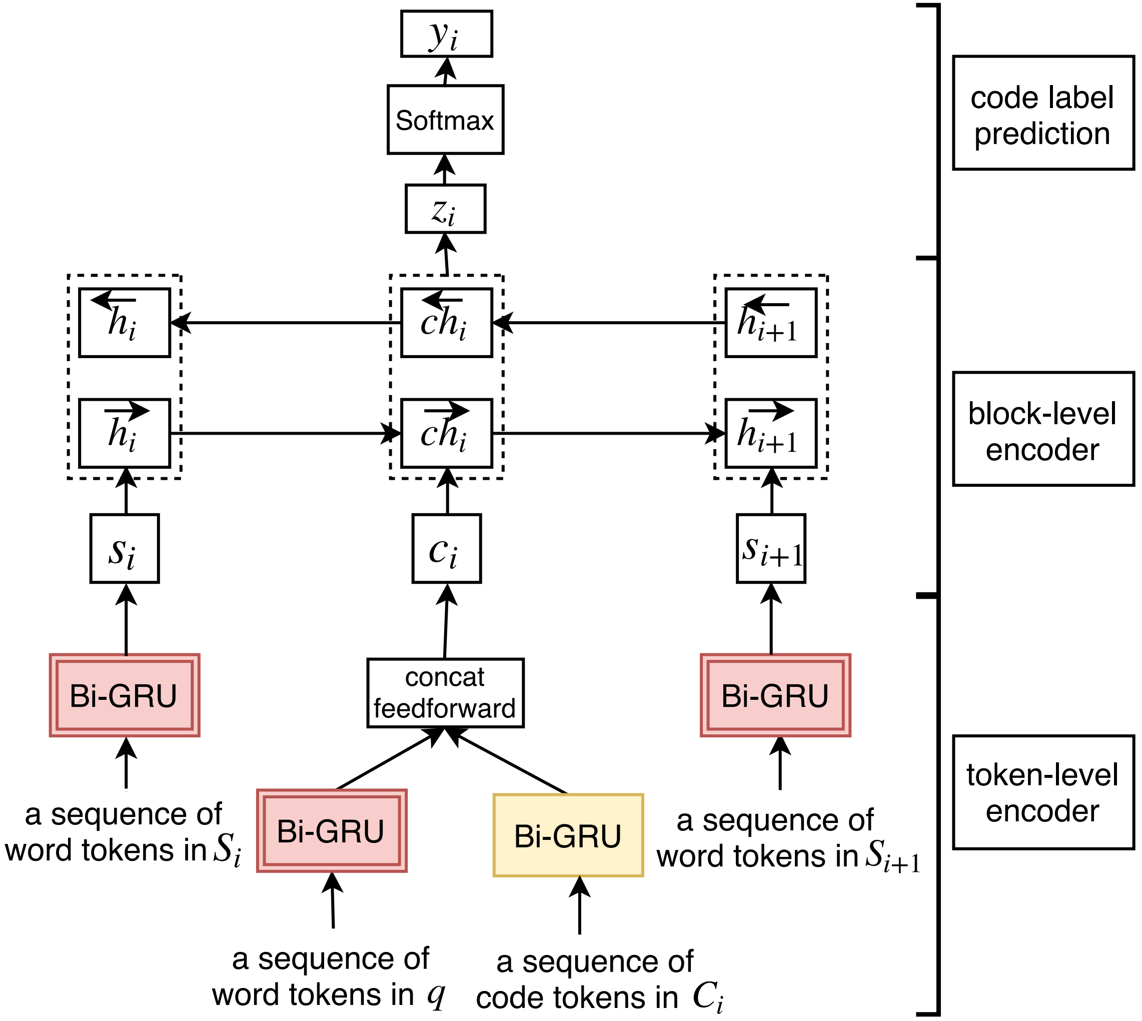
Token-level encoder for code blocks



Block-level encoder



Code label prediction



Experimental Setup

- Manually annotating “solution or not” label on code snippets in one answer post.
  - Python and SQL domain.
  - Four undergraduates with substantial Cohen’s kappa agreement.
- Training/validation/test split: 60% - 20% -20%.

	Python	SQL
# of Question-Code pairs	4,884	3,637
% of positive Question-Code pairs	44%	57%

Main Results

- Heuristic methods: Select-First, Select-All.
- Feature engineering based methods:
  - Logistic Regression (LR), Support Vector Machine (SVM).
  - Features: text-based (uni-/bi-grams, the connectives, etc) and code-based (code tokens, etc).

	Python	SQL
Select-First	0.607	0.613
Select-All	0.642	0.737
LR	0.766	0.846
SVM	0.753	0.850
BiV-HNN	**0.841**	**0.888**

(comparison on F1)

Research questions for understanding BiV-HNN

Q1: text view, code view, or bi-view?

Q2: hierarchical structure or flat structure?

Q3: block-level encoder: sequential or feedforward?

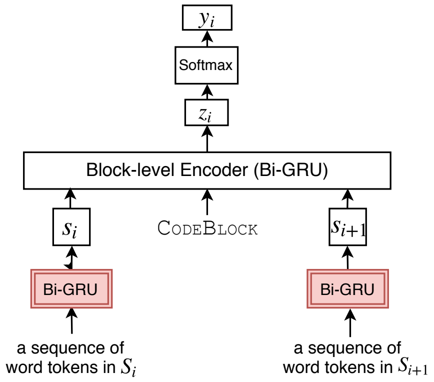
Research questions for understanding BiV-HNN

Q1: text view, code view, or bi-view?

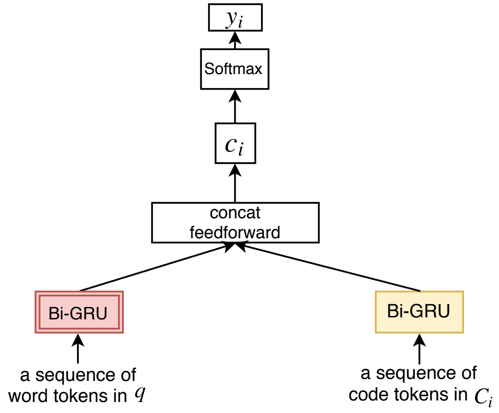
Q2: hierarchical structure or flat structure?

Q3: block-level encoder: sequential or feedforward?

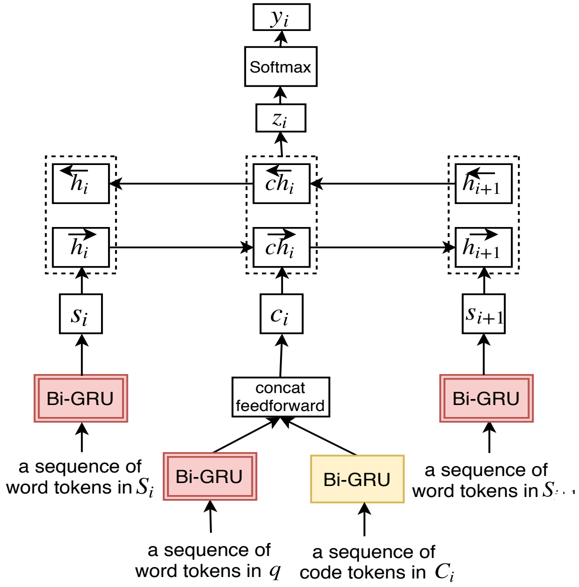
Q1: text view, code view, or bi-view?


y_i
Softmax
z_i
Block-level Encoder (Bi-GRU)
s_i
CodeBlock
s_{i+1}
Bi-GRU
Bi-GRU
a sequence of word tokens in S_i
a sequence of word tokens in S_{i+1}

Text-HNN
(text view)


y_i
Softmax
c_i
concat feedforward
Bi-GRU
Bi-GRU
a sequence of word tokens in q
a sequence of code tokens in C_i

Code-HNN
(code view)


y_i
Softmax
z_i
$\overleftarrow{h_i}$
$\overleftarrow{ch_i}$
$\overleftarrow{h_{i+1}}$
$\overrightarrow{h_i}$
$\overrightarrow{ch_i}$
$\overrightarrow{h_{i+1}}$
s_i
c_i
s_{i+1}
Bi-GRU
concat feedforward
Bi-GRU
a sequence of word tokens in S_i
Bi-GRU
Bi-GRU
a sequence of word tokens in S_{i+1}
a sequence of word tokens in q
a sequence of code tokens in C_i

BiV-HNN
(bi-view)

	Python	SQL
Text-HNN	0.771	0.840
Code-HNN	0.812	0.851
BiV-HNN	**0.841**	**0.888**

(comparison on F1)

Model combination

- `Text-HNN`, `Code-HNN` and `BiV-HNN` are observed complementary to each other.
  - On Python validation set, 60%~70% of mistakes made by one model can be corrected by the other two models.

Model combination

- `Text-HNN`, `Code-HNN` and `BiV-HNN` are observed complementary to each other.
  - On Python validation set, 60%~70% of mistakes made by one model can be corrected by the other two models.
- **Model combination**: the label of a code snippet is predicted only when the three models agree on it.

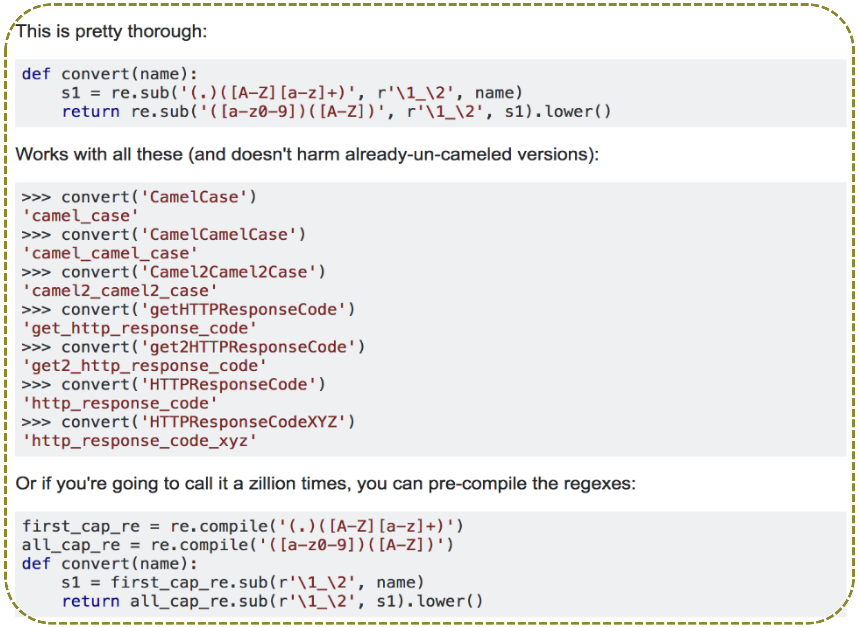
Model combination

- `Text-HNN`, `Code-HNN` and `BiV-HNN` are observed complementary to each other.
  - On Python validation set, 60%~70% of mistakes made by one model can be corrected by the other two models.

- **Model combination**: the label of a code snippet is predicted only when the three models agree on it.

- Model combination on testing set:
  - Python: ~70% of code snippets are labeled with **0.92** F1.
  - SQL: ~80% of code snippets are labeled with **0.94** F1.

Systematically mined StaQC



This is pretty thorough:

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Works with all these (and doesn't harm already-un-cameled versions):

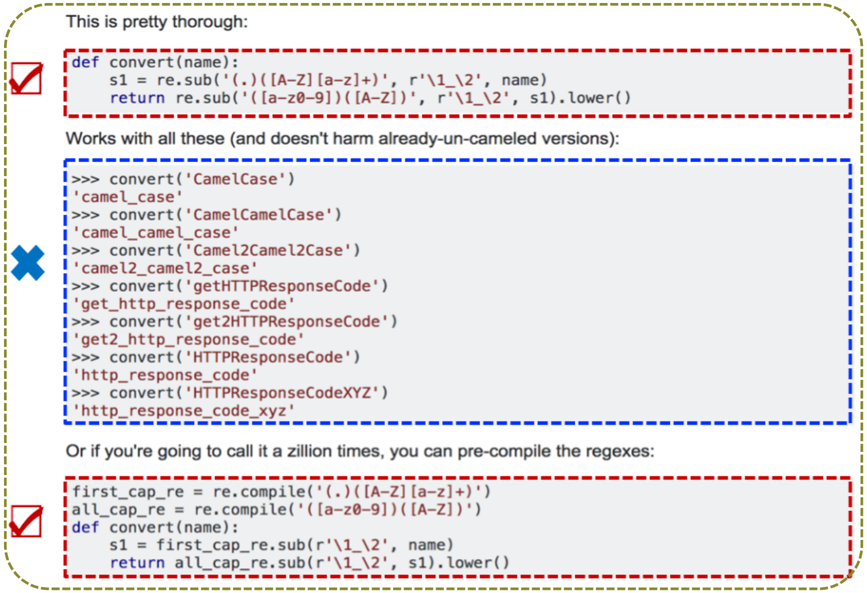
```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Or if you're going to call it a zillion times, you can pre-compile the regexes:

```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

New or unannotated post

Model combination



This is pretty thorough:

✓

```
def convert(name):
    s1 = re.sub('(.)([A-Z][a-z]+)', r'\1_\2', name)
    return re.sub('([a-z0-9])([A-Z])', r'\1_\2', s1).lower()
```

Works with all these (and doesn't harm already-un-cameled versions):

✗

```
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('Camel2Camel2Case')
'camel2_camel2_case'
>>> convert('getHTTPResponseCode')
'get_http_response_code'
>>> convert('get2HTTPResponseCode')
'get2_http_response_code'
>>> convert('HTTPResponseCode')
'http_response_code'
>>> convert('HTTPResponseCodeXYZ')
'http_response_code_xyz'
```

Or if you're going to call it a zillion times, you can pre-compile the regexes:

✓

```
first_cap_re = re.compile('(.)([A-Z][a-z]+)')
all_cap_re = re.compile('([a-z0-9])([A-Z])')
def convert(name):
    s1 = first_cap_re.sub(r'\1_\2', name)
    return all_cap_re.sub(r'\1_\2', s1).lower()
```

StaQC: Systematically mined Question-Code pairs

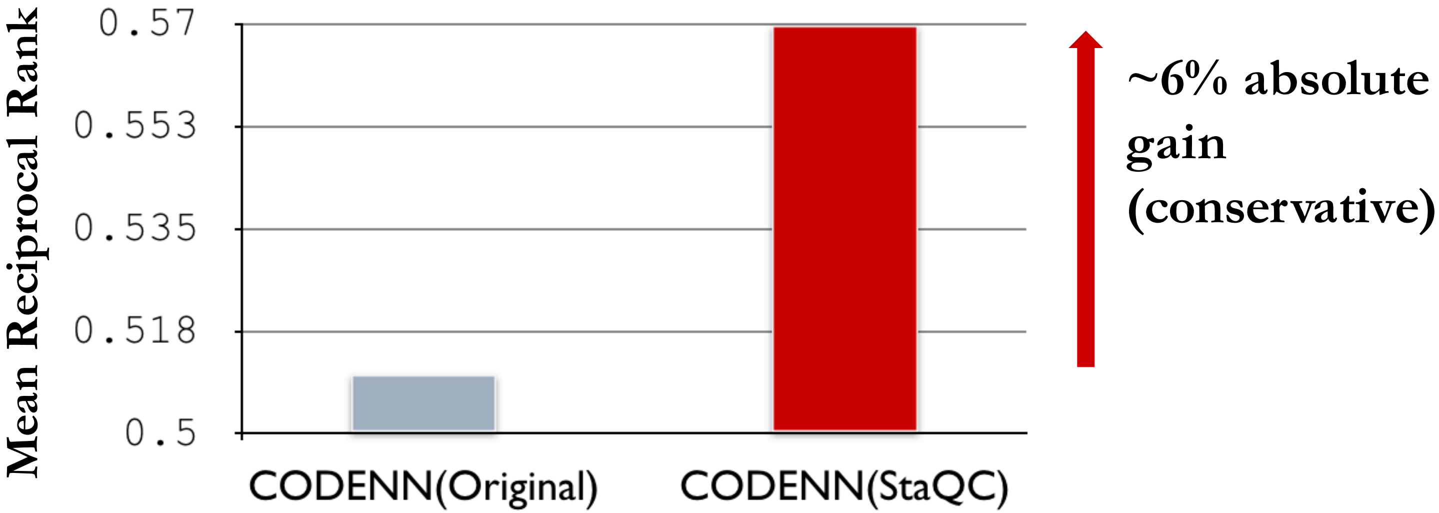
StaQC: the largest dataset to date of ~148K Python and ~120K SQL “how-to-do-it”* Question-Code pairs!

Continuously growing in *size* and *diversity*

A better source for constructing models mapping between natural language and programming language

StaQC: A better source for downstream tasks

- Code Retrieval as an exemplar downstream task (SQL domain).
- Neural Network model CODENN. [Iyer et al., 2016]
  - CODENN(Original): trained on ~26K heuristically collected QC pairs.
  - CODENN(StaQC): trained on ~120K systematically mined QC pairs.



~6% absolute gain (conservative)

StaQC: Systematically mined Question-Code pairs

StaQC: the largest dataset to date of ~148K Python and ~120K SQL “how-to-do-it”* Question-Code pairs!

Continuously growing in *size* and *diversity*

stackoverflow

A better source for constructing models mapping between natural language and programming language

Data and code are available at:
https://github.com/LittleYUYU/StackOverflow-Question-Code-Dataset

Thank you! Questions?