Task and Motivation

Goal: Synthesize programs that are long (10–20 lines) and functionally correct.

Input: pseudocode lines \( x_i \) + public test cases
Output: a program with code lines \( y_i \)
Evaluation: Functional correctness: The program must pass both public + private test cases.
Performance metric: number of synthesis trials
(1 trial = compiler call + execution on all public test cases)

Why? Most existing works either generate short programs or ignore functional correctness during evaluation.

Semantic parsing natural language usually short (e.g., SQ, logical forms) yes (e.g., database execution)
Language to code natural language long (e.g., methods, classes) mostly no (e.g., exact match, BLEU)
Test-driven program synthesis test cases usually short yes
This work natural language long (program) yes

SPoC Dataset

bit.ly/spoc-dataset

Main features:
- Complex programs from programming competitions + test cases, inspired by the NAPS dataset (Kaveshnyksy et al., NAACL 2018).
- 18356 programs
- All programs come with human-authored pseudocode.

Local-level challenges: Translating each line is non-trivial.

High-level descriptions:
- read \( n \) values into array \( a \) and array \( b \)
- sort \( a \) and \( b \) in a loop, printing
- print all elements of \( a \)

Complex sentences and diverse operations:
- change \( m \) to \( k \) if \( k \) and \( n \) are odd
- add \( a \) and \( b \) is a digit return \( 1 \)

Context-dependent interpretation:
- add \( q \) to \( q \) (\( q \) is a set)
- add \( x \) to \( v \) (\( x \) is an integer)
- add \( q \) to \( v \) (\( x \) is a vector)

Global-level challenges:
- The programs are 14 lines long on average.
- One wrong code line can make the whole program incorrect!
- And most programs have at least one difficult line. (See the experiments)

Two data splits: TestP (split by problem) and TestW (split by pseudocode author).

Summary: Given pseudocode and test cases, the task is to synthesize a program, which will be evaluated on functional correctness. We release the SPoC dataset (18K programs + human-authored pseudocode), a search-based synthesizer, and error localization models to guide search based on information from compilation errors.

Experiments and Takeaways

Takeaway 1: Long programs more chances to go wrong. Even though line-level translation accuracy is 85%, the top translations gives a success rate of 24.6%.

Takeaway 2: Search increases the success rate. Under the budget of 100 trials, the success rate goes up to 44.7%.

Takeaway 3: Error localization reduces the number of trials needed:
- The multiclass classification model reduces the number of trials needed in 15.5% of success rate (%)
- Prefix-based pruning reduces the number of trials on easy problems (since we need to compile prefixes) but greatly helps on harder programs.