华为云AI时代的智能云

程序设计语言知识增强的深度代码模型

汇报人：熊英飞

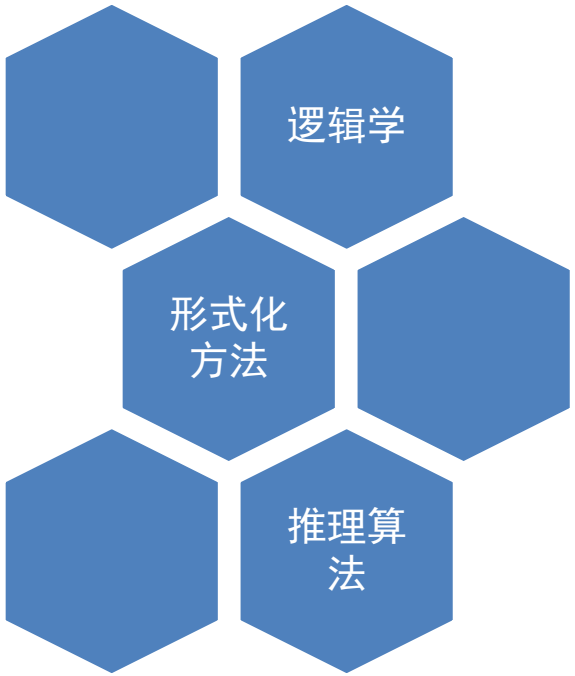
北京大学计算机学院软件研究所

长聘副教授、研究员

人工智能的两条主要途径

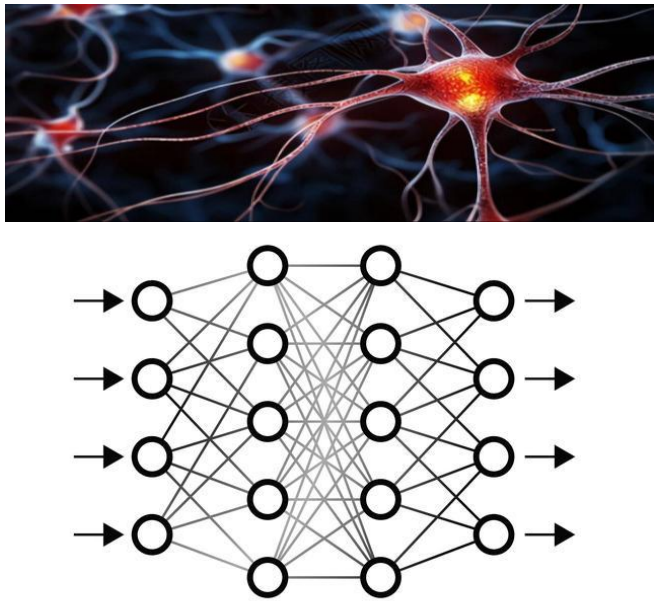
符号主义

- 从人类的知识体系出发构建智能

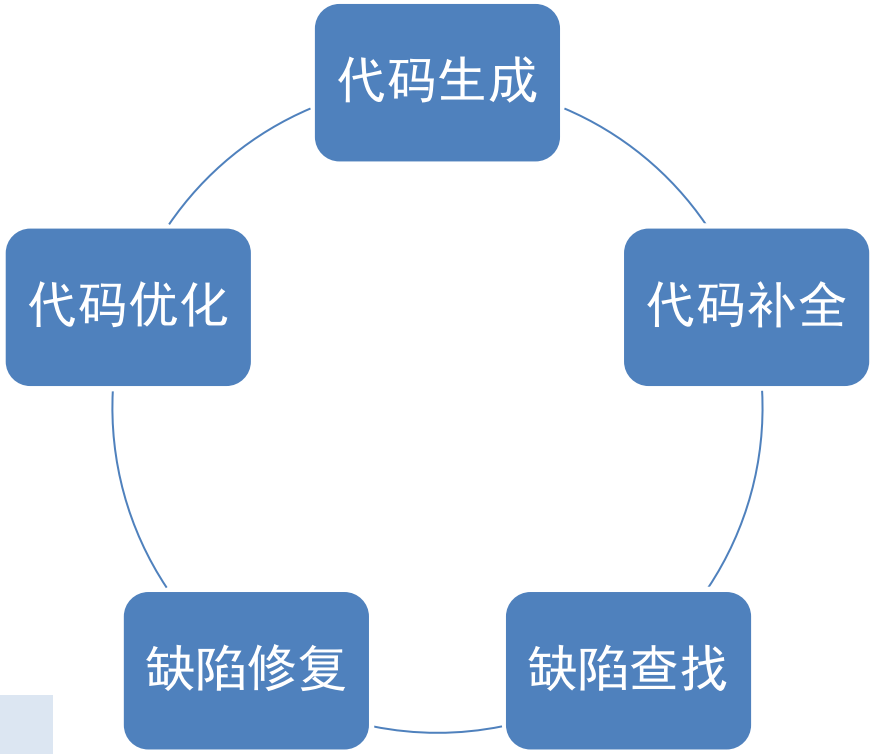


联结主义

- 从大自然进化的结果出发构造智能



辅助软件开发：当代AI最重要的应用之一



麦肯锡报告：采用生成式大模型辅助之后，软件研发成本可以节省 20%-45%。

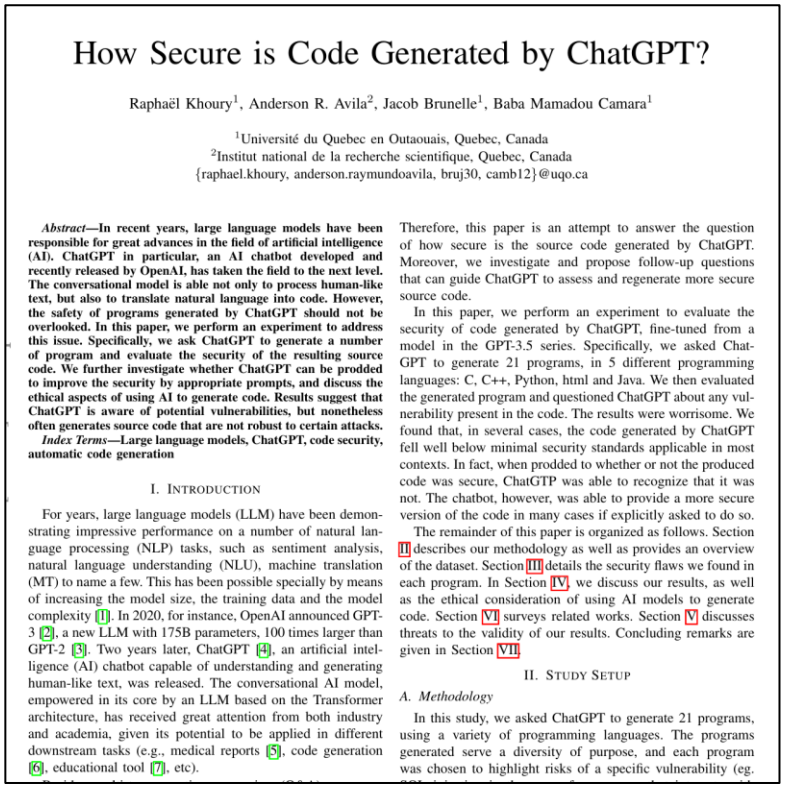
文心快码 Baidu Comate　通义灵码　deepseek　Copilot

联结主义和程序生成的矛盾

程序天然是符号系统，包含基于符号规则定义的

- 语法约束：()+5(不合法
- 类型约束：1+true不合法
- 语义约束：malloc分配的内存不调用free会导致泄漏

等，基于联结主义的神经网络难以学会这些规则，也难以基于规则推导。



How Secure is Code Generated by ChatGPT?

Raphaël Khoury[1], Anderson R. Avila[2], Jacob Brunelle[1], Baba Mamadou Camara[1]

[1]Université du Quebec en Outaouais, Quebec, Canada
[2]Institut national de la recherche scientifique, Quebec, Canada
{raphael.khoury, anderson.raymundoavila, bruj30, camb12}@uqo.ca

***Abstract*—In recent years, large language models have been responsible for great advances in the field of artificial intelligence (AI). ChatGPT in particular, an AI chatbot developed and recently released by OpenAI, has taken the field to the next level. The conversational model is able not only to process human-like text, but also to translate natural language into code. However, the safety of programs generated by ChatGPT should not be overlooked. In this paper, we perform an experiment to address this issue. Specifically, we ask ChatGPT to generate a number of program and evaluate the security of the resulting source code. We further investigate whether ChatGPT can be prodded to improve the security by appropriate prompts, and discuss the ethical aspects of using AI to generate code. Results suggest that ChatGPT is aware of potential vulnerabilities, but nonetheless often generates source code that are not robust to certain attacks.**

***Index Terms*—Large language models, ChatGPT, code security, automatic code generation**

I. Introduction

For years, large language models (LLM) have been demonstrating impressive performance on a number of natural language processing (NLP) tasks, such as sentiment analysis, natural language understanding (NLU), machine translation (MT) to name a few. This has been possible specially by means of increasing the model size, the training data and the model complexity [1]. In 2020, for instance, OpenAI announced GPT-3 [2], a new LLM with 175B parameters, 100 times larger than GPT-2 [3]. Two years later, ChatGPT [4], an artificial intelligence (AI) chatbot capable of understanding and generating human-like text, was released. The conversational AI model, empowered in its core by an LLM based on the Transformer architecture, has received great attention from both industry and academia, given its potential to be applied in different downstream tasks (e.g., medical reports [5], code generation [6], educational tool [7], etc).

Therefore, this paper is an attempt to answer the question of how secure is the source code generated by ChatGPT. Moreover, we investigate and propose follow-up questions that can guide ChatGPT to assess and regenerate more secure source code.

In this paper, we perform an experiment to evaluate the security of code generated by ChatGPT, fine-tuned from a model in the GPT-3.5 series. Specifically, we asked ChatGPT to generate 21 programs, in 5 different programming languages: C, C++, Python, html and Java. We then evaluated the generated program and questioned ChatGPT about any vulnerability present in the code. The results were worrisome. We found that, in several cases, the code generated by ChatGPT fell well below minimal security standards applicable in most contexts. In fact, when prodded to whether or not the produced code was secure, ChatGTP was able to recognize that it was not. The chatbot, however, was able to provide a more secure version of the code in many cases if explicitly asked to do so.

The remainder of this paper is organized as follows. Section II describes our methodology as well as provides an overview of the dataset. Section III details the security flaws we found in each program. In Section IV, we discuss our results, as well as the ethical consideration of using AI models to generate code. Section VI surveys related works. Section V discusses threats to the validity of our results. Concluding remarks are given in Section VII.

II. Study Setup

A. Methodology

In this study, we asked ChatGPT to generate 21 programs, using a variety of programming languages. The programs generated serve a diversity of purpose, and each program was chosen to highlight risks of a specific vulnerability (eg.

GPT生成的代码片段中，至少76%包含各种安全漏洞。

北京大学团队和毕业生相关工作

TreeGen [AAAI20]
- 使用Transformer实现玲珑框架
- 首个基于Transformer的代码生成工作
- 千万参数级别效果综合最优

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare [ICSE23]
- 引导神经网络学习Java类型规则

GrammarT5 [ICSE24]
- 拓展玲珑框架到预训练(200M)
- 亿级参数级别效果综合最优

DeepSeek-Coder [arxiv]
- 引导神经网络学习Def-Use关系
- 我国首个达到国际顶尖水平的开源代码模型

GrammarCoder [arxiv]
- 拓展玲珑框架到Decoder-only
- 十五亿级别效果综合最优

实现

玲珑框架L2S [TOSEM22]
- 通过语法规则序列表示程序
- 保证语法正确性
- 在生成过程中保持程序结构，允许进一步分析

应用

ACS [ICSE17]
- 应用到修复，首个高准确率（>70%）程序修复方法

Recoder [FSE21]
- 应用到修复，首个达到SOTA的神经网络修复方法

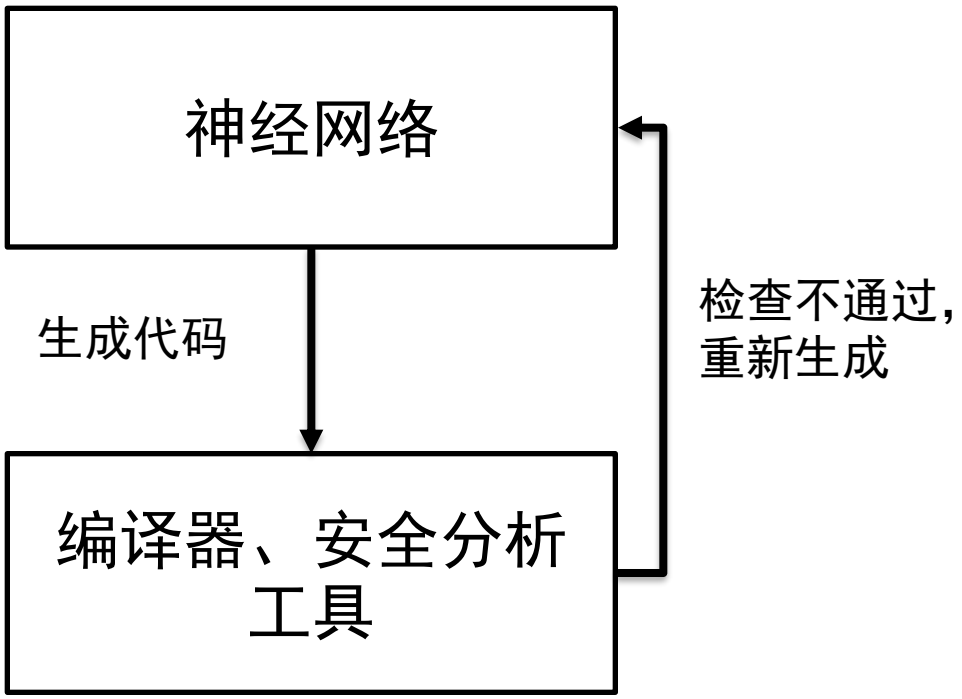
ET [APRCOMP24]
- 应用到修复，获国际修复大赛全球第一名

OCoR [ASE20]
- 应用到代码搜索，显著超过已有方法

LEAM [ASE22]
- 应用到变异生成，显著超过已有方法

能否让神经网络仅生成安全代码？

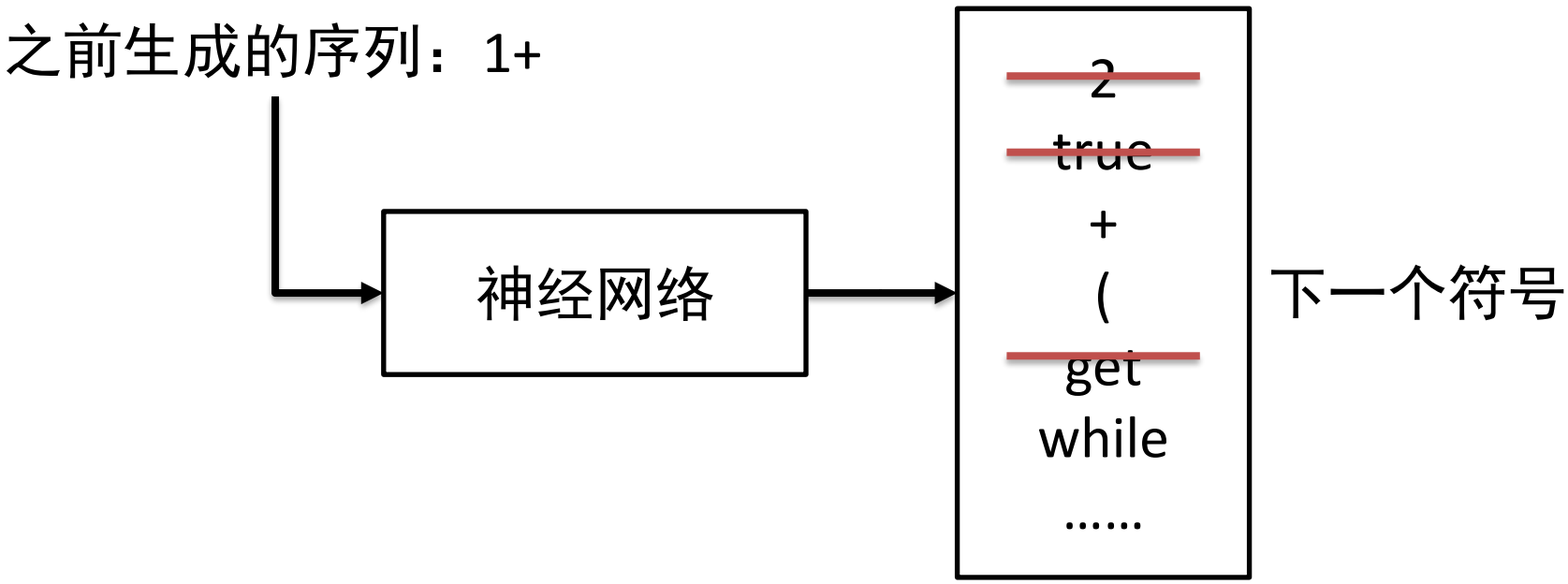
尝试1：生成之后检查



低效，反复生成包含同样错误的代码。

能否让神经网络仅生成安全代码？

尝试2：在生成的时候实时分析，过滤掉不合法的输出



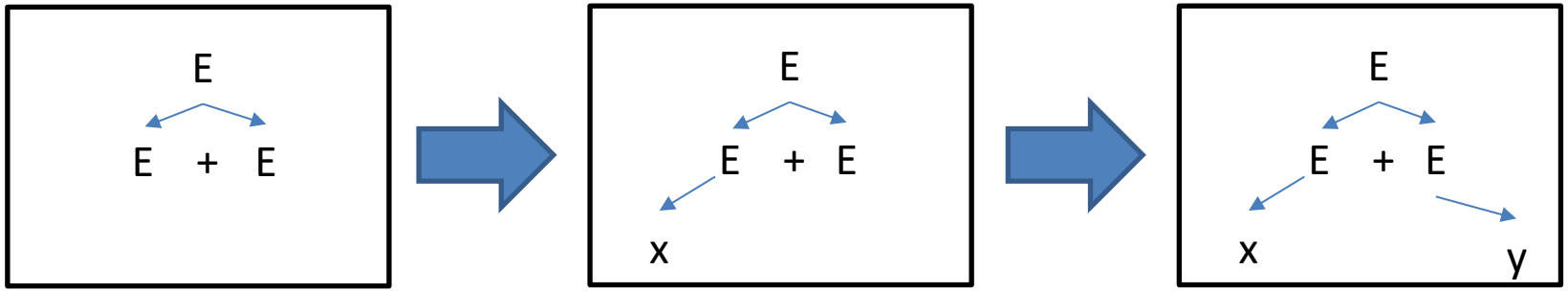
部分程序的分析非常困难：神经网络输出的符号是BPE分词结果，甚至难以词法分析，更不要说语法解析代码结构。

解决方案：语法规则序列表示程序

玲珑框架[TOSEM22]：通过语法规则序列表示程序。

程序	x+y
单词序列	x, +, y
规则序列	r_1, r_2, r_3

$$r_1: E \rightarrow E + E$$
$$r_2: E \rightarrow x$$
$$r_3: E \rightarrow y$$



如何保证找到的程序满足编程语言的约束？

满足语法约束：直接可检查

如何满足类型和语义约束？

基于抽象解释可以对文法规则预分析

- E→E && E产生类型为Bool的表达式
- E→malloc(E)产生一块待释放的内存

基于预分析的结果快速剪枝

- 假设目前的部分程序1+E
- 可以知道E→E && E不是合法展开式

北京大学团队和毕业生相关工作

TreeGen [AAAI20]
- 使用Transformer实现玲珑框架
- 首个基于Transformer的代码生成工作
- 千万参数级别效果综合最优

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare [ICSE23]
- 引导神经网络学习Java类型规则

GrammarT5 [ICSE24]
- 拓展玲珑框架到预训练(200M)
- 亿级参数级别效果综合最优

DeepSeek-Coder [arxiv]
- 引导神经网络学习Def-Use关系
- 我国首个达到国际顶尖水平的开源代码模型

GrammarCoder [arxiv]
- 拓展玲珑框架到Decoder-only
- 十五亿级别效果综合最优

实现

玲珑框架L2S [TOSEM22]
- 通过语法规则序列表示程序
- 保证语法正确性
- 在生成过程中保持程序结构，允许进一步分析

应用

ACS [ICSE17]
- 应用到修复，首个高准确率（>70%）程序修复方法

Recoder [FSE21]
- 应用到修复，首个达到SOTA的神经网络修复方法

ET [APRCOMP24]
- 应用到修复，获国际修复大赛全球第一名

OCoR [ASE20]
- 应用到代码搜索，显著超过已有方法

LEAM [ASE22]
- 应用到变异生成，显著超过已有方法

玲珑框架局限性

在玲珑框架主要从外部加上语法、类型、语义等约束。
神经网络并没有建立起能正确推断符号规则的概率模型，
可能导致概率推断出现偏差。

```
bool m(bool a, bool b) {
    return _____
}
```

加法语句很常见，多半要用加法

不懂类型的神经网络

参数都是bool形，与或更有可能

懂类型的神经网络

北京大学团队和毕业生相关工作

TreeGen [AAAI20]
- 使用Transformer实现玲珑框架
- 首个基于Transformer的代码生成工作
- 千万参数级别效果综合最优

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare [ICSE23]
- 引导神经网络学习Java类型规则

GrammarT5 [ICSE24]
- 拓展玲珑框架到预训练(200M)
- 亿级参数级别效果综合最优

DeepSeek-Coder [arxiv]
- 引导神经网络学习Def-Use关系
- 我国首个达到国际顶尖水平的开源代码模型

GrammarCoder [arxiv]
- 拓展玲珑框架到Decoder-only
- 十五亿级别效果综合最优

实现

玲珑框架L2S [TOSEM22]
- 通过语法规则序列表示程序
- 保证语法正确性
- 在生成过程中保持程序结构，允许进一步分析

应用

ACS [ICSE17]
- 应用到修复，首个高准确率（>70%）程序修复方法

Recoder [FSE21]
- 应用到修复，首个达到SOTA的神经网络修复方法

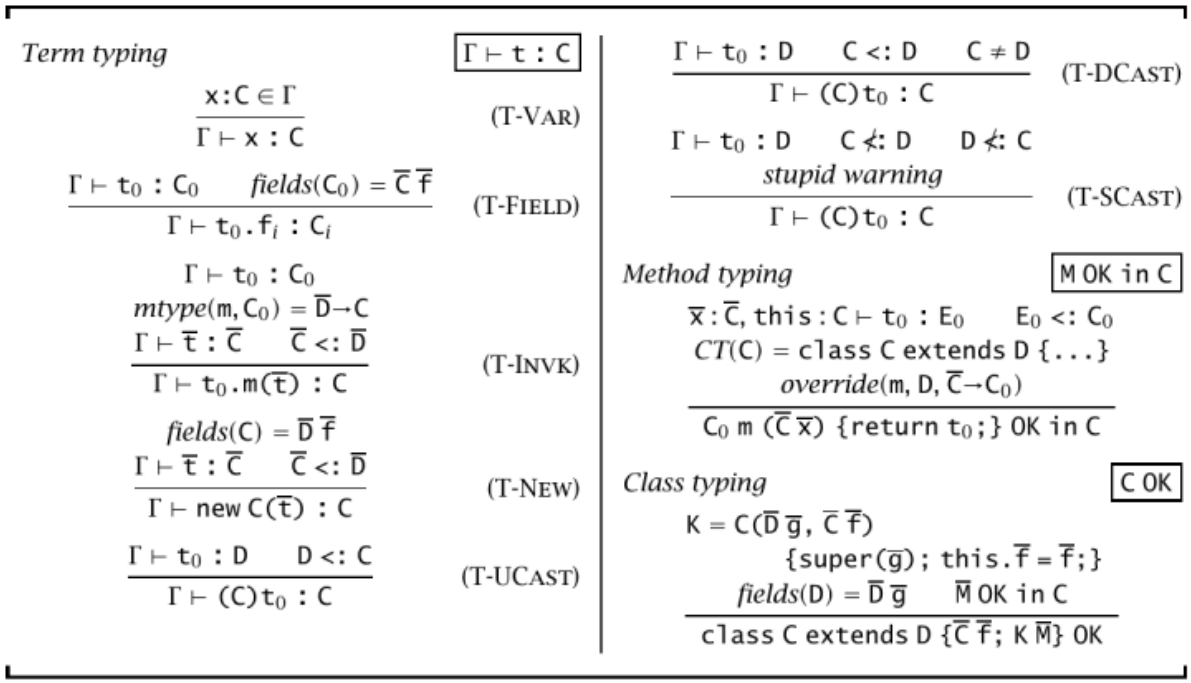
ET [APRCOMP24]
- 应用到修复，获国际修复大赛全球第一名

OCoR [ASE20]
- 应用到代码搜索，显著超过已有方法

LEAM [ASE22]
- 应用到变异生成，显著超过已有方法

引导学习类型规则［ICSE23］

完整类型系统由多条规则组成，很复杂，难以从数据中直接学会



Term typing $\boxed{\Gamma \vdash t : C}$

$$\frac{x{:}C \in \Gamma}{\Gamma \vdash x : C} \quad \text{(T-Var)}$$

$$\frac{\Gamma \vdash t_0 : C_0 \qquad \mathit{fields}(C_0) = \overline{C}\ \overline{f}}{\Gamma \vdash t_0.f_i : C_i} \quad \text{(T-Field)}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_0 : C_0 \\ \mathit{mtype}(m, C_0) = \overline{D} \to C \\ \Gamma \vdash \overline{t} : \overline{C} \qquad \overline{C} <: \overline{D}\end{array}}{\Gamma \vdash t_0.m(\overline{t}) : C} \quad \text{(T-Invk)}$$

$$\frac{\begin{array}{c}\mathit{fields}(C) = \overline{D}\ \overline{f} \\ \Gamma \vdash \overline{t} : \overline{C} \qquad \overline{C} <: \overline{D}\end{array}}{\Gamma \vdash \mathsf{new}\ C(\overline{t}) : C} \quad \text{(T-New)}$$

$$\frac{\Gamma \vdash t_0 : D \qquad D <: C}{\Gamma \vdash (C)t_0 : C} \quad \text{(T-UCast)}$$

$$\frac{\Gamma \vdash t_0 : D \qquad C <: D \qquad C \neq D}{\Gamma \vdash (C)t_0 : C} \quad \text{(T-DCast)}$$

$$\frac{\Gamma \vdash t_0 : D \qquad C \not<: D \qquad D \not<: C \qquad \textit{stupid warning}}{\Gamma \vdash (C)t_0 : C} \quad \text{(T-SCast)}$$

Method typing $\boxed{M\ \mathsf{OK\ in}\ C}$

$$\frac{\begin{array}{c}\overline{x} : \overline{C}, \mathsf{this} : C \vdash t_0 : E_0 \qquad E_0 <: C_0 \\ \mathit{CT}(C) = \mathsf{class}\ C\ \mathsf{extends}\ D\ \{\ldots\} \\ \mathit{override}(m, D, \overline{C} \to C_0)\end{array}}{C_0\ m\ (\overline{C}\ \overline{x})\ \{\mathsf{return}\ t_0;\}\ \mathsf{OK\ in}\ C}$$

Class typing $\boxed{C\ \mathsf{OK}}$

$$\frac{\begin{array}{l}K = C(\overline{D}\ \overline{g}, \overline{C}\ \overline{f}) \\ \qquad \{\mathsf{super}(\overline{g}); \mathsf{this}.\overline{f} = \overline{f};\} \\ \mathit{fields}(D) = \overline{D}\ \overline{g} \qquad \overline{M}\ \mathsf{OK\ in}\ C\end{array}}{\mathsf{class}\ C\ \mathsf{extends}\ D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}\ \mathsf{OK}}$$

Figure 19-4: Featherweight Java (typing)

修复工具生成的程序中只有30%-40%是类型正确的

引导学习类型规则-方法

假设单条规则是容易学会的

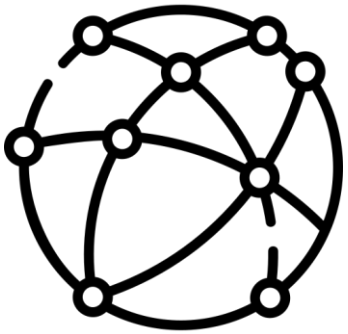
1. 设计数据结构表征学习单条规则所需的信息
2. 修改文法规则记录类型推导的结果

类型规则

$$\frac{\Gamma \vdash v : D \quad \Gamma \vdash t : C \quad C <: D}{\Gamma \vdash v = t : \text{Void}}$$

类型关系

(1)AST和类型
(2)变量和类型
(3)类型之间的子关系



T-Graph

T-Grammar

引导学习类型规则-效果

应用于缺陷修复，形成Tare方法（参数量3510万），成功修复的数量提升26.5%

Project	Bugs	CapGen	SimFix	TBar	DLFix	Hanabi	Recoder	Recoder-F	Recoder-T	Tare
Chart	26	4/4	4/8	9/14	5/12	3/5	8/14	9/15	8/16	**11/16**
Closure	133	0/0	6/8	8/12	6/10	-/-	13/33	14/36	15/31	**15/29**
Lang	64	5/5	9/13	5/14	5/12	4/4	9/15	9/15	11/23	**13/22**
Math	106	12/16	14/26	18/36	12/28	19/22	15/30	16/31	16/40	**19/42**
Time	26	0/0	1/1	1/3	1/2	2/2	**2/2**	**2/2**	**2/4**	**2/4**
Mockito	38	0/0	0/0	1/2	1/1	-/-	**2/2**	**2/2**	**2/2**	**2/2**
Total	393	21/25	34/56	42/81	30/65	28/33	49/96	52/101	54/116	**62/115**

在2024年的国际修复比赛上获得Java组冠军
超过其他团队用几亿、几十亿参数预训练模型构建的工具

北京大学团队和毕业生相关工作

TreeGen [AAAI20]
- 使用Transformer实现玲珑框架
- 首个基于Transformer的代码生成工作
- 千万参数级别效果综合最优

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare [ICSE23]
- 引导神经网络学习Java类型规则

GrammarT5 [ICSE24]
- 拓展玲珑框架到预训练(200M)
- 亿级参数级别效果综合最优

DeepSeek-Coder [arxiv]
- 引导神经网络学习Def-Use关系
- 我国首个达到国际顶尖水平的开源代码模型

GrammarCoder [arxiv]
- 拓展玲珑框架到Decoder-only
- 十五亿级别效果综合最优

实现

玲珑框架L2S [TOSEM22]
- 通过语法规则序列表示程序
- 保证语法正确性
- 在生成过程中保持程序结构，允许进一步分析

应用

ACS [ICSE17]
- 应用到修复，首个高准确率（>70%）程序修复方法

Recoder [FSE21]
- 应用到修复，首个达到SOTA的神经网络修复方法

ET [APRCOMP24]
- 应用到修复，获国际修复大赛全球第一名

OCoR [ASE20]
- 应用到代码搜索，显著超过已有方法

LEAM [ASE22]
- 应用到变异生成，显著超过已有方法

大型预训练模型

现代大型代码预训练模型都采用多种编程语言训练。
不同编程语言文法规则、类型规则各不相同。
以上方法能否应用于预训练模型？

<identifier> ::= <initial> | <initial> <more>
<initial> ::= <letter> | _ | $
<more> ::= <final> | <more > <final>
<final> ::= <initial> | <digit>
<letter> ::= a | b | c | … z | A | B| … | Z
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Java文法

Python文法

```
stringliteral   ::=  [stringprefix](shortstring | longstring)
stringprefix    ::=  "r" | "u" | "R" | "U" | "f" | "F"
                     | "fr" | "Fr" | "fR" | "FR" | "rf" | "rF" | "Rf" | "RF"
shortstring     ::=  "'" shortstringitem* "'" | '"' shortstringitem* '"'
longstring      ::=  "'''" longstringitem* "'''" | '"""' longstringitem* '"""'
shortstringitem ::=  shortstringchar | stringescapeseq
longstringitem  ::=  longstringchar | stringescapeseq
shortstringchar ::=  <any source character except "\" or newline or the quote>
longstringchar  ::=  <any source character except "\">
stringescapeseq ::=  "\" <any source character>
```

大型预训练模型

可简单将文法合并为更大的文法
神经网络能学会不同文法规则之间的关联
其他预训练技术（如BPE分词）也能做到和文法兼容

```
Root -> JavaRoot
      |  PythonRoot
      |  ...
JavaRoot -> Imports Classes
...
```

学习“声明–使用”关系

已有预训练任务随机排列文件

神经网络可能先看到方法调用再看到方法定义

上下文依赖解析：

- 从文件中挖掘“声明–使用”关系
- 对文件排序，保证声明出现在使用之前

大型预训练模型 – 训练效果

GrammarT5 [ICSE24]	GrammarCoder [arxiv25]	DeepSeekCoder [arxiv24]
• 应用语法规则编码 • 5亿参数以下的综合效果最优代码生成模型	• 和快手合作训练 • 应用语法规则编码 • 15亿参数级别效果综合最优	• 和深度求索公司合作训练 • 引导学习“声明-使用”关系 • 我国首个达到国际顶尖水平的开源代码模型

小结

- 代码是符号主义的产物
- 联结主义的神经网络天然不擅长学习和推断符号规则
- 结合符号方法和神经网络可有效提升代码任务的效果
  - **只高效搜索符号约束下的目标程序空间**
  - **引导神经网络学习符号知识**