Neural Code Generation Models with Programming Language Knowledge

Yingfei Xiong

Peking University

Training Code LLM

- More parameters, more effective, more expensive

Large model costs millions and even billions to train

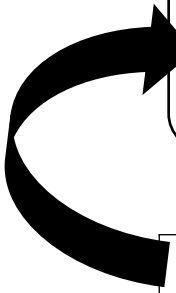
- Can we reduce the needed parameters, or improve performance without using more parameters?

Neural Code Generation: Predicting BPE tokens one by one

return the maximum of two integers

Neural Model

=	0.40
+	0.25
= a	0.15
is_	0.06
…	



if (a > b) then r	0.30
if (a == b) then	0.27
a += 10; b	0.26
if (a > b) then b =	0.21
…	

Beam search to find the most likely program

Problem: Ignoring PL knowledges

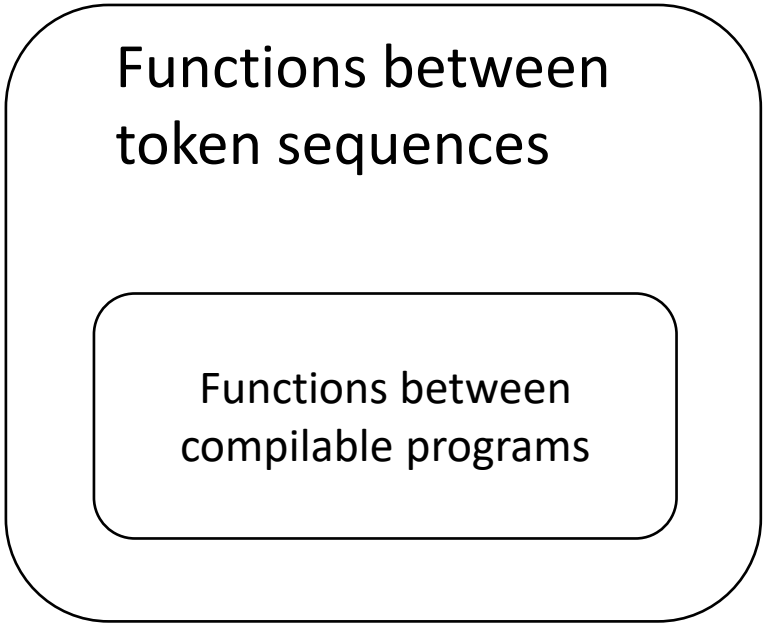
- Program languages have knowledge such as grammar, type system, and semantics, ignoring them making learning more difficult.

Syntactic constraint: ()+5(illegal

Type constraint: 1+true illegal

Semantic constraint:

use without initialization illegal



Hypothesis space (PAC learning)

Problem: Ignoring PL knowledges

- Correct code generation requires PL knowledges

```
bool and(bool a, bool b) {
   ________________
}
```

assignment is common and should be used

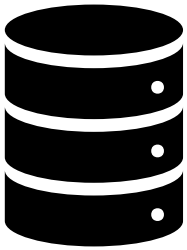
NN that does not know types

all parameters are Boolean so ‘if’ is more likely

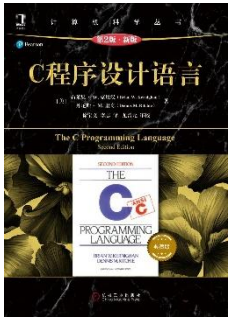
NN that knows types

Problem: Ignoring PL knowledges

- It is hard to learn PL knowledge by end-to-end training.



Huge code corpus



C grammar, type system, semantics

Can we guide neural network to learn PL knowledge?

Overview

TreeGen [AAAI20]

- Using transformer to implement L2S
- The first transformer-based code generator

Grape [IJCAI22]

- Guiding NN to learn grammar rule definitions

Tare [ICSE23]

- Guide NN to learn typing rules

GrammarT5 [ICSE24]

- Integrating L2S with pretraining`

Implements

L2S Framework [TOSEM22]

- Representing code as grammar rule sequences
- Ensuring syntactic correctness
- Allowing easy pruning

Apply

ACS [ICSE17]

- First program repair approach whose precision > 70%

Recoder [FSE23]

- First neural program repair approach outperforming traditional approaches

OCoR [ASE20]

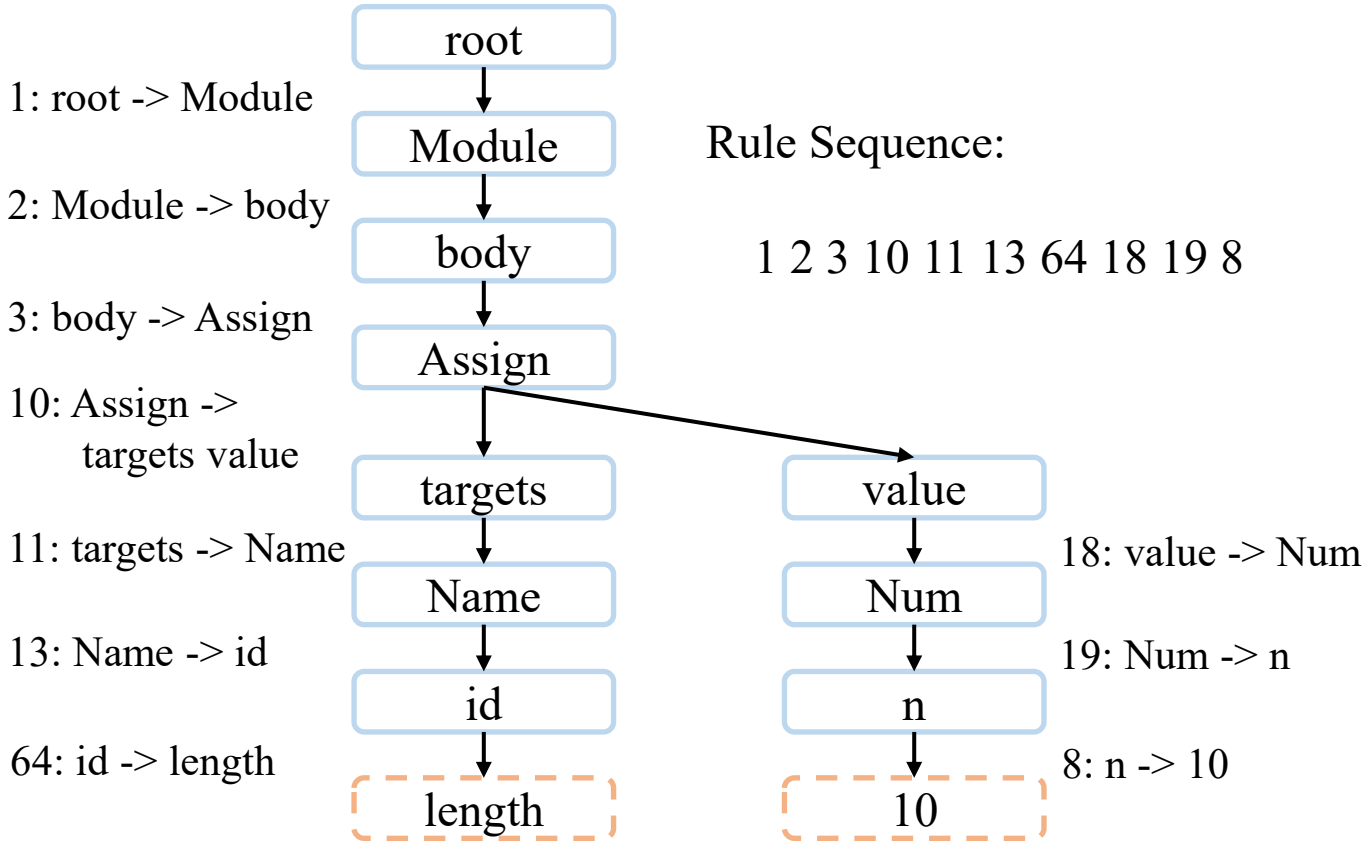
- Code search engine significantly outperforming existing ones

LEAM [ASE22]

- Mutation generation engine that significantly outperforming existing ones

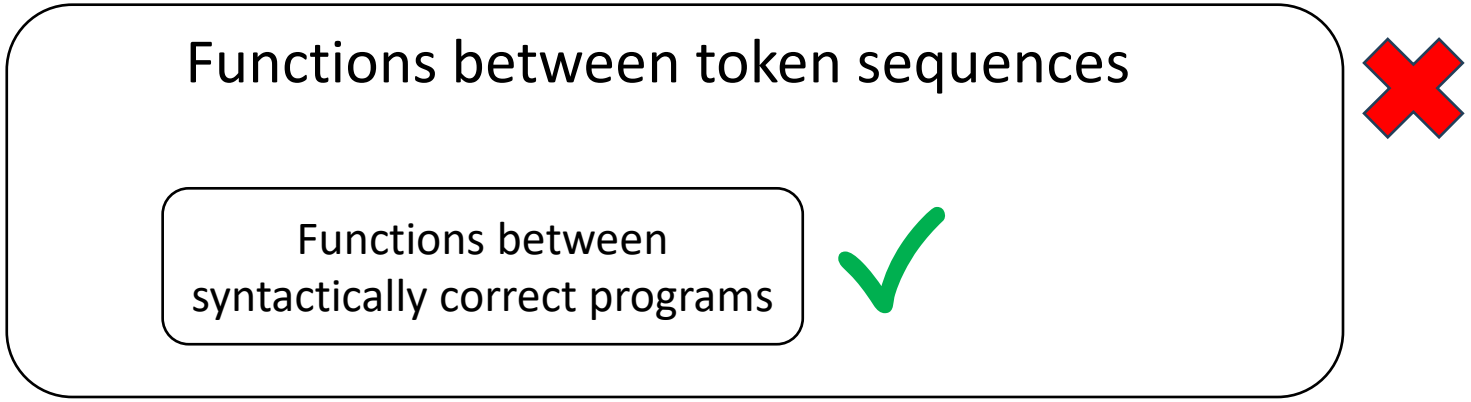
Representing programs as grammar rule lists [TOSEM22]

- Traverse the AST and record the rule ID used to expand each node



Benefits: Overcoming the problems

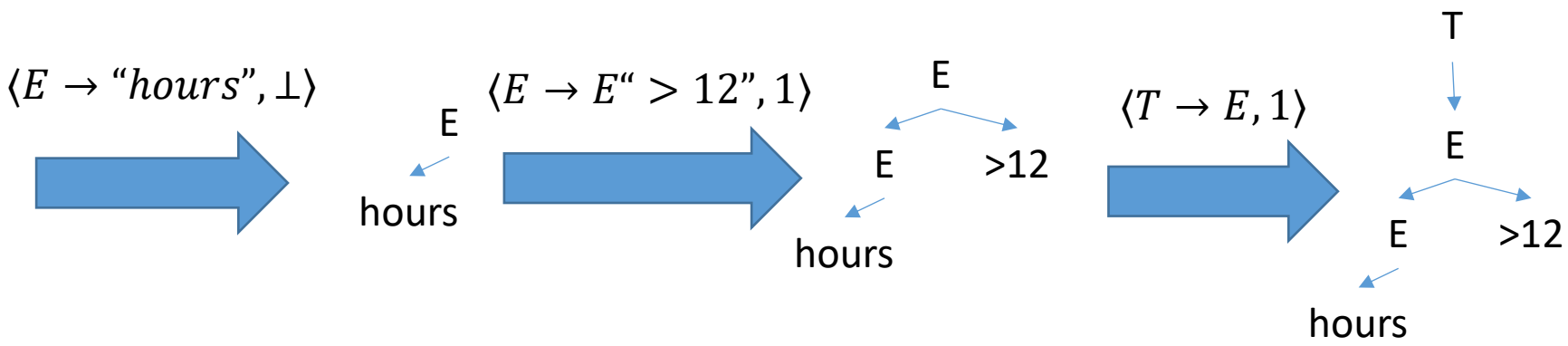
- Generate only syntactically correct program
  - if (dp[j][i] == -1) {
    dp[j][i] = newval;
    }}}}}}}
- Much smaller space for more effective learning



Hypothesis space (PAC learning)

Benefits:
Controlling the generation Order

- Different generation order may lead to significantly different performance
- Generating order can be controlled
  - by extending context-free grammar into expansion grammar



Benefits:
Pruning partial solutions

- Design static program analysis for pruning infeasible partial solutions
- Analyzing partial programs generated by LLM is hard
  - BPE tokens are not even scannable
  - Not to mention parsing
- Analyzing partial AST is much easier
  - if (BoolExpr) then x else x
  - 1 + “x” + Expr

Overview

TreeGen [AAAI20]

- Using transformer to implement L2S
- The first transformer-based code generator

Grape [IJCAI22]

- Guiding NN to learn grammar rule definitions

Tare [ICSE23]

- Guide NN to learn typing rules

GrammarT5 [ICSE24]

- Integrating L2S with pretraining`

Implements

L2S Framework [TOSEM22]

- Representing code as grammar rule sequences
- Ensuring syntactic correctness
- Allowing easy pruning

Apply

ACS [ICSE17]

- First program repair approach whose precision > 70%

Recoder [FSE23]

- First neural program repair approach outperforming traditional approaches

OCoR [ASE20]

- Code search engine significantly outperforming existing ones

LEAM [ASE22]

- Mutation generation engine that significantly outperforming existing ones

Using Transformer to implement L2S [AAAI20]

- The earliest work that applies Transformer for code generation
  - TreeGen: a Transformer model designed for grammar rule sequences

	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

TreeGen has been widely applied to decompilation, program repair, code search, automating editing by different researchers

Overview

TreeGen [AAAI20]

- Using transformer to implement L2S
- The first transformer-based code generator

Grape [IJCAI22]

- Guiding NN to learn grammar rule definitions

Tare [ICSE23]

- Guide NN to learn typing rules

GrammarT5 [ICSE24]

- Integrating L2S with pretraining`

Implements

L2S Framework [TOSEM22]

- Representing code as grammar rule sequences
- Ensuring syntactic correctness
- Allowing easy pruning

Apply

ACS [ICSE17]

- First program repair approach whose precision > 70%

Recoder [FSE23]

- First neural program repair approach outperforming traditional approaches

OCoR [ASE20]

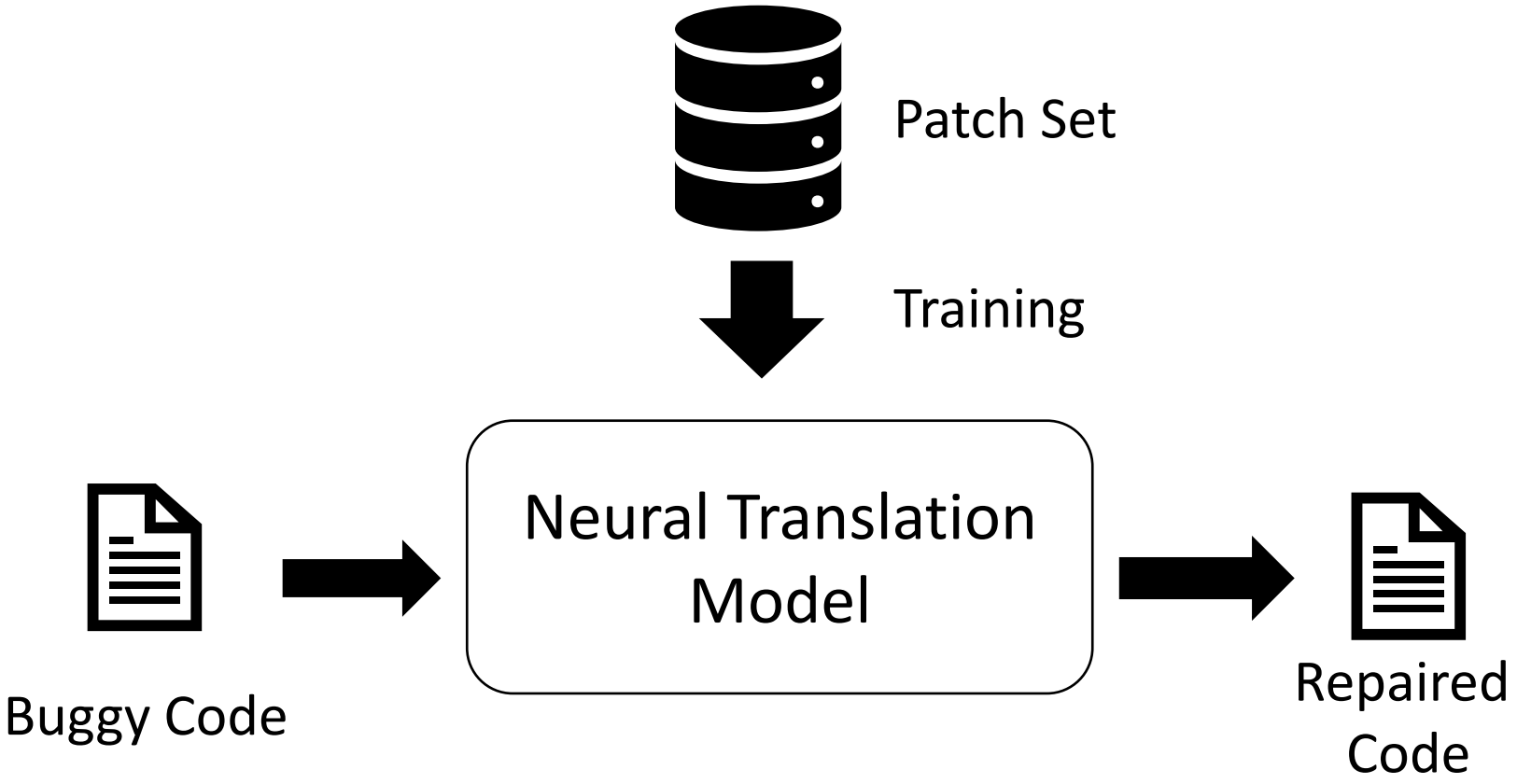
- Code search engine significantly outperforming existing ones

LEAM [ASE22]

- Mutation generation engine that significantly outperforming existing ones

Existing Neural Program Repair

- Treating a patch as a pair of code



A finding in bidirectional transformation [Models'11 MIP]

- State-based representation is ineffective

```
cfa.createEdge(fromNode, Branch.UNCOND, finallyNode);
```

```
cfa.createEdge(fromNode, Branch.ON_EX, finallyNode);
```

1. Need to learn diff during training
2. Repr is long (13 tokens)

- Delta-based representation is more desirable

```
modify(9, ON_EX)
```

1. Change is directly given
2. Repr is short (3 tokens)

A grammar of change

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*⟩

Ensuring the changed code is still syntactically correct.

Recoder [ESEC/FSE'21]

- TreeGen for generating changes
- Neural program repair outperformed traditional approaches for the first time

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.

Overview

TreeGen [AAAI20]

- Using transformer to implement L2S
- The first transformer-based code generator

Grape [IJCAI22]

- Guiding NN to learn grammar rule definitions

Tare [ICSE23]

- Guide NN to learn typing rules

GrammarT5 [ICSE24]

- Integrating L2S with pretraining`

Implements

L2S Framework [TOSEM22]

- Representing code as grammar rule sequences
- Ensuring syntactic correctness
- Allowing easy pruning

Apply

ACS [ICSE17]

- First program repair approach whose precision > 70%

Recoder [FSE23]

- First neural program repair approach outperforming traditional approaches

OCoR [ASE20]

- Code search engine significantly outperforming existing ones

LEAM [ASE22]

- Mutation generation engine that significantly outperforming existing ones

LEAM [ASE'22 Distinguished]

- From Junjie Chen and Lingming Zhang's group
- Exchange the input and output of Recoder
- Program Repairer -> Bug Seeder

Table 4: Overall effectiveness in mutation-based FL

FL	Tech.	Top-1	Top-3	Top-5	MFR	MAR
Metallaxis	Major	35	92	114	9.56	12.42
	PIT	56	102	128	8.16	11.83
	DM	19	47	98	16.64	20.65
	LEAM	**118**	**182**	**188**	**3.86**	**4.57**
MUSE	Major	35	89	111	10.99	13.11
	PIT	52	97	124	9.15	11.72
	DM	18	53	94	18.70	22.47
	LEAM	**126**	**181**	**189**	**3.88**	**5.05**

Overview

TreeGen [AAAI20]

- Using transformer to implement L2S
- The first transformer-based code generator

Grape [IJCAI22]

- Guiding NN to learn grammar rule definitions

Tare [ICSE23]

- Guide NN to learn typing rules

GrammarT5 [ICSE24]

- Integrating L2S with pretraining`

Implements

L2S Framework [TOSEM22]

- Representing code as grammar rule sequences
- Ensuring syntactic correctness
- Allowing easy pruning

Apply

ACS [ICSE17]

- First program repair approach whose precision > 70%

Recoder [FSE23]

- First neural program repair approach outperforming traditional approaches

OCoR [ASE20]

- Code search engine significantly outperforming existing ones

LEAM [ASE22]

- Mutation generation engine that significantly outperforming existing ones

Limit of L2S

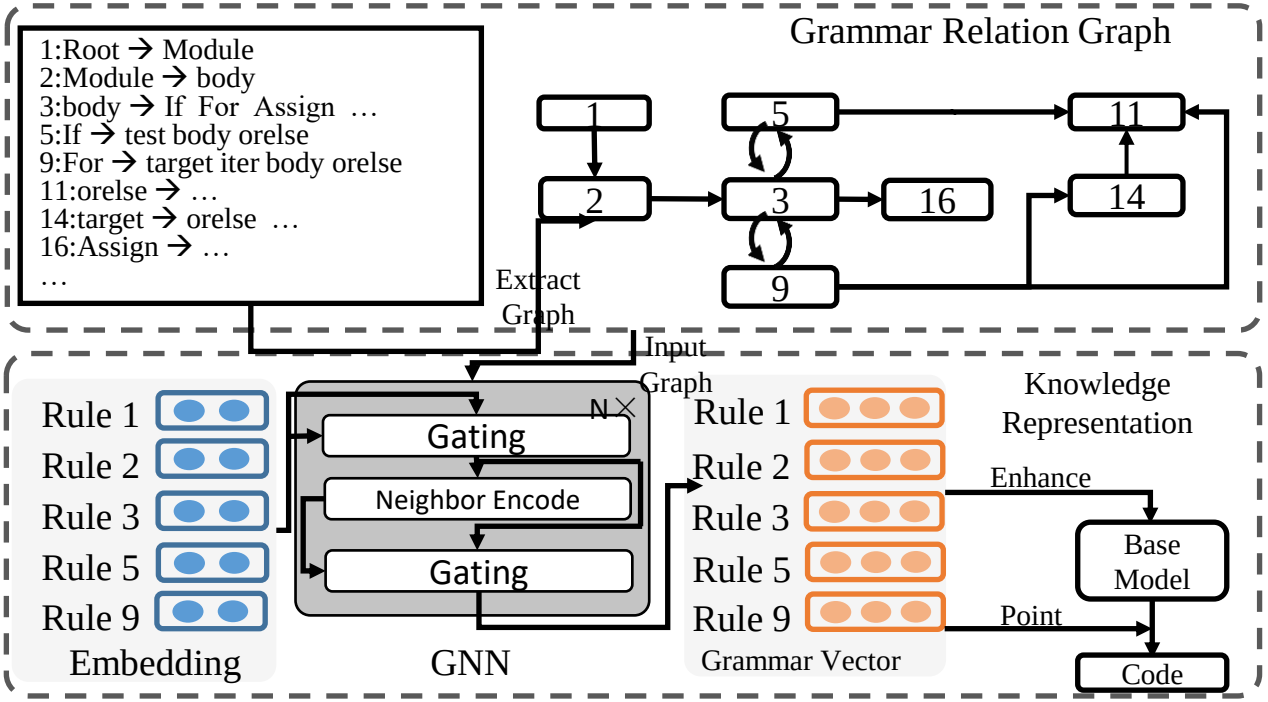
- Force syntactical and other constraint from outside
- NN does not learn their definitions

ifstmt -> ‘if’ ‘(‘ boolExpr ‘)’ stmt	10
whilestat -> ‘while’ ‘(‘ boolExpr ‘)’ stmt	11
boolExpr -> andExpr	12
boolExpr -> orExpr	13

Grammar rules are encoded as numbers without content.
NN could predict impossible sequences such as 10, 11.

Learning Grammar Rules [IJCAI22]

- Guide the NN to learn grammar definitions
- Word2Vec: assign each token a vector
- Grape: assign each grammar rule a vector, learned with its definition structure



Learning Grammar Rules [IJCAI22]

- Improve the performance of TreeGen up to 5 percentage points
- Outperforms larger pre-training models

		Code Generation					Semantic Parsing		Regex Synthesis
	Method	**HearthStone**			**Django**	**Concode**	**Atis**	**Job**	**StrReg**
	Metric	**StrAcc**	**BLEU**	**Acc+**	**StrAcc**	**StrAcc**	**ExeAcc**	**ExeAcc**	**DFAAcc**
	KCAZ13 [Kwiatkowski *et al.*, 2013]	-	-	-	-	-	89.0	-	-
	WKZ14 [Wang *et al.*, 2014]	-	-	-	-	-	91.3	90.7	-
Neural Networks	SEQ2TREE [Dong and Lapata, 2016]	-	-	-	-	-	84.6	90.0	-
Neural Networks	ASN+SUPATT [Rabinovich *et al.*, 2017]	22.7	79.2	-	-	-	85.9	**92.9**	-
Neural Networks	TRANX [Yin and Neubig, 2018]	-	-	-	73.7	-	86.3	90.0	-
Neural Networks	Iyer-Simp+200 idoms [Iyer *et al.*, 2018]	-	-	-	-	12.20	-	-	-
Neural Networks	GNN-Edge [Shaw *et al.*, 2019]	-	-	-	-	-	87.1	-	-
Neural Networks	SoftReGex [Park *et al.*, 2019]	-	-	-	-	-	-	-	28.2
Neural Networks	TreeGen [Sun *et al.*, 2020]	30.3±1.061	80.8	33.3	76.4	16.6	89.6±0.329	91.5±0.586	22.5
	GPT-2 [Radford *et al.*, 2019]	16.7	71	18.2	62.3	17.3	84.4	92.1	24.6
	CodeGPT [Lu *et al.*, 2021]	27.3	75.4	30.3	68.9	**18.3**	87.5	92.1	22.49
	TreeGen + Grape	**33.6±1.255**	**85.4**	**36.3**	**77.3**	17.6	**92.16±0.167**	**92.55±0.817**	**28.9**

Parameters：TreeGen+Grape: 35M GPT-2、CodeGPT：110M

Overview

TreeGen [AAAI20]
- Using transformer to implement L2S
- The first transformer-based code generator

Grape [IJCAI22]
- Guiding NN to learn grammar rule definitions

Tare [ICSE23]
- Guide NN to learn typing rules

GrammarT5 [ICSE24]
- Integrating L2S with pretraining`

Implements

L2S Framework [TOSEM22]
- Representing code as grammar rule sequences
- Ensuring syntactic correctness
- Allowing easy pruning

Apply

ACS [ICSE17]
- First program repair approach whose precision > 70%

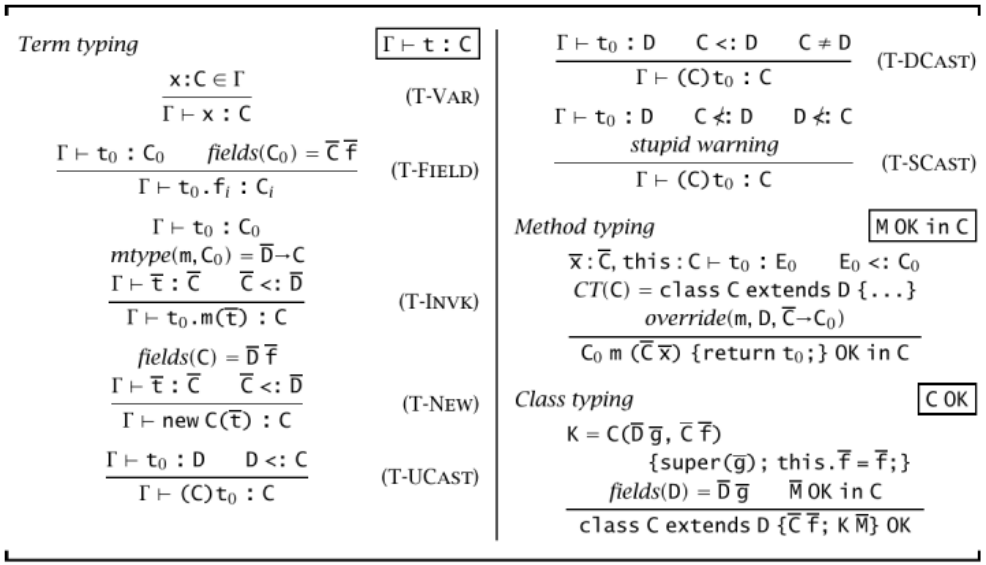
Recoder [FSE23]
- First neural program repair approach outperforming traditional approaches

OCoR [ASE20]
- Code search engine significantly outperforming existing ones

LEAM [ASE22]
- Mutation generation engine that significantly outperforming existing ones

Learning Typing Rules [ICSE23]

- Full type system is difficult to learn from data



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

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

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

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

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

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

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

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

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

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

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

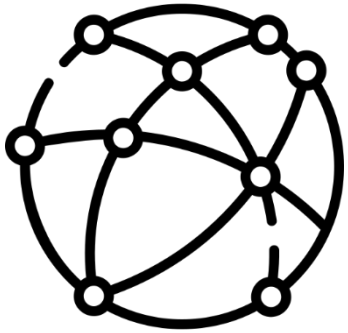
$$\frac{\begin{array}{c}K = C(\overline{D}\ \overline{g}, \overline{C}\ \overline{f})\ \{\mathsf{super}(\overline{g}); \mathsf{this}.\overline{f} = \overline{f};\} \\ \mathit{fields}(D) = \overline{D}\ \overline{g} \qquad \overline{M}\ \mathsf{OK\ in}\ C\end{array}}{\mathsf{class}\ C\ \mathsf{extends}\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

Figure 19-4: Featherweight Java (typing)

- Only 30%-40% programs generated by Recoder is typable

Learning Typing Rules [ICSE23]

- A single rule is much easier to learn
  - T-Graph: present the input of a typing rule to the NN
  - T-Grammar: force NN to predict the output of a typing rule



T-Graph: Representing typing relations
- types of AST nodes
- types of variables
- subtyping relations

T-Grammar:
E -> E && E becomes
[Bool]E -> [Bool]E && [Bool]E

Learning Typing Rules [ICSE23]

- Applying to program repair, forming Tare

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**

Tare+ExpressAPR(efficient patch validation tool) got the first place in the Java functional bug track of APR-COMP'24.

Overview

TreeGen [AAAI20]

- Using transformer to implement L2S
- The first transformer-based code generator

Grape [IJCAI22]

- Guiding NN to learn grammar rule definitions

Tare [ICSE23]

- Guide NN to learn typing rules

GrammarT5 [ICSE24]

- Integrating L2S with pretraining`

Implements

L2S Framework [TOSEM22]

- Representing code as grammar rule sequences
- Ensuring syntactic correctness
- Allowing easy pruning

Apply

ACS [ICSE17]

- First program repair approach whose precision > 70%

Recoder [FSE23]

- First neural program repair approach outperforming traditional approaches

OCoR [ASE20]

- Code search engine significantly outperforming existing ones

LEAM [ASE22]

- Mutation generation engine that significantly outperforming existing ones

A Era of LLMs

LLMs (=pretrained large models) exhibit superior performance

Can we use grammar-based representation in LLMs?

Challenges

- Big vocabulary
  - User-defined identifiers can be added to the grammar when the training set is small
  - Pre-training sets are too large
- Heterogeneous grammars
  - Existing models: One programming language
  - Pretraining models: Many programming languages
- Pretraining Tasks
  - Self-supervised training tasks are needed
  - Tasks are expected to guide the neural network to learn the grammar structure

Big vocabulary

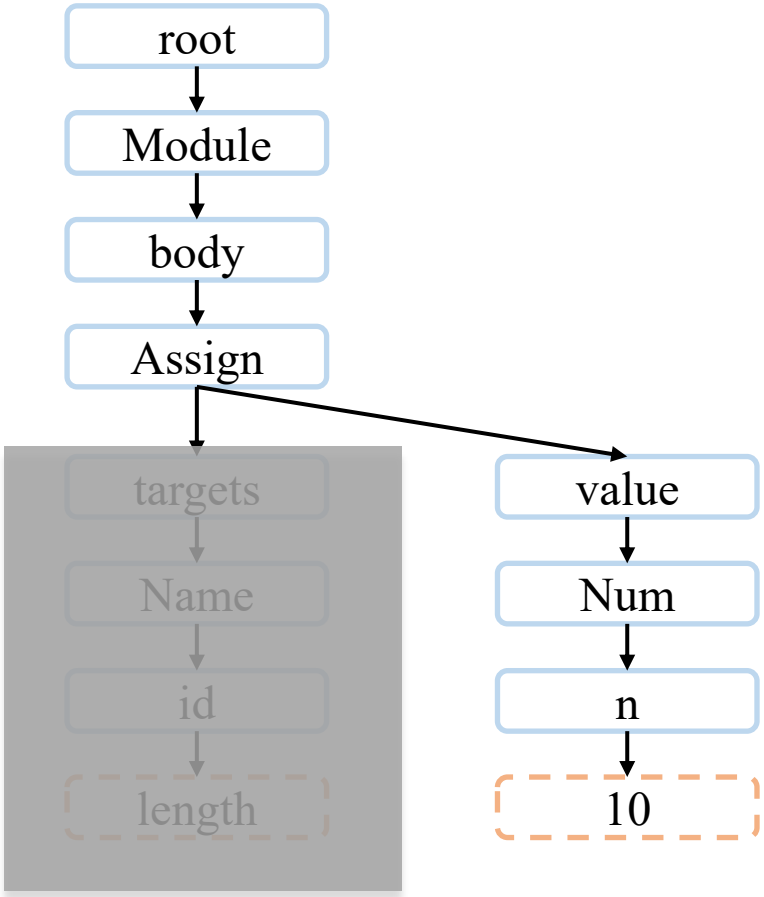
- Existing approaches
  - IDEN -> isodd | iseven
- Our approach
  - Using BPE (Byte Pair Encoding) to find a small set of subtokens
    - is, odd, even
  - Integrating them into the grammar
    - IDEN -> is IDEN | odd IDEN | even IDEN
      | #is | #odd | #even
    - # indicates the ending tokens
    - Leads to significantly shorter encoding than the standard sequence encoding
      - IDEN -> is IDEN | odd IDEN | even IDEN
        | ϵ

Heterogeneous grammars

- A hyper grammar that includes all grammars
  - Root -> Root@Python | Root@Java | …
- Experimentally has better performance than sharing some of non-terminals
  - While -> while ‘(‘ BoolExpr ‘)’ Statements
  - BoolExpr -> BoolExpr@Java | BoolExpr@C# | …

Pretraining Tasks

- Given a rule sequence, predicting the parent of a rule
  - 1 2 3 10 11 13 64 18 19 8
- Predicting some subtree of an AST



Learning Declarations

- Existing pre-training models sort files randomly
- LLMs may see a function or a variable before its declaration
- Dependency parsing:
  - Extract declaration-use relationship from files
  - Sort the files so that declarations appear before use

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**

Industrial Applications

EvalPlus Tests

#	Model	pass@1
1	GPT-4-Turbo (April 2024)	86.6
2	DeepSeek-Coder-V2-Instruct	82.3
3	GPT-4-Turbo (Nov 2023)	81.7
4	GPT-4 (May 2023)	79.3
5	CodeQwen1.5-7B-Chat	78.7
6	claude-3-opus (Mar 2024)	77.4
7	DeepSeek-Coder-33B-instruct	75
8	OpenCodeInterpreter-DS-33B	73.8
9	WizardCoder-33B-V1.1	73.2
10	Artigenz-Coder-DS-6.7B	72.6

One of the best open source LLM for code

Development leaded by my Ph.D. student Qihao Zhu

Dependency parsing was applied

Work-in-progress

Cooperating with a large company to train a LLM with grammar-based representation.

Industrial Applications

		Total (574 bugs)		
Organization	Model	Plausible @1	AST Match @1	Cost
Anthropic	claude-3-5-sonnet-20240620	**39,1%**	11,7%	$88.11
Google	gemini-1.5-pro-002	33,2%	**14,3%**	$30.98
OpenAI	gpt-4o-2024-08-06	31,7%	8,3%	$30.51
Google	gemini-1.5-pro-001	28,2%	12,5%	$78.65
Meta	llama-3.1-405b-instruct	27,0%	7,6%	$29.28
DeepSeek	deepseek-v2.5	25,1%	6,5%	$19.73
Alibaba Cloud	qwen-2.5-72b-instruct	24,2%	6,6%	$4.74
Mistral	mistral-large-2407	23,0%	6,6%	$47.70
Nvidia	llama-3.1-nemotron-70b-instruct	21,4%	3,4%	$4.08
OpenAI	o1-preview-2024-09-12*	N/A	N/A	N/A

One of the best open source LLM for code

Development leaded by my Ph.D. student Qihao Zhu

Dependency parsing was applied

Work-in-progress

Cooperating with a large company to train a LLM with grammar-based representation.

Conclusion

- Anxiety: what should we software researchers do if LLMs learn everything by themselves?
- LLMs do not learn programming languages knowledge by themselves
- Guiding them to learn improves their performance
- Future: more genetic ways to learn more software knowledge