NS3 – Neuro-Symbolic Semantic Code Search

Shushan Arakelyan, Anna Hakhverdyan, Miltos Allamanis, Luis Garcia, Christophe Hauser, Xiang Ren

University of Southern California, National Polytechnic University of Armenia, Microsoft Research Cambridge, USC Information Sciences Institute

shushana@usc.edu

TL; DR

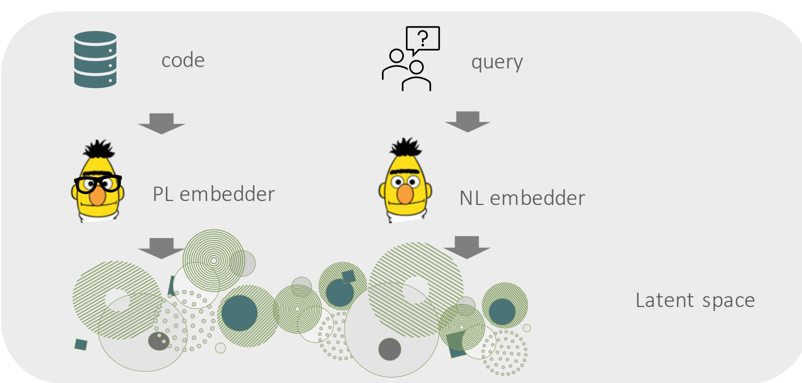
- We propose improving the ***query embedding*** for code retrieval so that it represents the query more ***faithfully***, and thus enable ***precise*** retrieval.
- We do that by performing ***multi-stage*** reasoning on the query and code, which also benefit queries with multiple clauses or conditions.

Introduction

A common way for performing semantic code search is using embedders for programming and natural languages and measuring the similarity of the query and the code in some latent space.

Prior work

- Made programming language embedders more expressive
- Developed new ways to define the similarity metric and the corresponding latent space.



In this work

- We focus on embedding of the query
- We base our approach on our intuition of how a real engineer would locate a snippet of code.

Example

Task A: 1) *Are there references to array or list?*

Task B: 1) *Are iterations over lists present?*

2) *Are there comparisons of elements?*

3) *Are elements swapped to produce sorted result?*

Task A: 1) ✓
Task B: 1) ✓ 2) ✓ 3) ✓

```
def bubbleSort(arr):
  n = len(arr)                          ← 1
  for i in range(n-1):                  ← 1
    for j in range(0, n-i-1):           ← 2
      if arr[j] > arr[j + 1] :
        arr[j], arr[j + 1] =            ← 3
          arr[j + 1], arr[j]
```

Task A: 1) ✓
Task B: 1) ✓ 2) ✓ 3) ×

```
def findMax(arr):
  n = len(arr)
  m = arr[0]
  for i in range(n-1):                  ← 1
    if arr[i] > m:                      ← 2
      m = arr[i]
```

Task A Finding bubble sort could require locating parts of the code that look like they are handling arrays

Task B It then would require checking:

- whether the found array is being traversed;
- whether its elements are being compared to each other;
- and whether they are being swapped as a result of that comparison.

Method

We identify the entities and the actions in the query and how they relate to each other. We do that by creating the semantic parse of the query.

- entities are likely to be represented as nouns or noun phrases

E for discovering entities (N/NP)

- actions will be verbs/verbal phrases

A for discovering actions (V/VP)

E ("all tables") E("dataset")

A (LOAD FROM, E(ARG0), E(ARG1))

The query is modelled with a ***neural module network*** whose layout is defined by the structure of the semantic parse.

Two distinct modules - one for entities and one for actions - are jointly trained within the module network, where entities are leaves and their output is passed as input to some action module.

Entity Discovery Module

The entity module measures ***semantic relevance*** of every code token to its input entity.

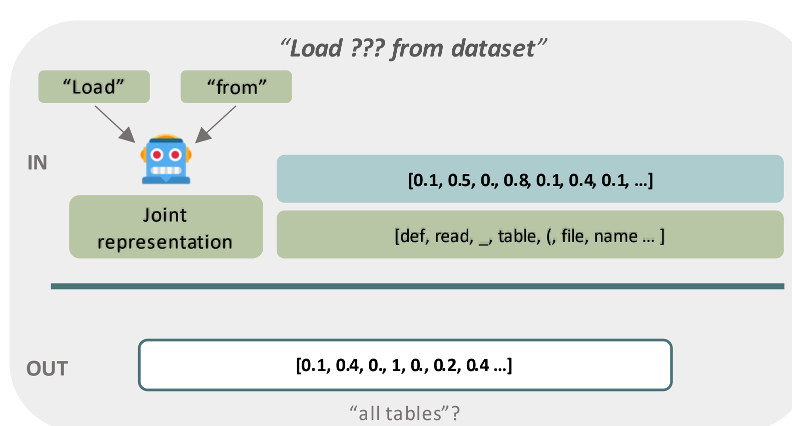
Example

IN *"dataset"* [def, read, _, table, (, file, name ...]

OUT [0.1, 0.5, 0.8, 0.1, 0.4, 0.1, 0.1, ...]

Action Module

- Action module works in a cloze fashion - it gets relevance scores and inputs for ***all but one*** entity in the query, and it tries to estimate the relevance scores for this missing entity.
- If the query corresponds to the code, the action module should be able to correctly estimate missing scores.
- The similarity of code and query is estimated by measuring the similarity between the prediction of the action module and masked relevance scores.



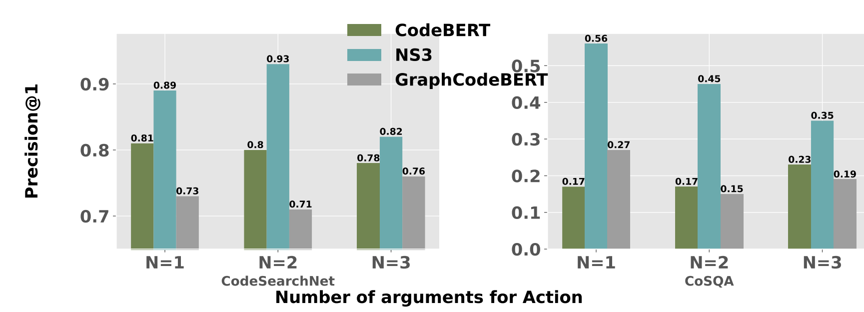
Results

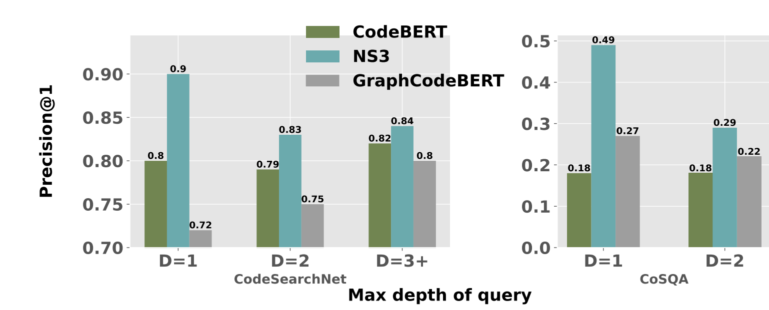
Results on CodeSearchNet and CoSQA datasets

Method	CSN			CSN-10K			CSN-5K			CoSQA		
	MRR	P@1	P@5	MRR	P@1	P@5	MRR	P@1	P@5	MRR	P@1	P@5
BM25	0.209	0.144	0.273	0.209	0.144	0.273	0.209	0.144	0.273	0.103	0.05	0.142
RoBERTa (code)	0.842	0.768	0.933	0.461	0.296	0.664	0.29	0.146	0.438	0.279	0.159	0.434
CuBERT	0.225	0.168	0.294	0.144	0.081	0.214	0.081	0.03	0.118	0.127	0.067	0.187
CodeBERT	0.873	0. 803	0.958	0.69	0.550	0.873	0.680	0.535	0.870	0.345	0.175	0.54
GraphCodeBERT	0.812	0.725	0.919	0.786	0.684	0.901	0.773	0.677	0.892	0.435	0.257	0.628
NS3	**0.924**	**0.884**	**0.969**	**0.826**	**0.753**	**0.908**	**0.823**	**0.751**	**0.913**	**0.551**	**0.445**	**0.668**

Solid improvement over baselines in all evaluation setups

How different models perform on *deeper* or *wider* queries?





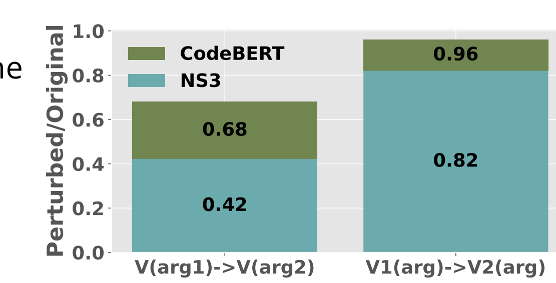
The number of arguments to the action affects the performance and can be used as a proxy for the complexity and for the level of detail in the query.

So does the number of nested action modules, which is a proxy for depth, or compositionality of the query.

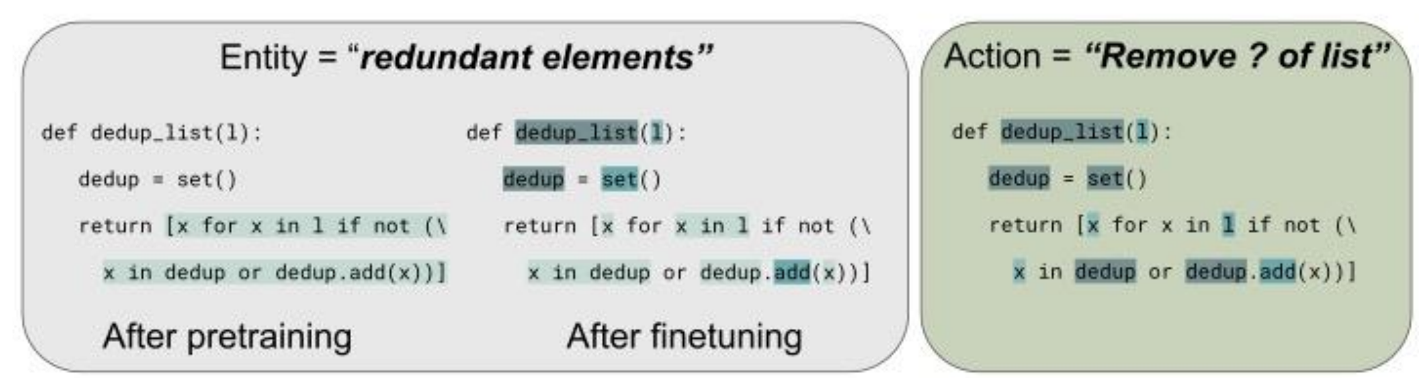
How sensitive the models are to perturbations in the query?

We evaluated model performance on some query-code pairs, and then ***perturbed*** those queries so that the queries no longer correctly match the code.

- Models were evaluated by computing the ratio between the new prediction for the incorrect pair and their original prediction for the correct pair.
- In a ***more sensitive model***, the similarity after a perturbation should drop significantly, thus leading to a lower ratio.



What do module outputs look like at different stages of training?



Entity discovery module scores vs their estimates by the action module