Appendix

1 Compare with Other Baselines

Table 1: Compare with CodeBERT

Tech	SR@1	SR@5	SR@10	MRR
CodeBERT	xxx	xxx	xxx	xxx
Neural-Bag-Of-Words	xxx	xxx	xxx	xxx
1D-CNN	xxx	xxx	xxx	xxx
Self-Attention (BERT)	xxx	xxx	xxx	xxx
1D-CNN+Self-Attention Hybrid	xxx	xxx	xxx	xxx
TranCS	**0.560**	**0.764**	**0.824**	**0.651**

From Table 1, it is observed that our TranCS outperforms CodeBERT, which is a one of the best masked-language modeling techniques. CodeBERT only takes the tokens of the code snippet into account when generating the embedding representation of it. TranCS uses the semantic-preserving translation of the code snippet to generating the embedding representation, which considers not only tokens, but also contexts including constants, local variables, and data and control dependencies.

From Table 1, it is observed that our TranCS outperforms baselines in CodeSearchNet.

2 Apply Context-aware Code Translation to C/C++

In this section, we introduce a simple example of applying context-aware code translation to the code snippet in C.

```
1 void swap(int *xp, int *yp) {
2   int t0 = *xp;
3   int t1 = *yp;
4   *xp = t1;
5   *yp = t0;
6 }
```

(b) A Code Snippet s

```
4: movq   %rdi, -8(%rbp)
8: movq   %rsi, -16(%rbp)
c: movq   -8(%rbp), %rax
10: movl  (%rax), %eax
12: movl  %eax, -20(%rbp)
15: movq  -16(%rbp), %rax
19: movl  (%rax), %eax
1b: movl  %eax, -24(%rbp)
1e: movl  -24(%rbp), %ecx
21: movq  -8(%rbp), %rax
25: movl  %ecx, (%rax)
27: movl  -20(%rbp), %ecx
2a: movq  -16(%rbp), %rax
2e: movl  %ecx, (%rax)
```

(b) Instruction Sequences

4: move the address **value_1** of **xp** into the memory location which is -8 offsets from the address specified by the register rbp.
8: move the address **value_2** of **yp** into the memory location which is -16 offsets from the address specified by the register rbp.
c: move the **value_1** of the memory location which is -8 offsets from the register rbp address into the register rax.
10: move the content **value_3** specified by **value_1** in register rax into the register eax.
12: move the **value_3** in eax into the memory location **t0** which is -20 offsets from the address specified by the register rbp.
15: move the **value_2** in the memory location which is -16 offsets from the register rbp address into the register rax.
19: move the content **value_4** specified by **value_2** in register rax into the register eax.
1b: move the **value_4** in eax into the memory location **t1** which is -24 offsets from the address specified by the register rbp.
1e: move the **value_4** in the memory location **t1** into the register ecx.
21: move the **value_1** in the memory location which is -8 offsets from the register rbp address into the register rax.
25: move the **value_4** in ecx into the content **value_3** specified by **value_1** in register rax.
27: move the **value_3** in the memory location **t0** into the register ecx.
2a: move the **value_2** in the memory location which is -16 offsets from the register rbp address into the register rax.
2e: move the **value_3** in ecx into the content **value_4** specified by **value_2** in register rax.

(c) Instruction Translation

Figure 1: Example of Translating C Code Snippet

Figure 1(a) shows a code snippet s written in C which implement a functionality, that is, to swap two numbers. We use gcc/g++ (a compiler) and objdump (a disassembler) to generate the instruction sequence of the code snippet s (also known as the assembly code in C), as shown in

Figure 1(b). Further, we can use the specification of the assembly instructions [1] to build translation rules. Figure 1(c) shows the translation of the instruction sequence in Figure 1(b). In Figure 1(c), the blue color denotes variable names used in s, such as t0 and t1. The words in bold (e.g., **value_1**) denote the data dependencies among instructions. Instruction context, including local variable, data and control dependencies, can be collected by analyzing the interaction of instructions on registers and the stack in memory. Figure 2 shows an example of the interaction of the instruction sequence in Figure 1(b) on registers and stacks. The blue line represents that the instruction reads data from the stack and writes to the register, while the red line is the opposite. The blue line represents the order in which the instructions are executed.

The above examples were constructed manually to explain that applying TranCS to the C is not as difficult as imagined. In future work, we will automate the above-mentioned instruction generation and instruction translation process, and explore the performance of TranCS in C.

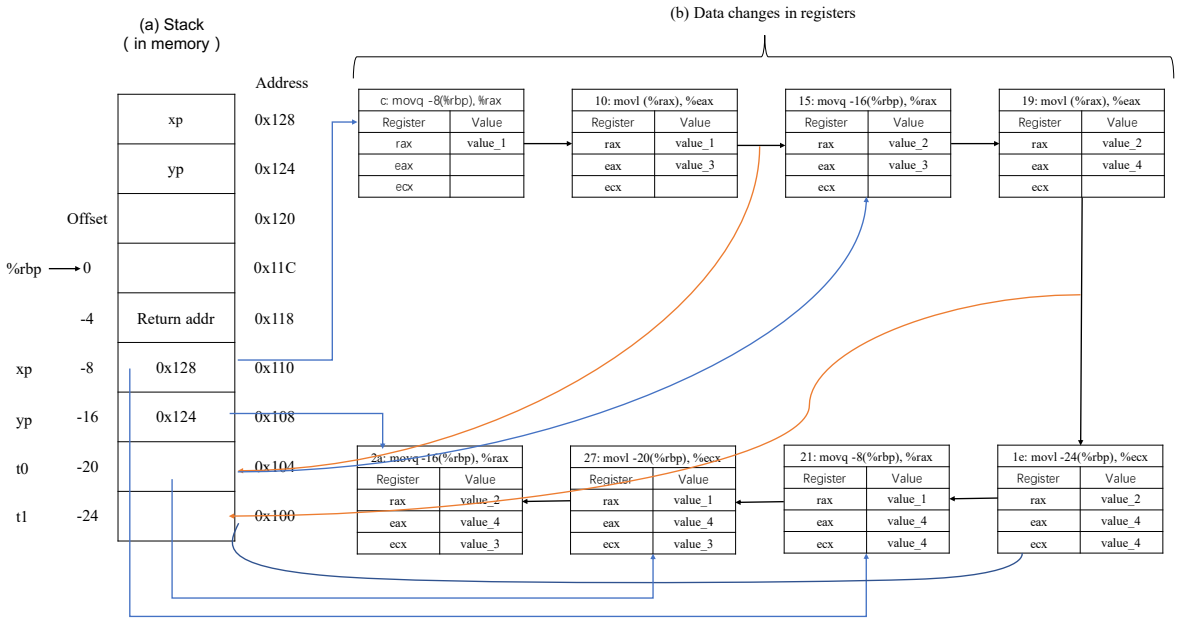


Figure 2: Example of Interaction of Instruction on Stack in Memory and Registers

3 A Case Study of Retrieving API-used Code Snippet

In this section, we provide a case study in which two code snippets with similar semantics have different implementations. As shown in Figure 3, the code snippets s_1 (a) and s_2 (b) implement the same functionality, that is, to swap two elements in the list (c). s_2 implements the functionality by calling the external API *Collection.swap*().

We mix two code snippets into 1,000 test samples, and then use query q to retrieve to check whether TranCS can find them in top k. The experimental results show that TranCS can find them in the top 2 results. This means that TranCS has a certain ability to deal with this case. An intuitive explanation is that when generating code translation, TranCS retains API information (including API name and parameters) through context analysis, so that the final generated representation contains API semantics. The name of the API usually refines the semantics of its implementation.

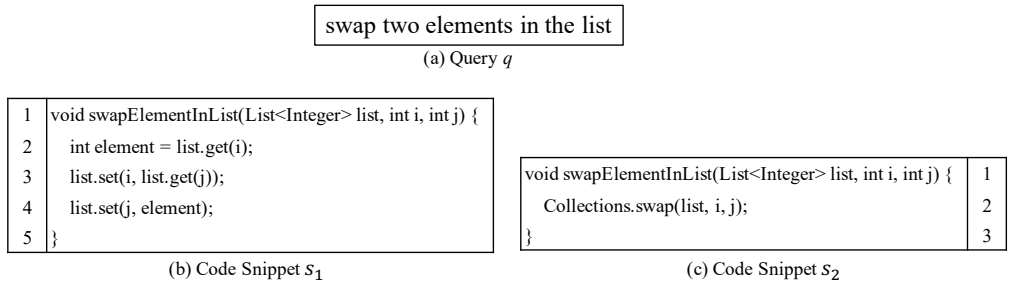


Figure 3: Example of Retrieving API-used Code Snippet

Table 2: Comparison on the 99 manually validated queries

Tech	NDCG@1	NDCG@5	NDCG@10	MRR
DeepCS	xxx	xxx	xxx	xxx
MMAN	xxx	xxx	xxx	xxx
CodeBERT	xxx	xxx	xxx	xxx
TranCS	**xxx**	**xxx**	**xxx**	**xxx**

4 Preliminary Experiment on the 99 manually validated queries from CodeSearchNet

In CodeSearchNet Challenge [1], the normalized discounted cumulative gain (NDCG) is used to evaluate each competing method. NDCG is a commonly used metric [2] in information retrieval. From Table 2, we can observed that TranCS outperforms the state-of-the-art techniques on the 99 manually validated queries.

5 Translations of Code Snippets in Figure 5

Figure 4 shows two code snippets s_a and s_1 with the same control flow graph have different semantics. Based on the pre-defined translation rules, we use TranCS to translate s_a and s_1 into natural language descriptions, as shown in Figure 5.

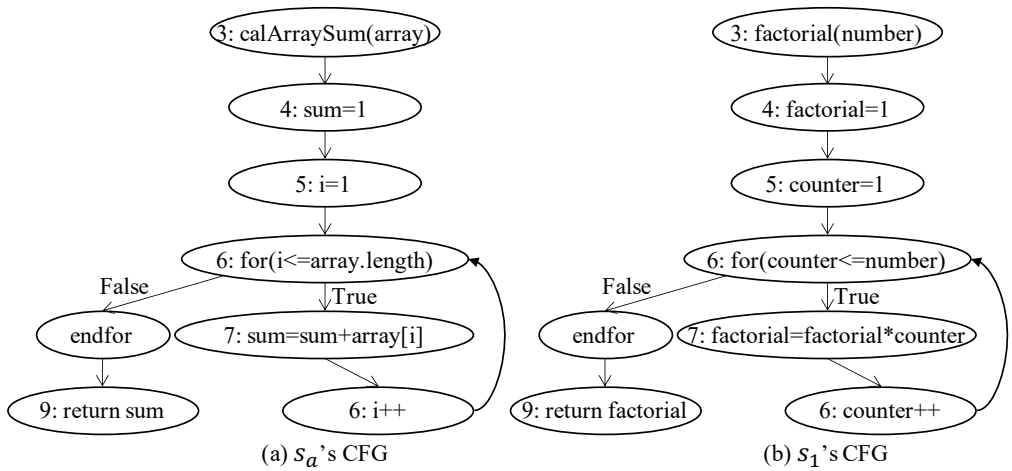
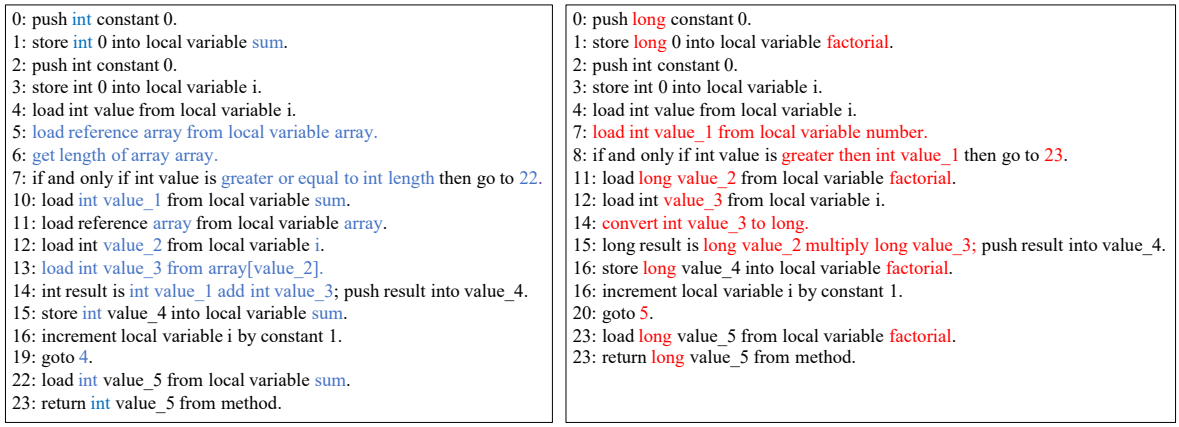


Figure 4: Control Flow Graphs, Corresponding to Figure 5 in the paper.

[1] https://docs.oracle.com/cd/E18752_01/html/817-5477/enmzx.html



(a) s_a's Translation

(b) s_1's Translation

Figure 5: Translations of the Code Snippets in Figure 4

In Figure 5, we use blue and red fonts to distinguish the difference between the two translations. **We can observed that the pre-defined translation rules can actually clearly distinguish the semantics of s_a and s_1.**

References

[1] Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[2] Schütze, H., Manning, C. D., and Raghavan, P. *Introduction to information retrieval*, vol. 39. Cambridge University Press Cambridge, 2008.