Next Syntactic-Unit Code Completion and Applications

Anh Tuan Nguyen
Axon

Aashish Yadavally
Department of Computer Science
University of Texas at Dallas

Tien N. Nguyen
Department of Computer Science
University of Texas at Dallas

Introduction

- Code completion is an important feature in an IDE to improve developers' productivity.

Introduction

- Code completion is an important feature in an IDE to improve developers' productivity.
- Recent research on code completion focuses mostly on next-token prediction, or aim to complete entire statements or blocks of code.

Introduction

- Code completion is an important feature in an IDE to improve developers' productivity.
- Recent research on code completion focuses mostly on next-token prediction, or aim to complete entire statements or blocks of code.
- In this work, we specifically focus on synthesizing *syntactic units* at ***any*** location.

Introduction

- Code completion is an important feature in an IDE to improve developers' productivity.
- Recent research on code completion focuses mostly on next-token prediction, or aim to complete entire statements or blocks of code.
- In this work, we

```
1  ...
2  while (textFile.hasNextLine())
3  {
4     String line;
5  }
```

```
1  ...
2  while (textFile.hasNextLine())
3  {
4     String line;
5     if (CondExpr) break;
6  }
```

actic units at *any* location.

Introduction

- Code completion is an important feature in an IDE to improve developers' productivity.
- Recent research on code completion focuses mostly on next-token prediction, or aim to complete entire statements or blocks of code.
- In this work, we specifically focus on synthesizing *syntactic units* at ***any*** location.
- This is especially useful for other general program synthesis tasks such as *automated program repair*, *test generation in automated testing*, etc., which make use of one such code completion (CC) engine.

Why is this useful?

- Program analysis (PA)-based approaches cannot rank the candidates based on occurrence likelihoods.

Why is this useful?

- Program analysis (PA)-based approaches cannot rank the candidates based on occurrence likelihoods.
- Code statements are project-specific, thus rendering code mining and information retrieval (IR)-based approaches ineffective.

Why is this useful?

- Program analysis (PA)-based approaches cannot rank the candidates based on occurrence likelihoods.
- Code statements are project-specific, thus rendering code mining and information retrieval (IR)-based approaches ineffective.
- More recent large language model (LLM)-based approaches
  - can recommend syntactically incorrect or undefined code
  - can invoke functions/methods outside the scope of the codebase.

Our Approach: ASTCC

- is an AST-based statistical language model.

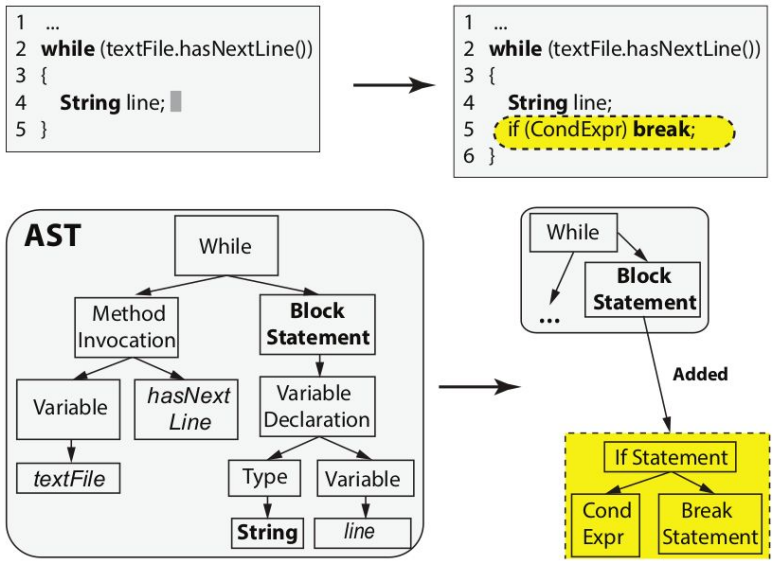
Our Approach: ASTCC

- is an AST-based statistical language model.
- leverages ASTLAN[1] to predict the next “expansion” of the current AST subtree.

[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi: 10.1109/ICSE.2015.336.

Our Approach: ASTCC

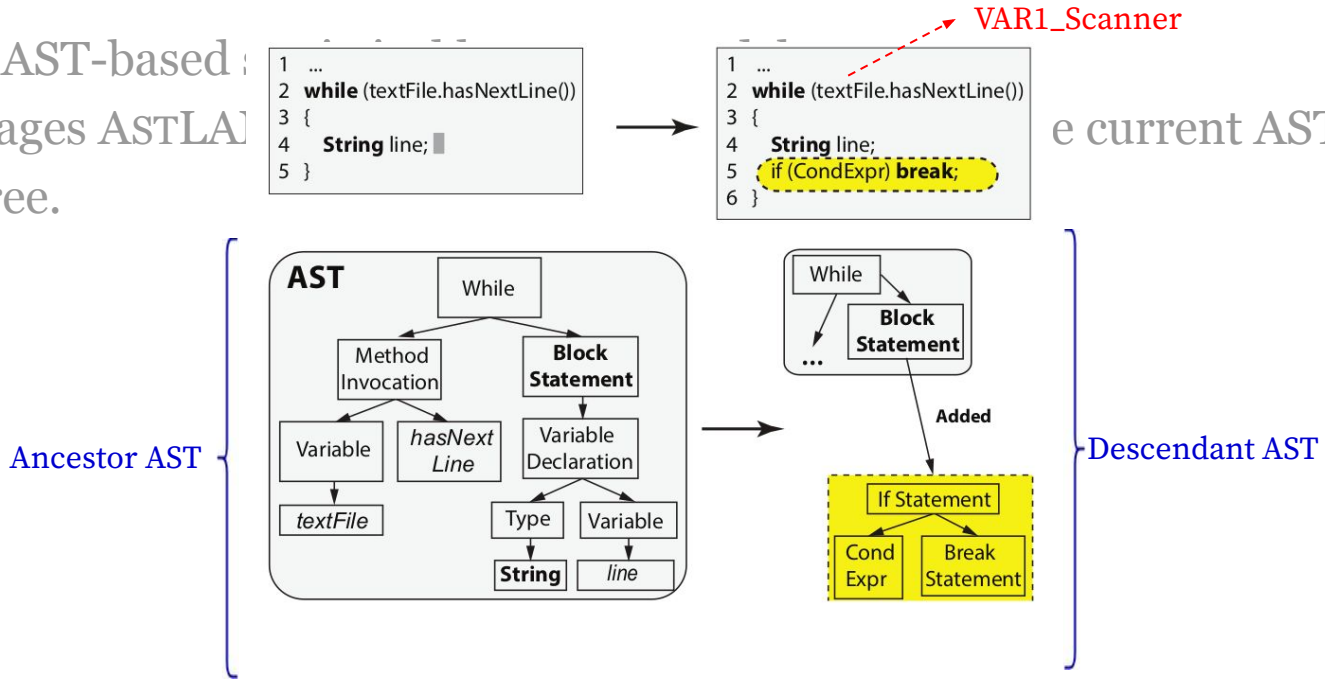
- is an AST-based
- leverages ASTLA e current AST subtree.



[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi: 10.1109/ICSE.2015.336.

Our Approach: ASTCC

- is an AST-based
- leverages ASTLA e current AST subtree.



[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi: 10.1109/ICSE.2015.336.

Our Approach: ASTCC

- is an AST-based statistical language model.
- leverages ASTLAN to predict the next “expansion” of the current AST subtree.
- integrates candidate syntactic verification into the process of learning the expansion from a smaller AST subtree to a larger one, thus ensuring the generated templates are syntactically valid

[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi: 10.1109/ICSE.2015.336.

Our Approach: ASTCC

- is an AST-based statistical language model
- leverages ASTLAN to predict the next “expansion” of the current AST subtree
- integrates candidate syntactic verification into the process of learning the expansion from a smaller AST subtree to a larger one, thus ensuring the generated templates are syntactically valid
- concretizes syntactic templates with variable names.

[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi: 10.1109/ICSE.2015.336.

Our Approach: ASTCC

Step 1. Training for ASTLAN.

Our Approach: AstCC

Step 1. Training for *A*

Syntax	Valid Expansion
If ::= **if** E S1 S2	If → E, S1 If → E, S1, S2
While ::= **while** E Stmt	While → E While → E, Stmt
For ::= **for** Init E Update Stmt	For → Init, E, Update For → Init, E, Update, Stmt
Switch ::= **switch** E Case* Def	Switch → E Switch → E, F with F ∈ all Case combinations Switch → E, Def Switch → E, F, Def with F ∈ all Case combs
Case ::= **case** E: Stmt	Case → E Case → E, Stmt
InfixOp ::= E1 Op E2	InfixOp → E1, E2
EnhancedFor ::= VarDec, Ref, Stmt	ForEach → VarDec, Ref ForEach → VarDec, Ref, Stmt
Do ::= Stmt, Cond	Do → Stmt, Cond
Try ::= **try** Block {Catches | Finally}	Try → Block, all combinations of Catches Try → Block, Finally Try → Block , all comb. of Catches, Finally
Conditional ::= E1 ? E2 : E3	Conditional → E1, E2, E3
Synchronized ::= Exp, Stmt	Synchronized → Exp, Stmt
Labeled ::= Lit, Stmt	Labeled → Lit, Stmt
Variable Dec. ::= TypeRef, VarSpec	VarDec → TypeRef, VarSpec
Variable Spec. ::= Name, Init	VarSpec → Name VarSpec → Name, Init
Type Reference ::= TypeName, TypeArg	TypeRef → TypeName TypeRef → TypeName, TypeArg
Other	All combinations

Our Approach: ASTCC

Step 1. Training for ASTLAN.

Step 2. Predicting/Generating the template of the next valid AST subtree.

Our Approach: ASTCC

Step 1. Training for ASTLAN.

Step 2. Predicting /Generating the template of the next valid AST subtree.

$$Pr(C(t)|Ctxt) = Pr((t, N^+, E^+)|t_1, .., t_n)$$
$$= \frac{\#methods(t_1, C(t))+\alpha}{\#method(C(t))+\alpha.\#methods} \cdots \frac{\#methods(t_{(i-1)}, C(t))+\alpha}{\#method(C(t))+\alpha.\#methods} \cdot \frac{\#methods(t, C(t))}{\#methods(t)} \cdot \frac{\#methods(t)}{\#methods} \cdots \frac{\#methods(t_n, C(t))+\alpha}{\#method(C(t))+\alpha.\#methods}$$

Our Approach: ASTCC

Step 1. Training for ASTLAN.

Step 2. Predicting/Generating the template of the next valid AST subtree.

Step 3. Variable names' concretization in the syntactic template.

Our Approach: AstCC

Step 1. Training for AstLAN.

Step 2. Predicting /Generating the template of the next valid AST subtree.

Step 3. Variable n mplate.

Algorithm 1 Concretizing Syntactic Template

```
1:  function Main(templ, V)
2:      candList = concretizeNext(templ, V, ∅, 1)
3:      return candList

4:  function Concretize(templ,V,curCandList,loc)
5:      if loc > size(templ) then return curCandList
6:      codeCands = ∅
7:      codeTokens = α(templ[loc], V)
8:      if curCandList = ∅ then
9:          for all t ∈ codeTokens do
10:             newCand = connect(EMPTY_TREE, t)
11:             codeCands.adds(newCand)
12:     else
13:         for all t ∈ codeTokens do
14:             for all cand ∈ curCandList do
15:                 newCand = connect(cand, t)
16:                 codeCands.adds(newCand)
17:     return Concretize(templ, V, codeCands, DFS.next(loc))
```

Preliminary Empirical Evaluation

We evaluate accuracy of ASTCC in:

1. suggesting next *syntactic-unit*
2. suggesting next *statement*

1. *Next Syntactic-Unit Code Completion*

- Data Collection
  -

Total projects	1,000
Total classes	104,645
Total methods	638,293
Total SLOCs	7,144,198
Total valid AST's fragments	1,047,614,720
Total distinctive fragments	36,608,102
Total distinctive AST nodes	302,367

1. *Next Syntactic-Unit Code Completion*

- Data Collection
  -

Total projects	1,000
Total classes	104,645
Total methods	638,293
Total SLOCs	7,144,198
Total valid AST's fragments	1,047,614,720
Total distinctive fragments	36,608,102
Total distinctive AST nodes	302,367

- Evaluation Metrics
  - *'top-k* Accuracy' is defined as the ratio between the number of hits over the total number of suggestions.

1. *Next Syntactic-Unit Code Completion*

- Data Collection
  -

Total projects	1,000
Total classes	104,645
Total methods	638,293
Total SLOCs	7,144,198
Total valid AST's fragments	1,047,614,720
Total distinctive fragments	36,608,102
Total distinctive AST nodes	302,367

- Evaluation Metrics
  - *'top-k* Accuracy' is defined as the ratio between the number of hits over the total number of suggestions.

- Experimental Results
  -

Top-1	Top-2	Top-3	Top-4	Top-5
33.2	**42.6**	**43.7**	**50.6**	**62.1**

2. *Next Statement Code Completion*

- Experimental Results
  -

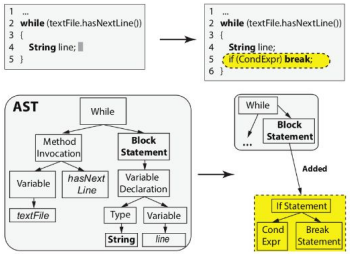
	Top 1	Top 3	Top 6	Top 10
AutoSC [15]	20.3	28.5	32.0	42.2
PCC [19]	28.9	51.1	54.8	59.3
AstCC	35.1	59.0	67.8	80.7

Future Applications and Plan

1. Real-world Code Completion Benchmark and Human Studies.
2. Syntactic Patterns Mining
3. Automated Program Repair
4. Using ASTCC in Automated Unit Test Generation

Summary

Our Approach: ASTCC

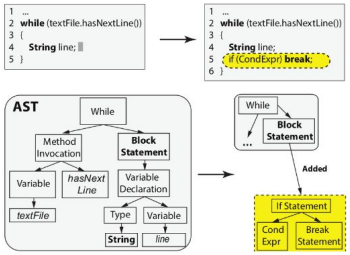


Our Approach: ASTCC

- is an AST-based statistical language model
- leverages ASTLAN to predict the next "expansion" of the current AST subtree
- integrates candidate syntactic verification into the process of learning the expansion from a smaller AST subtree to a larger one, thus ensuring the generated templates are syntactically valid
- concretizes syntactic templates with variable names.

[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi:10.1109/ICSE.2015.336.

Our Approach: ASTCC



Our Approach: ASTCC

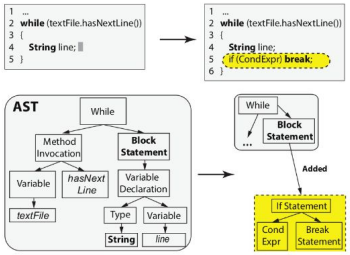
- is an AST-based statistical language model
- leverages ASTLAN to predict the next "expansion" of the current AST subtree
- integrates candidate syntactic verification into the process of learning the expansion from a smaller AST subtree to a larger one, thus ensuring the generated templates are syntactically valid
- concretizes syntactic templates with variable names.

[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi:10.1109/ICSE.2015.336.

Preliminary Empirical Evaluation

- AstCC can correctly suggest the next *syntactic unit* in 33% of the cases, and in 62% of the cases, it correctly suggests within five candidates
- AstCC can correctly suggest the next *statement* in 35% of the cases, and in 80% of the cases, it correctly suggests within ten candidates

Our Approach: ASTCC



Our Approach: ASTCC

- is an AST-based statistical language model
- leverages ASTLAN to predict the next "expansion" of the current AST subtree
- integrates candidate syntactic verification into the process of learning the expansion from a smaller AST subtree to a larger one, thus ensuring the generated templates are syntactically valid
- concretizes syntactic templates with variable names.

[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi:10.1109/ICSE.2015.336.

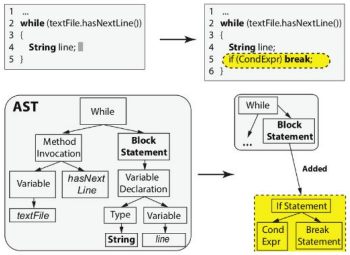
Preliminary Empirical Evaluation

- AstCC can correctly suggest the next *syntactic unit* in 33% of the cases, and in 62% of the cases, it correctly suggests within five candidates
- AstCC can correctly suggest the next *statement* in 35% of the cases, and in 80% of the cases, it correctly suggests within ten candidates

Future Applications and Plan

1. Real-world Code Completion Benchmark and Human Studies.
2. Syntactic Patterns Mining
3. Automated Program Repair
4. Using ASTCC in Automated Unit Test Generation

Our Approach: ASTCC



Our Approach: ASTCC

- is an AST-based statistical language model
- leverages ASTLAN to predict the next "expansion" of the current AST subtree
- integrates candidate syntactic verification into the process of learning the expansion from a smaller AST subtree to a larger one, thus ensuring the generated templates are syntactically valid
- concretizes syntactic templates with variable names.

[1] A. T. Nguyen and T. N. Nguyen, "Graph-Based Statistical Language Model for Code," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 858-868, doi:10.1109/ICSE.2015.336.

Preliminary Empirical Evaluation

- AstCC can correctly suggest the next *syntactic unit* in 33% of the cases, and in 62% of the cases, it correctly suggests within five candidates
- AstCC can correctly suggest the next *statement* in 35% of the cases, and in 80% of the cases, it correctly suggests within ten candidates

Future Applications and Plan

1. Real-world Code Completion Benchmark and Human Studies.
2. Syntactic Patterns Mining
3. Automated Program Repair
4. Using ASTCC in Automated Unit Test Generation

...feel free to reach out to us if you've any questions :)

Anh Tuan Nguyen
ntanhbk44@gmail.com

Aashish Yadavally
aashish.yadavally@utdallas.edu

Tien N. Nguyen
tien.n.nguyen@utdallas.edu