CCF软工专委青年论坛

神经符号融合的智能软件开发

汇报人：熊英飞

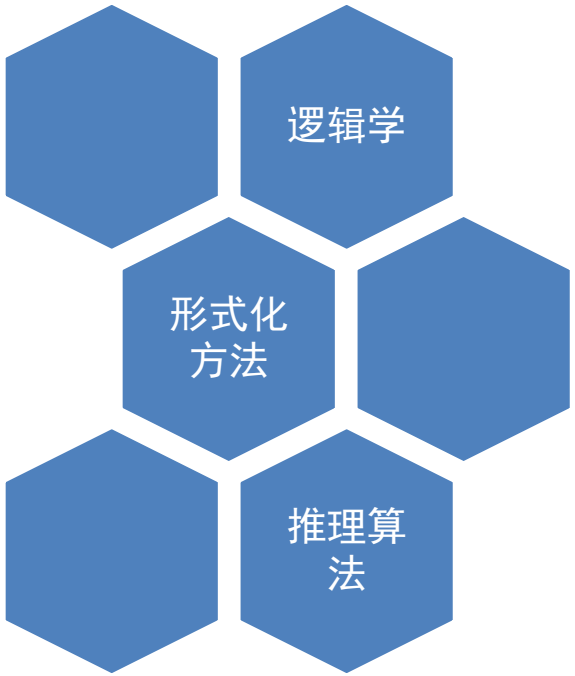
北京大学计算机学院软件研究所

长聘副教授、副所长

人工智能的两条主要途径

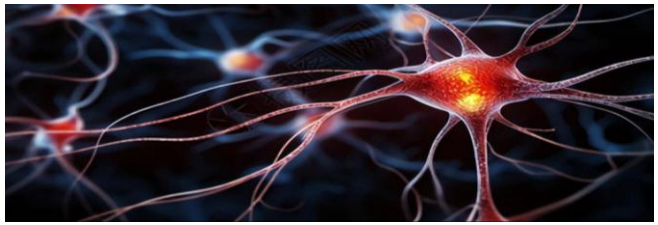
符号主义

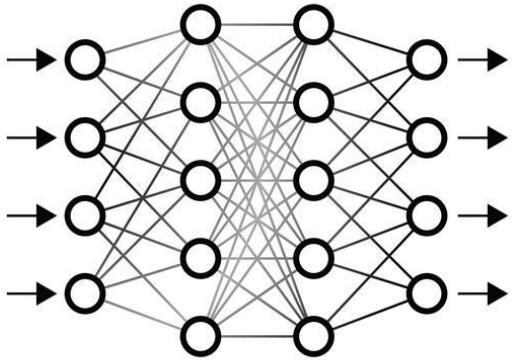
- 从人类的知识体系出发构建智能



连接主义

- 从大自然进化的结果出发构造智能

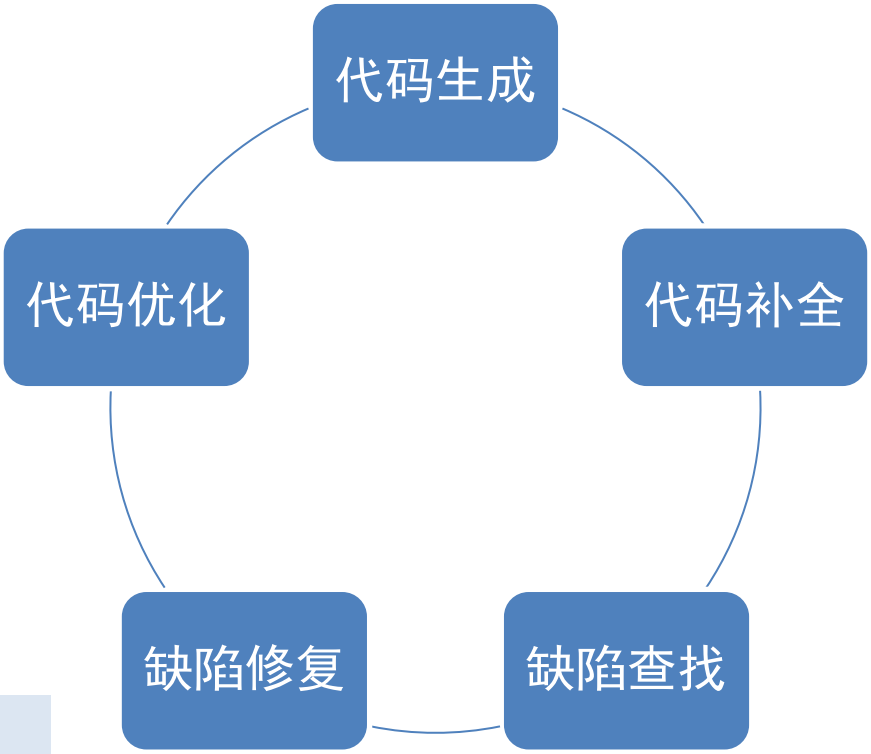




辅助软件开发：当代AI最重要的应用之一



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%。



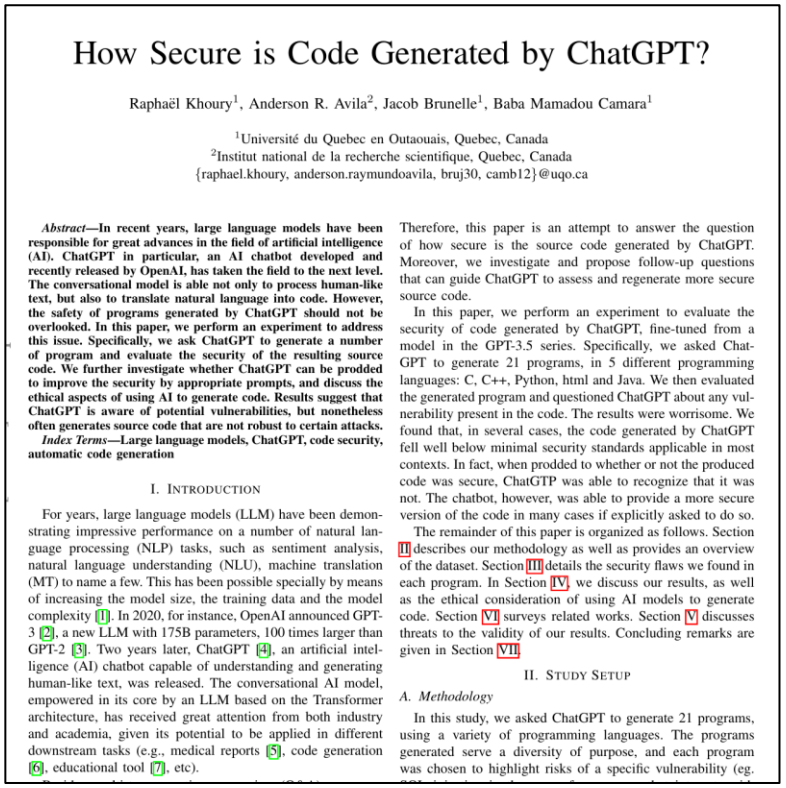
文心快码 Baidu Comate　通义灵码　deepseek　Copilot

连接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，基于连接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。

能否用符号方法加强神经网络？

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

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

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare/Tacs [ICSE23/Submitted]
- 引导神经网络学习类型规则

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

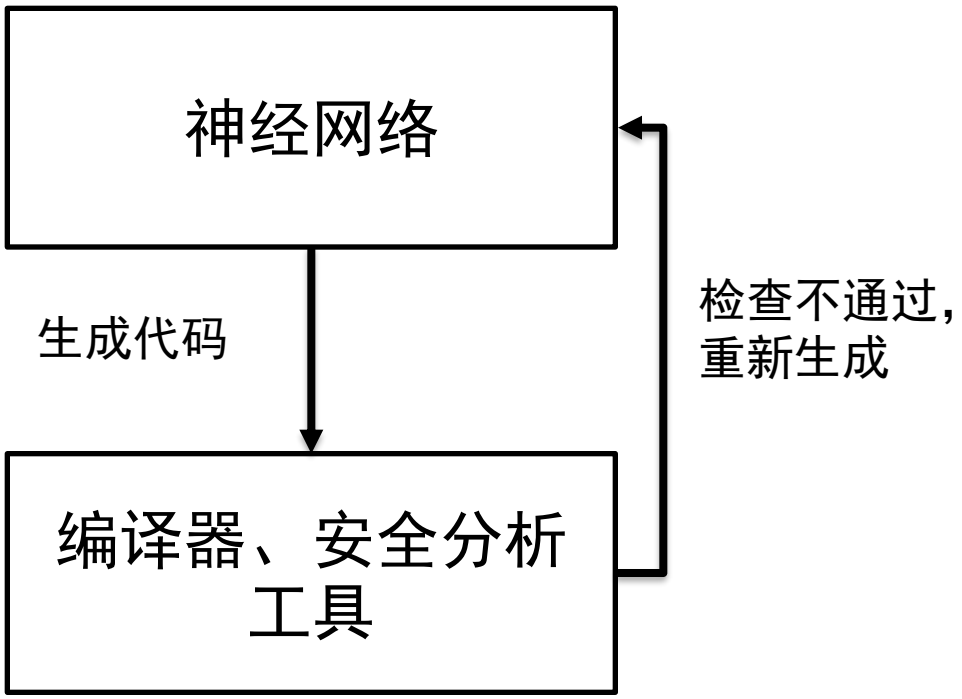
ET [APRCOMP24]
- 应用到修复，获国际修复大赛全球第一名

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？

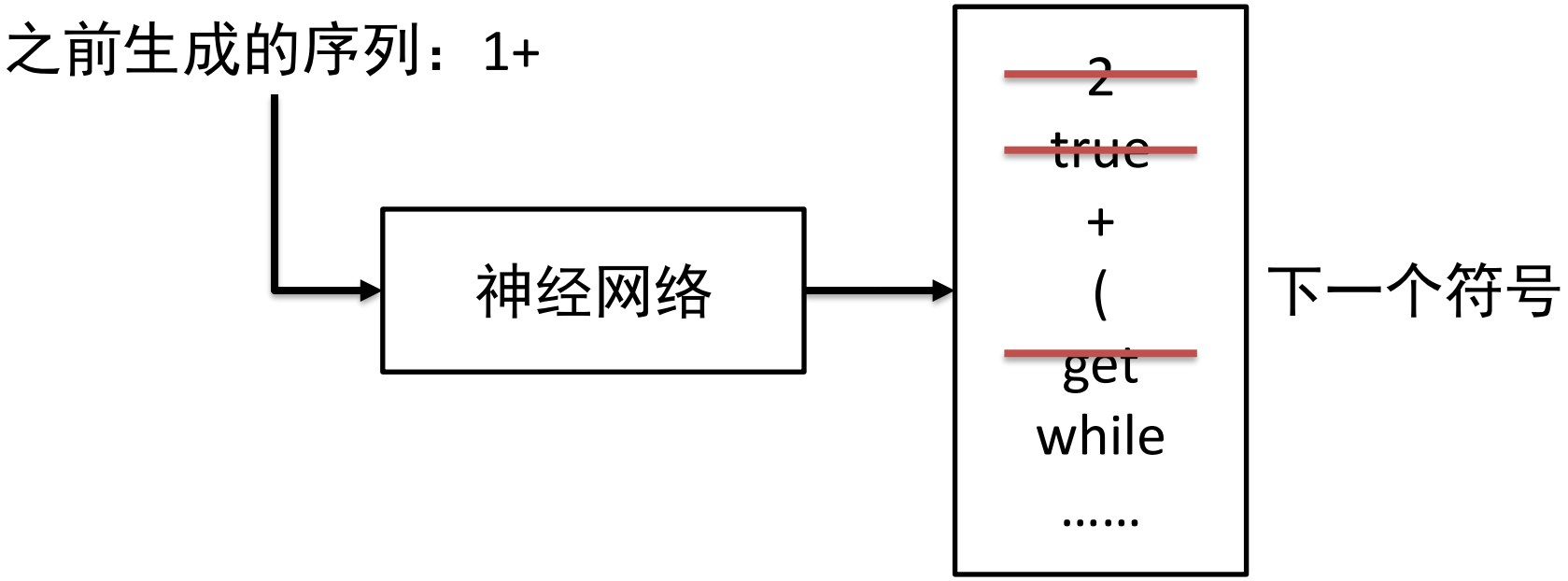
尝试1：生成之后检查



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-约束解码：在生成的时候实时分析，过滤掉不合法的输出



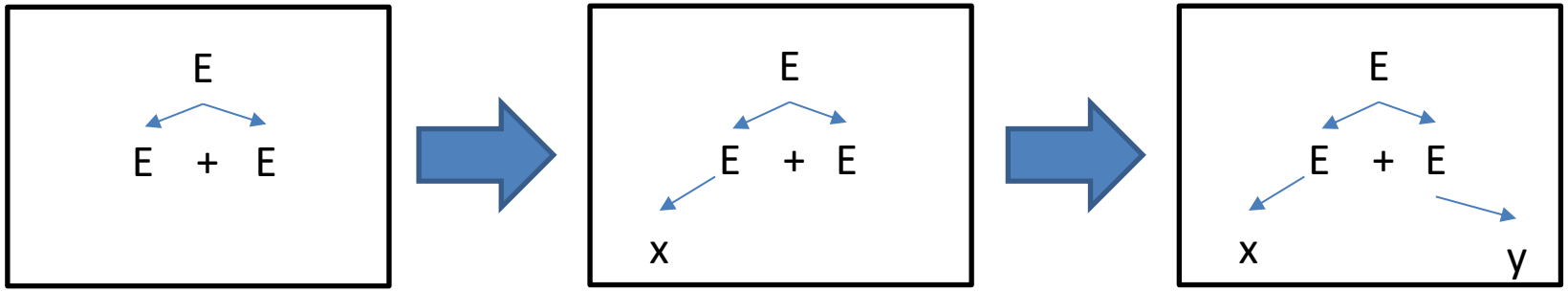
部分程序的分析非常困难：神经网络输出的符号是BPE分词结果，甚至难以词法分析，更不要说语法解析代码结构。
分布错误：由于没有学会安全规则，神经网络可能生成错误的输出分布。

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

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



优势1：约束解码变得容易

很容易就能保证语法正确

类型和语义正确性可以在部分AST上做分析

- 1 + “x” + Expr //类型错误
- if (BoolExpr) then x else x
  //如果输入x=1，期望输出2，则语义错误

做法：

- 采用抽象解释进行分析
- 首先在文法规则上进行预分析：知道非终结符对应的所有终结符
- 然后在部分程序上分析是否一定不正确

优势2：和语义更好的对应[ACL25-Finding]

相同语义

- if (x<0) y=y+1;
- if (x < 0) {
  y = y + 1;
  }

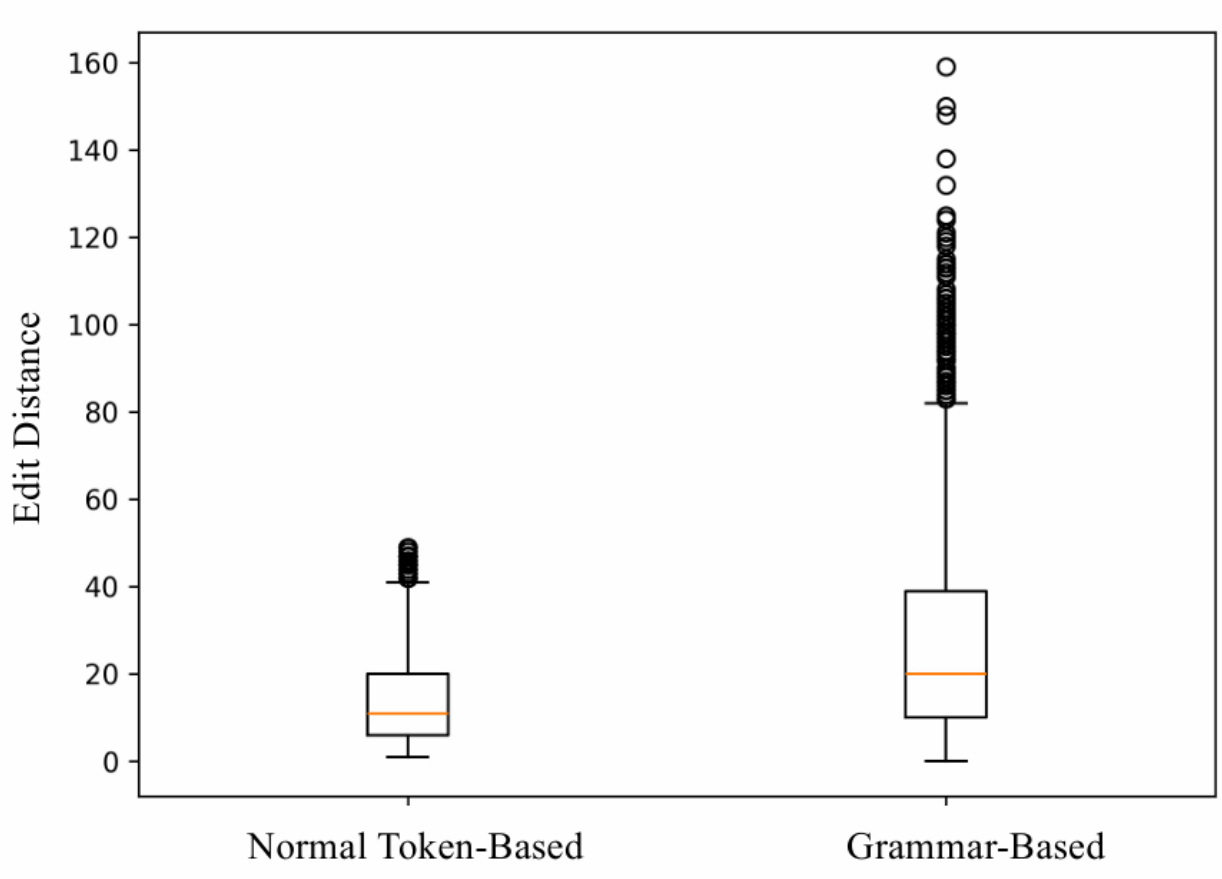
在语法规则序列下相似
在单词序列下不同

不同语义

- for i in range(1, 6):
  x = x + 1
  sum = sum + x
- for i in range(1, 6):
  x = x + 1
  sum = sum + x

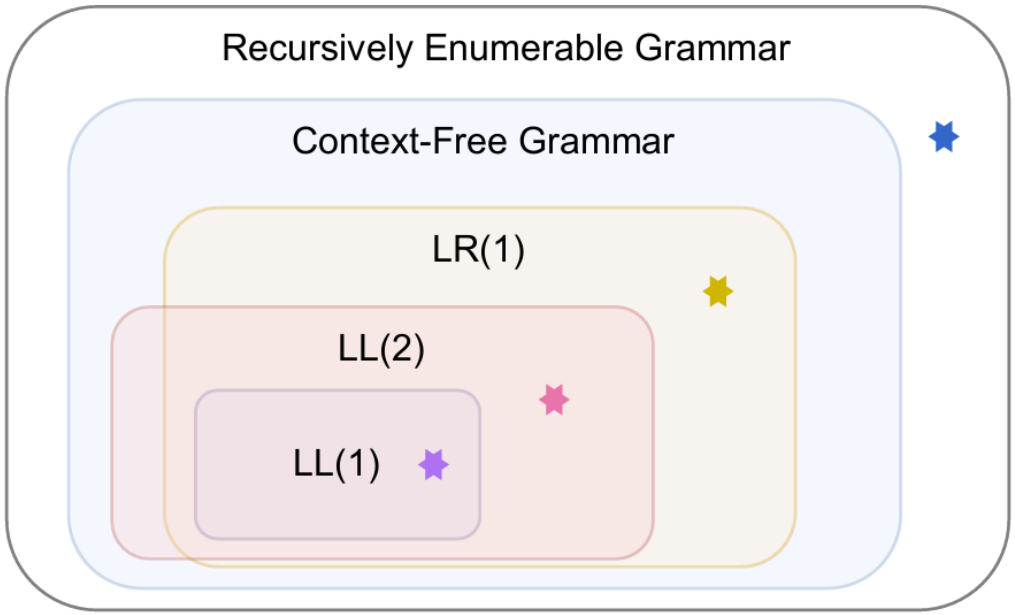
在语法规则序列下不同
在单词序列下相似

语义不同代码之间的编辑距离



优势3：更容易的代码解析[投稿中论文]

编程语言的解析越容易，模型表现就越好。





Language	Mean(%)
$DSL_{LL(1)}$	82.00
$DSL_{LL(2)}$	81.74
$DSL_{LR(1)}$	81.14
DSL_{NCFG}	80.41

优势3：更容易的代码解析[投稿中论文]

编程语言的解析越容易，模型表现就越好。

在更大模型（1-1.5B），Python语言的更多表示上的结果一致。

If n is an integer and $101 * n^2 \leq 3600$, what is the greatest possible value of n? n0 = 101.0 n1 = 2.0 n2 = 3600.0

a. Question

```
1. import math
2. n0 = 101.0
3. n1 = 2.0
4. n2 = 3600.0
5. t0 = n2 / n0
6. t1 = math.sqrt(max(0, t0))
7. answer = math.floor(t1)
```

b. Answer in $DSL_{LR(1)}$

```
1. import math
2. n0 = 101.0
3. n1 = 2.0
4. n2 = 3600.0
5. t0 = / n2 n0
6. t1 = <call> <attr> math.sqrt(<call> max(0, t0))
7. answer = <call> <attr> math.floor(t1)
```

c. Answer in $DSL_{LL(1)}$

```
1. import math
2. <exp> n0 = 101.0
3. <exp> n1 = 2.0
4. <exp> n2 = 3600.0
5. <exp> t0 = <exp> / n2 n0
6. <exp> t1 = <exp> <call> <exp> <attr> math.sqrt(<exp> <call> max(0, t0))
7. <exp> answer = <exp> <call> <exp> <attr> math.floor(t1)
```

d. Answer in $DSL_{LL(2)}$

```
1. import math
2. n0 = 101.0 ; n0
3. n1 = 2.0 ; n1
4. n2 = 3600.0 ; n2
5. t0 = n2 / n0 ; t0
6. t1 = math.sqrt(max(0, t0)) ; t1
7. answer = math.floor(t1) ; ans
```

e. Answer in DSL_{CSG}

Language	Mean(%)
$DSL_{LL(1)}$	82.00
$DSL_{LL(2)}$	81.74
$DSL_{LR(1)}$	81.14
DSL_{NCFG}	80.41

优势3：更容易的代码解析[投稿中论文]

- 语法序列表示属于LL(1)
- 比大多数主流编程语言都要容易解析
  - Python：非上下文无关语言
  - Java：LR

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

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

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare/Tacs [ICSE23/Submitted]
- 引导神经网络学习类型规则

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

采用神经网络实现玲珑框架[AAAI20]

用Transformer实现玲珑框架

- 最早的采用Transformer生成代码的工作
- 将Transformer适配到文法规则上形成TreeGen

	Model	StrAcc	Acc+	BLEU
Plain	LPN (Ling et al. 2016)	6.1	–	67.1
	SEQ2TREE (Dong and Lapata 2016)	1.5	–	53.4
	YN17 (Yin and Neubig 2017)	16.2	~18.2	75.8
	ASN (Rabinovich, Stern, and Klein 2017)	18.2	–	77.6
	ReCode (Hayati et al. 2018)	19.6	–	78.4
	TreeGen-A	**25.8**	**25.8**	**79.3**
Structural	ASN+SUPATT (Rabinovich, Stern, and Klein 2017)	22.7	–	79.2
	SZM19 (Sun et al. 2019)	27.3	30.3	79.6
	TreeGen-B	**31.8**	**33.3**	80.8

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

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

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare/Tacs [ICSE23/Submitted]
- 引导神经网络学习类型规则

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

设计了关于修改的语法

1. *Edits* → *Edit*; *Edits* | `end`
2. *Edit* → *Insert* | *Modify*
3. *Insert* → `insert(`⟨*HLStatement*⟩`)`
4. *Modify* → `modify(`
   ⟨*ID of an AST Node with a NTS*⟩,
   ⟨*the same NTS as the above NTS*⟩`)`
5. ⟨*Any NTS in HL*⟩ →
   `copy(`⟨*ID of an AST Node with the same NTS*⟩`)`
   | ⟨*The original production rules in HL*⟩
6. ⟨*HLIdentifier*⟩ → `placeholder`
   | ⟨*Identifiers in the training set*⟩

保证修改之后的代码是语法正确的

修复方法Recoder [ESEC/FSE’21]

在玲珑框架下，采用TreeGen来构造概率模型，并用修改操作定义了程序空间

Table 2: Comparison without Perfect Fault Localization

Project	jGenProg	HDRepair	Nopol	CapGen	SketchFix	FixMiner	SimFix	TBar	DLFix	PraPR	AVATAR	Recoder
Chart	0/7	0/2	1/6	4/4	6/8	5/8	4/8	**9/14**	5/12	4/14	5/12	8/14
Closure	0/0	0/7	0/0	0/0	3/5	5/5	6/8	8/12	6/10	12/62	8/12	**17/31**
Lang	0/0	2/6	3/7	5/5	3/4	2/3	**9/13**	5/14	5/12	3/19	5/11	**9/15**
Math	5/18	4/7	1/21	12/16	7/8	12/14	14/26	**18/36**	12/28	6/40	6/13	15/30
Time	0/2	0/1	0/1	0/0	0/1	1/1	1/1	1/3	1/2	0/7	1/3	**2/2**
Mockito	0/0	0/0	0/0	0/0	0/0	0/0	0/0	1/2	1/1	1/6	**2/2**	**2/2**
Total	5/27	6/23	5/35	21/25	19/26	25/31	34/56	42/81	30/65	26/148	27/53	**53/94**
P(%)	18.5	26.1	14.3	**84.0**	73.1	80.6	60.7	51.9	46.2	17.6	50.9	56.4

In the cells, x/y:x denotes the number of correct patches, and y denotes the number of patches that can pass all the test cases.

神经网络修复四年多来首次超过传统修复的效果

录用于ESEC/FSE21，是该会议上引用前三的论文

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

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

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare/Tacs [ICSE23/Submitted]
- 引导神经网络学习类型规则

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：

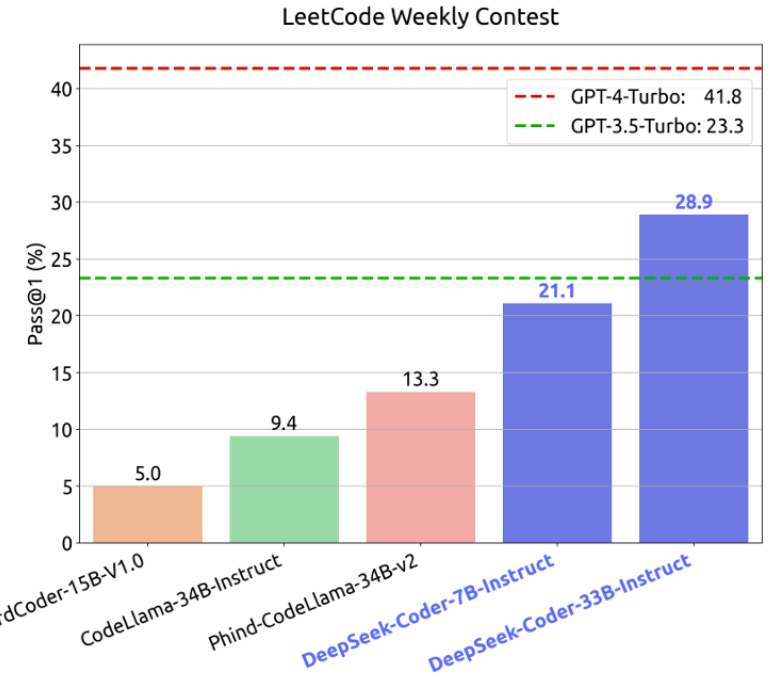
- 从文件中挖掘“声明–使用”关系
- 对文件排序，保证声明出现在使用之前

GrammarT5 [ICSE24]

Natural-Language-Based Code Generation									
Models	**Concode**			**Conala**		**Django**		**MBPP**	**MathQA**
Metric	**BLEU**	**EM**	**C-BLEU**	**BLEU**	**EM**	**BLEU**	**EM**	**pass@80**	**pass@80**
TreeGen + Grape(35M)	26.45	17.60	30.05	20.16	2.80	75.86	77.30	2.00	26.58
GPT-C(110M)	30.85	19.85	33.10	30.32	4.80	72.56	68.91	10.40	58.94
CodeGPT-adapted(110M)	35.94	20.15	37.27	31.04	4.60	71.24	72.13	12.60	55.90
CoTexT(220M)	19.19	19.72	38.13	31.45	6.20	75.91	78.43	14.00	58.18
PLBART(220M)	36.69	18.75	38.52	32.44	5.10	72.81	79.12	12.00	57.25
CodeT5-small(60M)	38.13	21.55	41.39	31.23	6.00	76.91	81.77	19.20	61.58
CodeT5-base(220M)	40.73	22.30	43.2	38.91	8.40	81.40	84.04	24.00	71.52
CodeT5-large(770M)	**42.66**	22.65	45.08	39.96	7.40	82.11	83.16	32.40	83.14
Unixcoder(110M)	38.73	22.65	40.86	36.09	10.20	78.42	75.35	22.40	70.16
GrammarT5-small(60M)	38.68	21.25	41.62	39.18	8.00	81.20	82.77	26.00	84.91
GrammarT5-base(220M)	42.30	**24.75**	**45.38**	**41.42**	**10.40**	**82.20**	**84.27**	**33.20**	**87.46**

用同样的训练集在CodeT5-base上微调
效果基本超过CodeT5-large

DeepSeek-Coder [arxiv24]



- 和DeepSeek合作推出
- 应用学习“声明-使用”关系的技术
- 我国首个达到国际顶尖水平的开源代码模型

GrammarCoder [ACL25-Finding]

Model	HumanEval(+)	MBPP(+)
Original		
DeepSeek-Coder-1.3B-Base	34.8 (28.7)	56.7 (47.9)
Qwen2.5-1.5B-Base	37.2 (32.9)	60.2 (49.6)
Normal Token-Based CPT		
DeepSeek-Coder-1.3B-Base (CPT)	43.9 (39.6)	61.4 (51.3)
Qwen2.5-1.5B-Base (CPT)	50.6 (42.7)	60.3 (51.1)
Grammar-Based CPT		
GrammarCoder-1.3B-Base	**63.4** (57.3)	**68.3 (56.9)**
GrammarCoder-1.5B-Base	**63.4 (59.1)**	64.8 (55.3)

和快手合作推出

当时在HumanEval和MBPP上效果最佳的1.xB模型

基于DeepSeek-Coder和Qwen2.5，用OpenCoder训练集微调

GrammarCoder [ACL25-Finding]

Model	HumanEval	HumanEval+	MBPP	MBPP+
DeepSeek-Coder-1.3B-Instruct (Guo et al., 2024)	65.9	60.4	64.3	54.8
Qwen2.5-1.5B-Instruct (Team, 2024)	61.6	49.4	63.2	55.6
OpenCoder-1.5B-Instruct (Huang et al., 2024)	72.5	67.7	72.7	**61.9**
Yi-Coder-1.5B-Chat (AI et al., 2025)	67.7	63.4	68.0	59.0
Phi-3-Mini-4K-3.8B-Instruct (Abdin et al., 2024)	64.6	59.1	65.9	54.2
CodeGemma-7B-Instruct (Team et al., 2024)	60.4	51.8	70.4	56.9
GrammarCoder-1.3B-Instruct	70.7	64.0	71.2	58.7
GrammarCoder-1.5B-Instruct	**73.2**	**68.3**	**73.3**	61.1

和快手合作推出
当时在HumanEval和MBPP上效果最佳的1.xB模型
基于DeepSeek-Coder和Qwen2.5，用OpenCoder训练集微调

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

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

Grape [IJCAI22]
- 引导神经网络学习文法规则

Tare/Tacs [ICSE23/Submitted]
- 引导神经网络学习类型规则

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

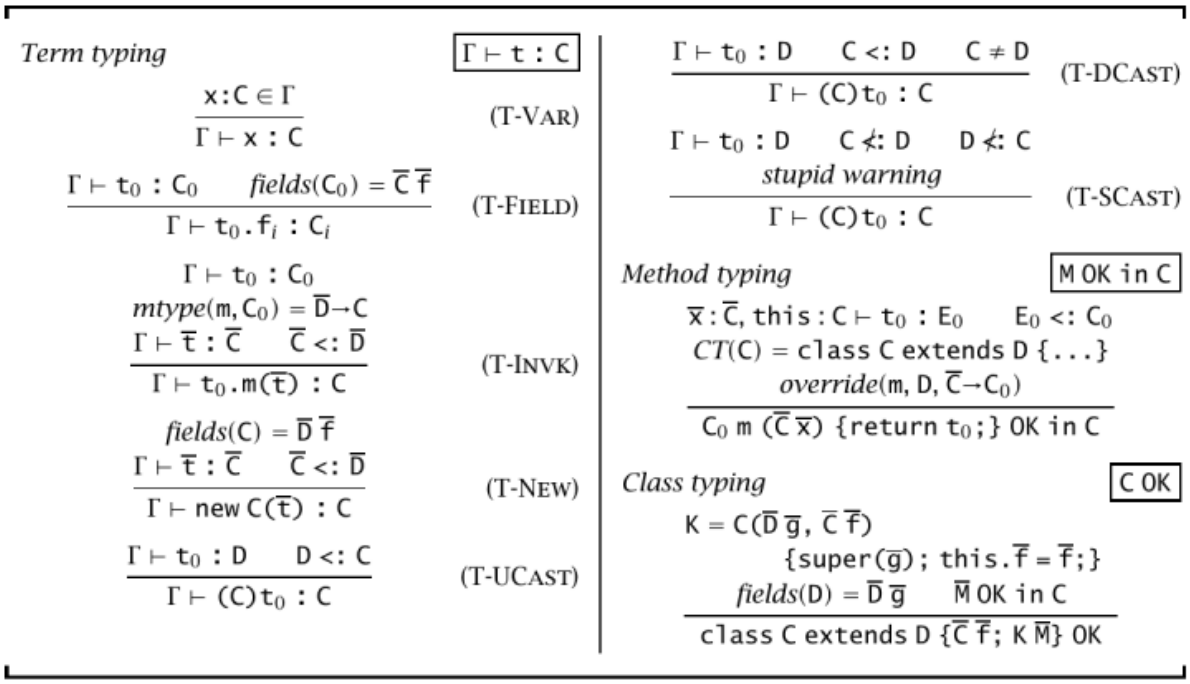
ET [APRCOMP24]
- 应用到修复，获国际修复大赛全球第一名

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, 投稿中论文］

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

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

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

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

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

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

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

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

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

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

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

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

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

类型推导的过程通常也定义了程序结构

$$\frac{x : t \in \Gamma}{\Gamma \vdash x : t}\ (\text{T-Var}) \qquad \frac{\Gamma, x : t_1 \vdash p : t_2}{\Gamma \vdash (\lambda x : t_1.\ p) : t_1 \to t_2}\ (\text{T-Abs}) \qquad \frac{\Gamma \vdash p_1 : t_1 \to t_2 \quad \Gamma \vdash p_2 : t_1}{\Gamma \vdash p_1\ p_2 : t_2}\ (\text{T-App})$$

$$\dfrac{\dfrac{\dfrac{x : \mathsf{bool} \in \{\mathsf{empty}, x : \mathsf{bool}\}}{\mathsf{empty}, x : \mathsf{bool} \vdash x : \mathsf{bool}}\ (\text{T-Var})}{\mathsf{empty} \vdash \lambda x : \mathsf{bool}.\ x : \mathsf{bool} \to \mathsf{bool}}\ (\text{T-Abs}) \qquad \dfrac{\cdots}{\mathsf{empty} \vdash \mathsf{true} : \mathsf{bool}}\ (\ldots)}{\mathsf{empty} \vdash (\lambda x : \mathsf{bool}.\ x)\ \mathsf{true} : \mathsf{bool}}\ (\text{T-App})$$

T-APP定义程序结构为$p_1\ p_2$
T-ABS定义p_1为$(\lambda x{:}\ t_1.\ p_3)$
T-VAR定义p_3为单个变量
……

引导学习类型规则–方法

将程序编码为类型规则的应用序列

$$\frac{x : t \in \Gamma}{\Gamma \vdash x : t}\ (\text{T-Var}) \qquad \frac{\Gamma, x : t_1 \vdash p : t_2}{\Gamma \vdash (\lambda x : t_1.\ p) : t_1 \to t_2}\ (\text{T-Abs}) \qquad \frac{\Gamma \vdash p_1 : t_1 \to t_2 \quad \Gamma \vdash p_2 : t_1}{\Gamma \vdash p_1\ p_2 : t_2}\ (\text{T-App})$$

$$\dfrac{\dfrac{\dfrac{x : \mathsf{bool} \in \{\mathsf{empty}, x : \mathsf{bool}\}}{\mathsf{empty}, x : \mathsf{bool} \vdash x : \mathsf{bool}}\ (\text{T-Var})}{\mathsf{empty} \vdash \lambda x : \mathsf{bool}.\ x : \mathsf{bool} \to \mathsf{bool}}\ (\text{T-Abs}) \qquad \dfrac{\cdots}{\mathsf{empty} \vdash \mathsf{true} : \mathsf{bool}}\ (\ldots)}{\mathsf{empty} \vdash (\lambda x : \mathsf{bool}.\ x)\ \mathsf{true} : \mathsf{bool}}\ (\text{T-App})$$

T-APP, T-ABS, T-VAR, ……

引导学习类型规则-方法

类型规则可以看做是一个构造逻辑，或者经过简单改造可以形成构造逻辑。

$$\frac{x : t \in \Gamma}{\Gamma \vdash x : t}\ (\text{T-Var}) \qquad \frac{\Gamma, x : t_1 \vdash p : t_2}{\Gamma \vdash (\lambda x : t_1.\ p) : t_1 \to t_2}\ (\text{T-Abs}) \qquad \frac{\Gamma \vdash p_1 : t_1 \to t_2 \quad \Gamma \vdash p_2 : t_1}{\Gamma \vdash p_1\ p_2 : t_2}\ (\text{T-App})$$

$$\dfrac{\dfrac{\dfrac{x : \mathsf{bool} \in \{\mathsf{empty}, x : \mathsf{bool}\}}{\mathsf{empty}, x : \mathsf{bool} \vdash x : \mathsf{bool}}\ (\text{T-Var})}{\mathsf{empty} \vdash \lambda x : \mathsf{bool}.\ x : \mathsf{bool} \to \mathsf{bool}}\ (\text{T-Abs}) \qquad \dfrac{\cdots}{\mathsf{empty} \vdash \mathsf{true} : \mathsf{bool}}\ (\ldots)}{\mathsf{empty} \vdash (\lambda x : \mathsf{bool}.\ x)\ \mathsf{true} : \mathsf{bool}}\ (\text{T-App})$$

程序生成转变为等价于证明$\exists p, \mathrm{welltyped}(p)$
因为构造逻辑的特点，证明过程一定会给出p

实际上，不限于类型规则，所有CHC规则都可以改造成构造逻辑。

引导学习类型规则-效果

基于CodeT5微调类型规则表示

效果显著超过原始模型和约束解码

Table 2. Comparison of Accuracy and Compilation Error Rate between CodeT5 and TyFlow on SuFu and Java Datasets

Language	Model	pass@1 (%)↑	pass@10 (%)↑	FSP↓	CER (%)↓
SuFu	CodeT5-220M	24.14	32.76	7.03	83.10
	T5Gemma2-2B	29.31	37.93	6.69	61.21
	TyFlow-220M	37.93	46.55	5.48	0.00
	TyFlow-2B	**43.10**	**50.00**	**5.03**	**0.00**
Java	CodeT5-220M	10.45	20.90	8.19	38.51
	T5Gemma2-2B	17.91	35.82	6.99	15.22
	TyFlow-220M	11.94	28.36	7.94	3.52
	TyFlow-2B	**23.19**	**40.30**	**6.76**	**3.12**

↑ Higher is better; ↓ Lower is better; **bold** indicates best value per column.
pass@k measures the proportion of problems solved by at least one of the top-k candidates (verified by test cases); FSP is the average rank of the first correct candidate; CER is the compilation error rate. See Sec. 6.1.3 for details.

For Peer Review

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

能否用神经网络加强符号方法？

神经符号融合软件的终极形态是什么样？

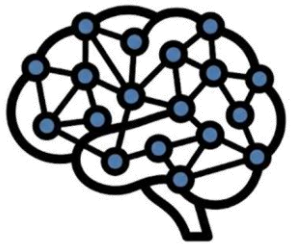
1. 保持神经符号融合

2. 只有神经网络

3. 只有符号系统

为什么符号系统是终极形态？

- 经过充分研究的具体问题上，理论最优算法的效率都是远远高于人类的。
  - 更不要说正确性保障等其他方面的优势
- 为什么现在有部分问题大模型解得更好？
  - 因为在理论上这些问题还没有被研究清楚
- 如何能研究清楚？
  - 参照对象：大模型
  - 观察大模型是如何解决这些问题的
  - 提取一般性规律
  - 编写符号算法
  - 并且这些步骤都能交给大模型来做



另一个证据

JOURNAL ARTICLE

Why Expert Systems Fail

Michael Z. Bell

The Journal of the Operational Research Society, Vol. 36, No. 7 (Jul., 1985), pp. 613-619 (7 pages)

https://doi.org/10.2307/2582480 · https://www.jstor.org/stable/2582480

- 专家没空
- 专家无法表述规则
- 专家不想表述规则
- **没有**专家
- **缺乏常**识
- **知**识表示语言不够强
- 难以测试
- **用**户不信任

劳动力不够

符号智能失败的核心原因是没有足够多的专家来写规则，而非其他本质局限性。

今天发布的OpenAI研究员文章

- https://trinkle23897.github.io/learning-beyond-gradients/#zh

为什么 Heuristic Learning 以前没发展起来

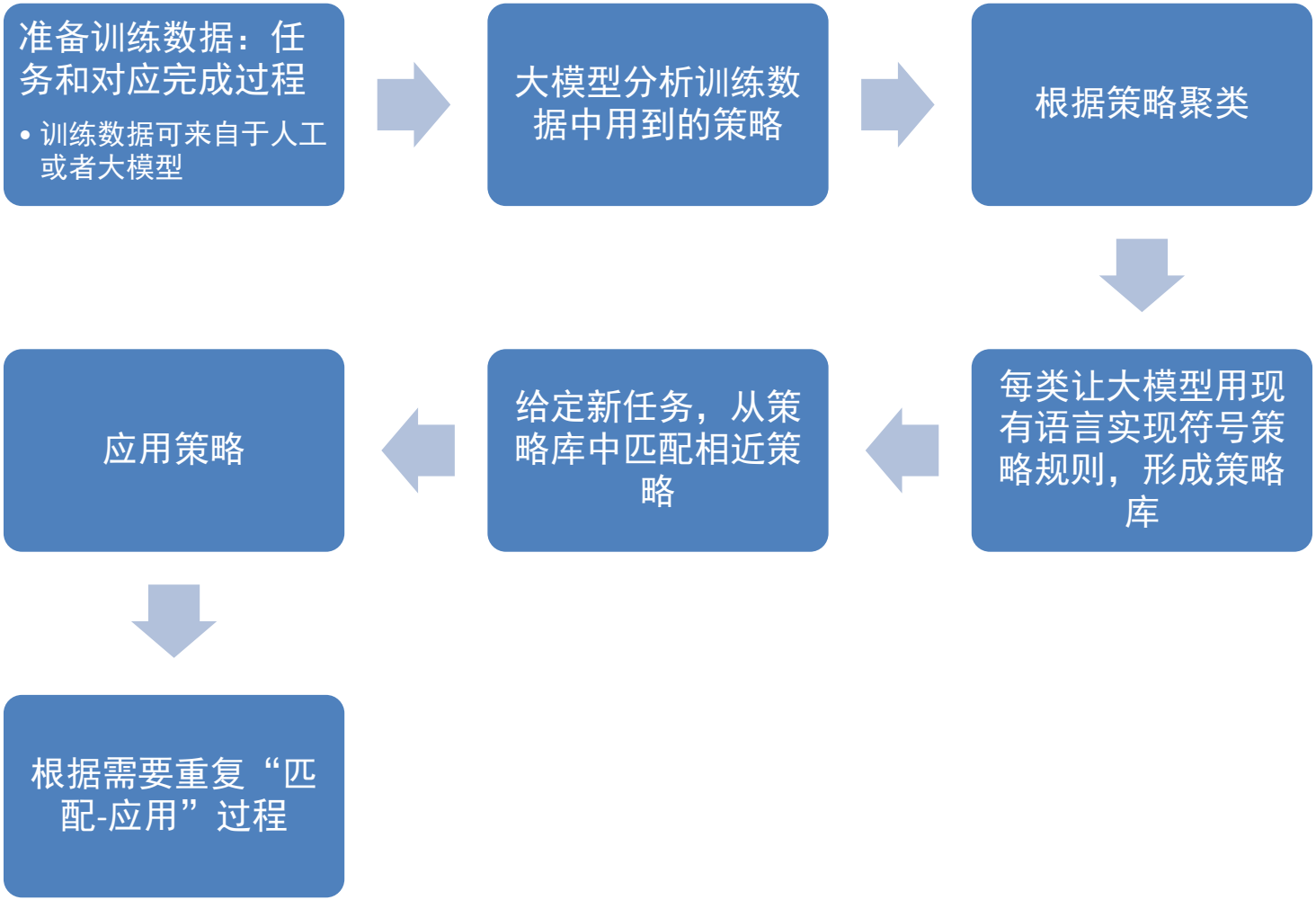
如果说 HL 的前身是专家系统、规则系统，那么在 coding agent 没发展起来之前，这玩意的维护成本十分高昂。

人类手工维护 heuristic 很容易变成这样：

```
今天加一条规则修 case A。
明天发现 case B 被修坏了。
后天再加一个 if。
大后天没人敢删了。
```

问题不在 heuristic 没用，在没人力能养得起。之前人力维护专家系统，有点像工业革命前手工纺纱：规模一大，稳定性和维护成本就压死人。纺织机改变的是产能曲线；coding agent 改变的是 heuristic 的维护曲线。它像一条可以输送智力的营养管道，可以持续浇灌一个 HS，让它自己迭代进化。

大模型辅助的符号策略学习流程



应用——自动定理证明

- 很多需要验证的场景无法使用大模型
  - **中小企业且需要保密**
- 收集大量已有证明
- 用上述框架提取Coq策略
- 结合策略和CoqHammer证明新的定理

		OpenAI GPT-5.2		*Deepseek-V3.2*	
	#Lemma	CoqHammer	CoqHammer +LLM2Ltac	CoqHammer	CoqHammer +LLM2Ltac
CompCert	4840	1022	**1269** (↑ **24.17%**)	1022	**1276** (↑ **24.85%**)
Ext-Lib	187	79	**100** (↑ **26.58%**)	79	**99** (↑ **25.32%**)
Coq-Art	842	411	**482** (↑ **17.27%**)	411	**498** (↑ **21.17%**)
Vfa	330	80	**99** (↑ **23.75%**)	80	**99** (↑ **23.75%**)
Total	6199	1592	**1950** (↑ **22.49%**)	1592	**1972** (↑ **23.87%**)

	Claude Code	Claude Code + CoqHammer	Claude Code + LLM2Ltac
#Thm	71	101	111 (↑ 9.90%)
Tokens	59899928	38346691	34317809 (↓ 10.51%)

应用——代码自动优化

- 大模型难以对大型代码库形成有效优化
  - **难以知道什么位置应该用什么优化策略**
- 从大量历史优化中提取静态分析规则
- 静态分析规则扫描代码库，匹配优化位置和策略
- 提示策略，让大模型进行优化

Repo	Perf Improvement ↑		# Test Cases	
	Max	Avg	↑ ≥ 5%	↑ ≥ 10%
RocksDB	5.04%	2.41%	1/2	0/2
Redis	6.14%	1.68%	3/21	0/21
gRPC	35.48%	3.10%	2/16	1/16
LevelDB	218.07%	10.40%	2/22	1/22
spdlog	20.00%	3.27%	6/45	4/45

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
  - **=>：将大模型中的代码表示换成证明表示**
  - **<=：大模型辅助的符号策略学习**