DeGPT: Optimizing Decompiler Output with LLM

Peiwei Hu, Ruigang Liang, Kai Chen*

SKLOIS, Institute of Information Engineering, CAS, China

School of Cyber Security, University of Chinese Academy of Sciences, China

PAPER

OPEN SOURCE

Background

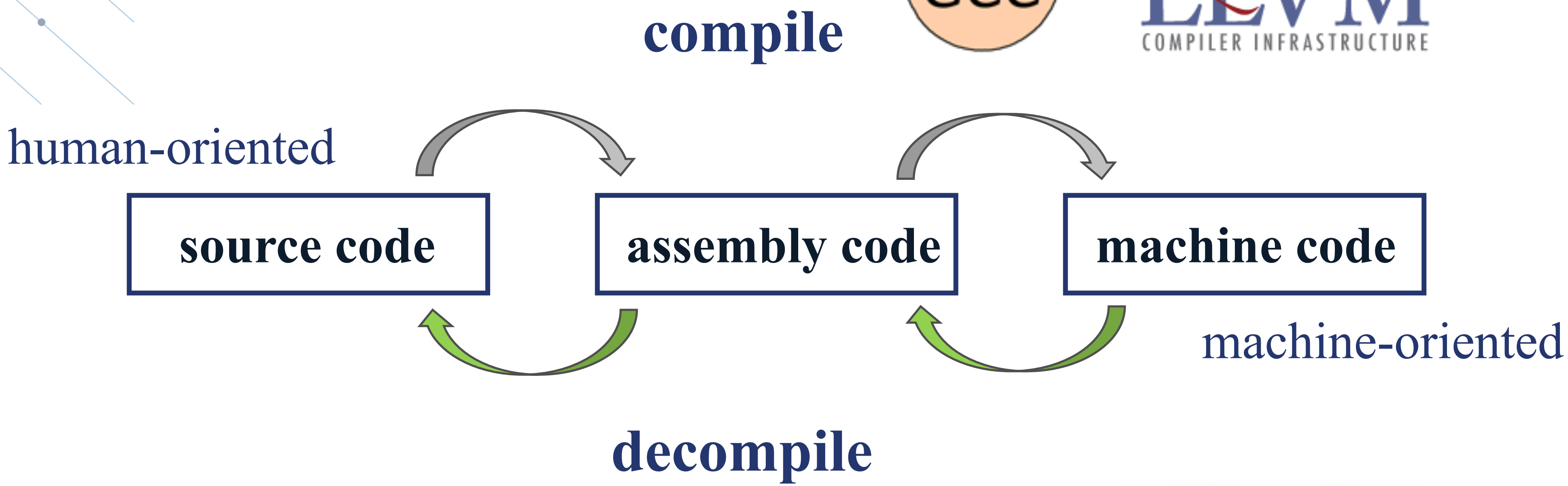
compile

human-oriented

source code → assembly code → machine code

machine-oriented

Background


compile
human-oriented
source code
assembly code
machine code
machine-oriented
decompile

Background

Decompiling is an important way to understand binaries.

vulnerability analysis

virus analysis

software cracking

and so on…

Background

There is still a gap in readability between decompiler output and human-written code.

Source Code

```
// Calculate Fibonacci numbers.
int Fibon(int number){

   if (number == 1 || number == 2) {
      return 1;
   } else{
      return Fibon(number - 1) +
 Fibon(number - 2);
   }
}
```

Decompiler Output

```
int Fibon(int param_1) {
   int iVar1;
   int iVar2;

   if ((param_1 == 1) || (param_1 == 2)) {
      iVar2 = 1;
   } else {
      iVar1 = Fibon(param_1 + -1);
      iVar2 = Fibon(param_1 + -2);
      iVar2 = iVar2 + iVar1;
   }
   return iVar2;
}
```

Variable Name

Comment

Redundant Structure

Drawbacks of decompiler outputs

Inspiration

Large Language Model (LLM) appears…

large number of parameters

trained on abundant codebases

⇩

decent ability for rewriting code, even the unseen code

Challenge

1. How to unlock the potential of LLM on our task?

2. How to filter out the gibberish of LLM?

Method

Break down the task for optimizing decompiler outputs

Step 1. What optimizations to do?

Step 2. How to perform these optimizations?

Step 3. Are optimizations correct?

Method

Break down the task for optimizing decompiler outputs and assign to different roles

Step 1. What optimizations to do? ⇨ Referee

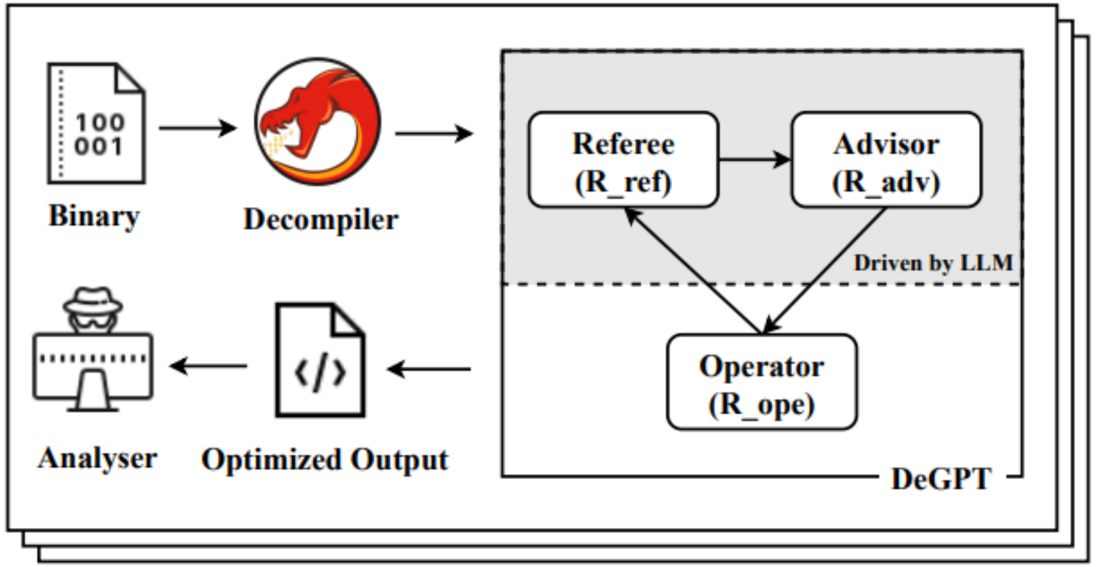
Step 2. How to perform these optimizations? ⇨ Advisor

Step 3. Are optimizations correct? ⇨ Operator

Method

Break down the task for optimizing decompiler outputs

Three-role Model



- ★ **Referee： optimization scheme**
- ★ **Advisor： specific rectification measures**
- ★ **Operator： filter out the gibberish of LLM**

Method

Filter out the gibberish of LLM by checking the changes of function semantics

What can represent function semantics?

Method

Filter out the gibberish of LLM by checking the changes of function semantics

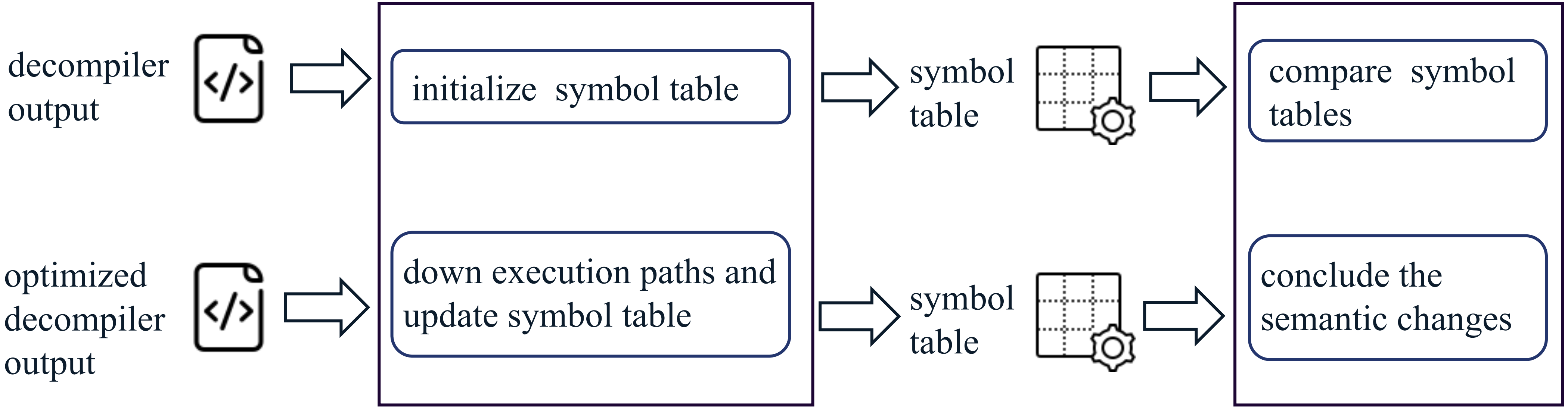
What can represent function semantics?

Return Values, Side Effects

Method

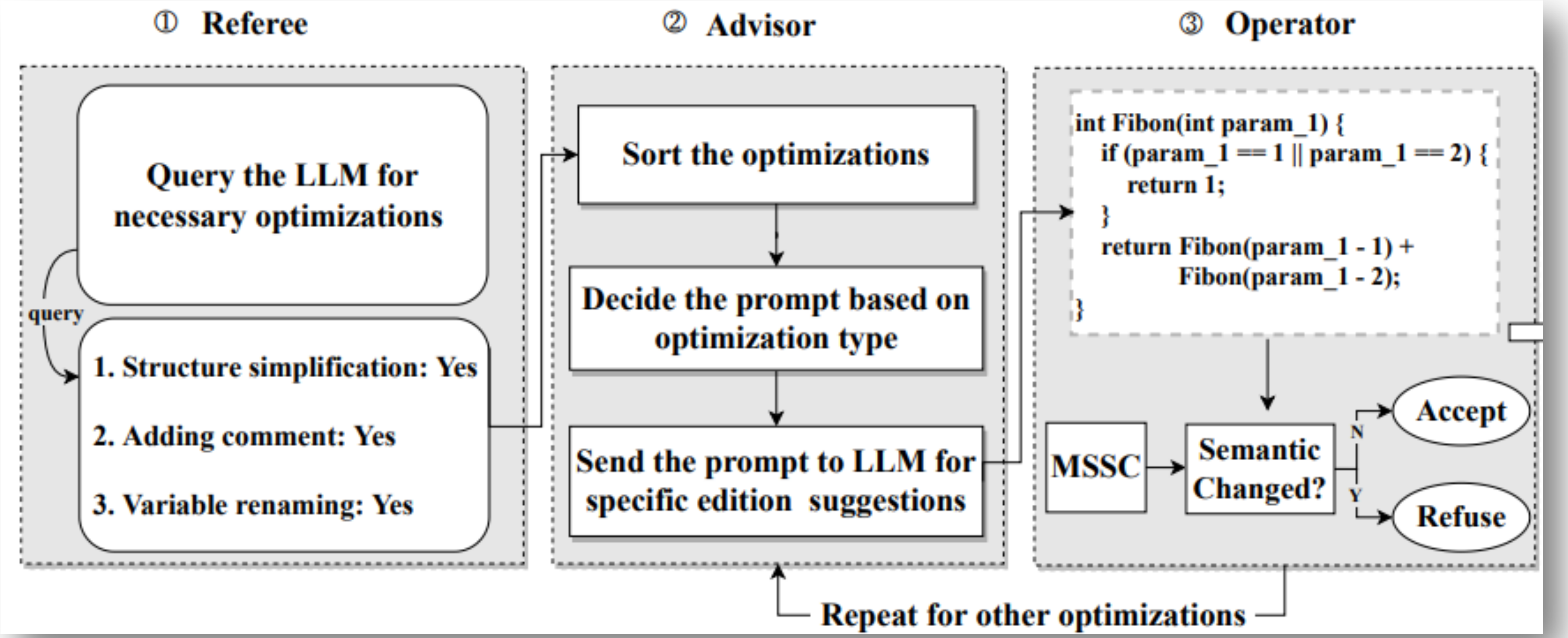
Filter out the gibberish of LLM by checking the changes of function semantics

Micro-Snippet Semantic Calculation (MSSC)



Method

A detailed workflow of DeGPT



Method

An example after optimizing by DeGPT

Decompiler Output

```
int Fibon(int param_1) {
    int iVar1;
    int iVar2;

    if ((param_1 == 1) || (param_1 == 2)) {
        iVar2 = 1;
    } else {
        iVar1 = Fibon(param_1 + -1);
        iVar2 = Fibon(param_1 + -2);
        iVar2 = iVar2 + iVar1;
    }
    return iVar2;
}
```

Optimized Output

```
// Description: Calculates the Fibonacci.
// Parameters:
//      - num: The input for which the ...
// Returns: The Fibonacci number

int Fibon(int num) {
        // Fibonacci for 1 and 2 are both 1.
        if (num == 1 || num == 2) {
                return 1;
        }
        // Recursive step
        return Fibon(num - 1) + Fibon(num - 2);
}
```

Evaluation

Metric for variable renaming:

$$\text{MVR} = \frac{\text{Meaningful Variable(optimized output)}}{\text{Variable Number(optimized output)}}$$

Metric for structure simplification:

$$\text{ER} = \frac{\text{Effort(optimized output)}}{\text{Effort(output)}}$$

Metric for comment appending:

$$\text{CR} = \frac{\text{Correct Comments(optimized output)}}{\text{All Comments(optimized output)}}$$

$$\text{NR} = \frac{\text{Non-trivial Comments(optimized output)}}{\text{Correct Comments(optimized output)}}$$

The codebases and metrics for evaluation

Evaluation

The symbol ① represents non-stripped binaries while ② represents the stripped binaries.

Codebase	MVR (%)		ER (%)		CR (%)		NR (%)	
	①	②	①	②	①	②	①	②
LeetCode	27.0	23.0	75.5	76.9	98.7	100	64.8	36.8
Mirai	29.1	17.3	77.5	75.8	99.4	96.8	71.6	36.7
Coreutils	28.0	20.2	72.0	74.2	98.9	100	62.0	40.6
AudioFlux	37.0	36.4	77.6	78.5	99.6	100	53.0	38.4
Average	30.3	24.2	75.6	76.3	99.2	99.2	62.9	38.1

Finding 1: DeGPT has decent performance on the unseen codebase.

(AudioFlux appeared later than LLM's latest training data.)

Evaluation

The symbol ① represents non-stripped binaries while ② represents the stripped binaries.

Codebase	MVR (%)		ER (%)		CR (%)		NR (%)	
	①	②	①	②	①	②	①	②
LeetCode	27.0	23.0	75.5	76.9	98.7	100	64.8	36.8
Mirai	29.1	17.3	77.5	75.8	99.4	96.8	71.6	36.7
Coreutils	28.0	20.2	72.0	74.2	98.9	100	62.0	40.6
AudioFlux	37.0	36.4	77.6	78.5	99.6	100	53.0	38.4
Average	30.3	24.2	75.6	76.3	99.2	99.2	62.9	38.1

Finding 2: Stripped symbols mainly affect the semantic-related optimizations (variable names and comments).

Evaluation

The symbol ① represents non-stripped binaries while ② represents the stripped binaries.

Codebase	MVR (%)		ER (%)		CR (%)		NR (%)	
	①	②	①	②	①	②	①	②
LeetCode	27.0	23.0	75.5	76.9	98.7	100	64.8	36.8
Mirai	29.1	17.3	77.5	75.8	99.4	96.8	71.6	36.7
Coreutils	28.0	20.2	72.0	74.2	98.9	100	62.0	40.6
AudioFlux	37.0	36.4	77.6	78.5	99.6	100	53.0	38.4
Average	30.3	24.2	75.6	76.3	99.2	99.2	62.9	38.1

Finding 3: DeGPT has a high correct rate in appending comments, but not all appended comments are useful.

Evaluation

- *Q1:* Do you think the optimized decompiler output is more concise (e.g., fewer redundant variables) and idiomatic compared with the original decompiler output?
- *Q2:* Do you think the optimized decompiler output owns more meaningful and helpful variable names compared with the original decompiler output?
- *Q3:* Do you think the optimized decompiler output owns more meaningful and helpful comments compared with the original decompiler output?
- *Q4:* Do you think the optimized decompiler output retains the function semantics (or behaviors) compared with the original decompiler output?
- *Q5:* Generally, do you think the optimized decompiler output is more helpful in understanding the target binary compared with the original decompiler output?

Three Groups

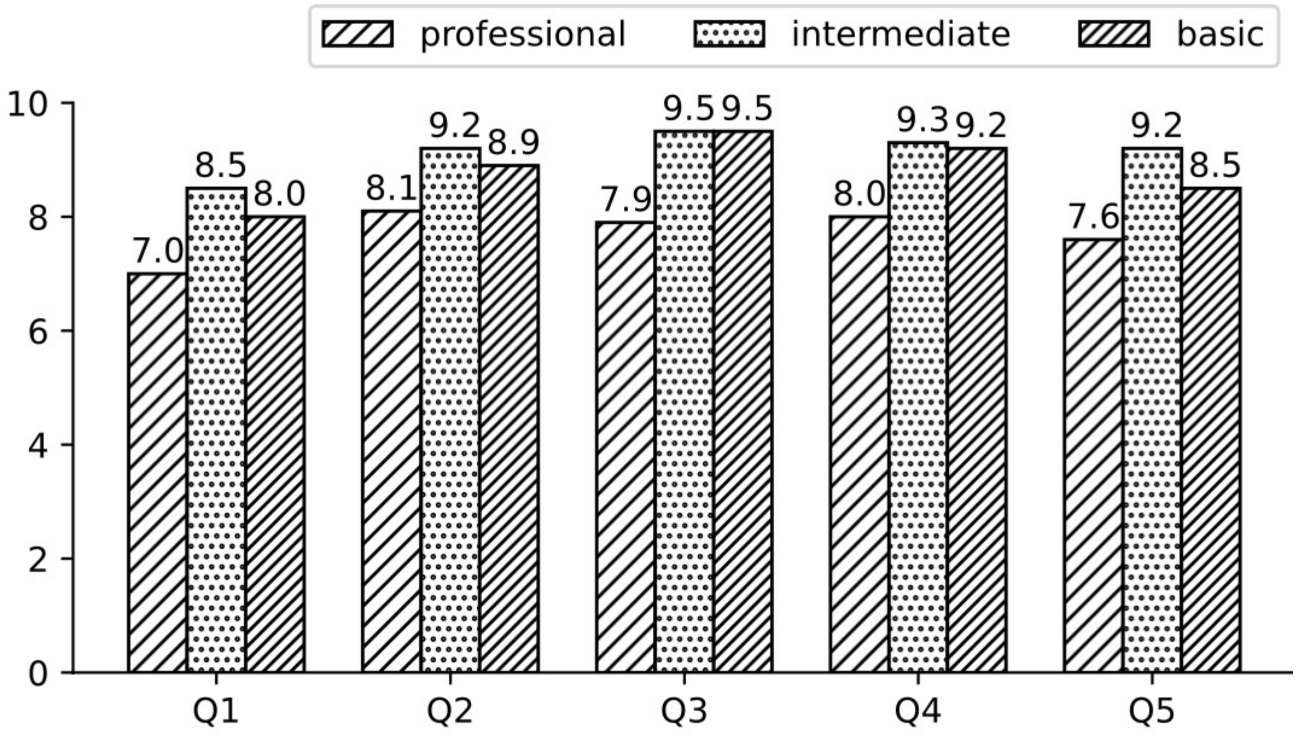
professional

intermediate

basic

The questions for user study

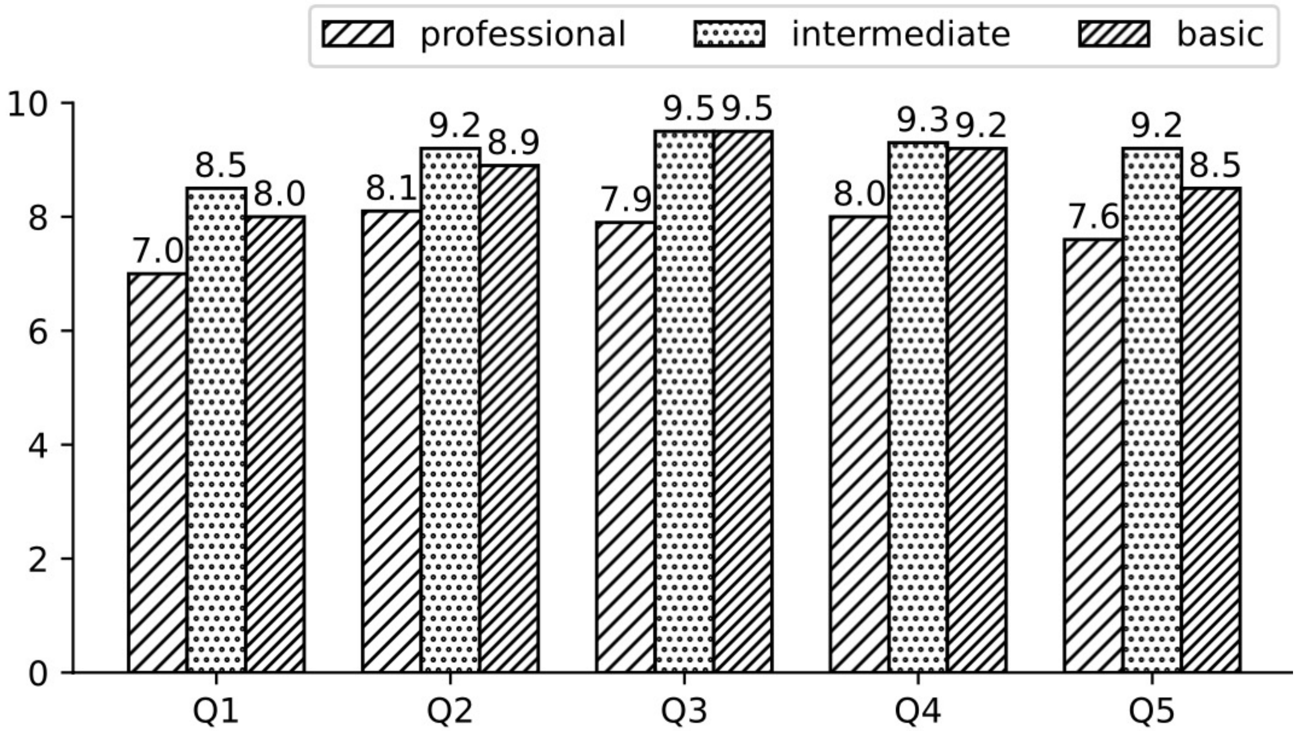
Evaluation



0 – strong disagreement
5 – neutral
10 – strong agreement

Finding 4: Participants of all groups show positive attitudes to the optimized decompiler outputs.

Evaluation



0 – strong disagreement
5 – neutral
10 – strong agreement

Finding 5: The professional group scored lower than participants in the other group, which reflects the limitations of the knowledge of the LLM.

Summary

We propose DeGPT to leverage LLM to improve the readability of decompiler outputs.

It contains a three-role model to unlock the potential of LLM.

It contains MSSC mechanism to filter out the gibberish of LLM.

DeGPT shows decent performance on various metrics and user studies.

中国科学院
CHINESE ACADEMY OF SCIENCES

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

Thanks for your attention!

DeGPT(paper)

Paper

DeGPT(GitHub)

Open Source

https://github.com/PeiweiHu/DeGPT