# A Grammar-Based Structural CNN Decoder for Code Generation

Zeyu Sun, Qihao Zhu, Lili Mou, **Yingfei Xiong**, Ge Li, Lu Zhang

Peking University
AdeptMind

# INTRODUCTION

- Generating code from natural language description.
  - Open the file, F1 → f = open('F1', 'r')

- Automatically code generation is beneficial in various scenarios.
  - Similar code snippets can be generated from another.
  - It takes a long time for a programmer to learn a new implement.

# INTRODUCTION

- Previous works with neural network are all based on RNN or LSTM.
  - Researchers [1, 2, 3] have proposed several approach based on AST using LSTM.

- A program is much larger than a natural language sentence and that RNNs suffer from the long dependency problem [4].
  - A program is made up of a large number of AST nodes.


1. Dong, L., and Lapata, M. 2016. Language to logical form with neural attention. In ACL, 33–43.
2. Yin, P., and Neubig, G. 2017. A syntactic neural model for general-purpose code generation. In ACL, 440–450.
3. Rabinovich, M.; Stern, M.; and Klein, D. 2017. Abstract syntax networks for code generation and semantic parsing. In ACL, 1139–1149.
4. Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5(2):157–166.

# INTRODUCTION

- Researchers are showing growing interest in using the CNN as the decoder.
  - QANet [1], a CNN encoder-decoder, achieves a significant improvement in SQuAD dataset for question answering.

- Can we use CNN for code generation?

1. Yu, A. W.; Dohan, D.; Luong, M.-T.; Zhao, R.; Chen, K.; Norouzi, M.; and Le, Q. V. 2018. QANet: Combining local convolution with global self-attention for reading comprehension. In ICLR.

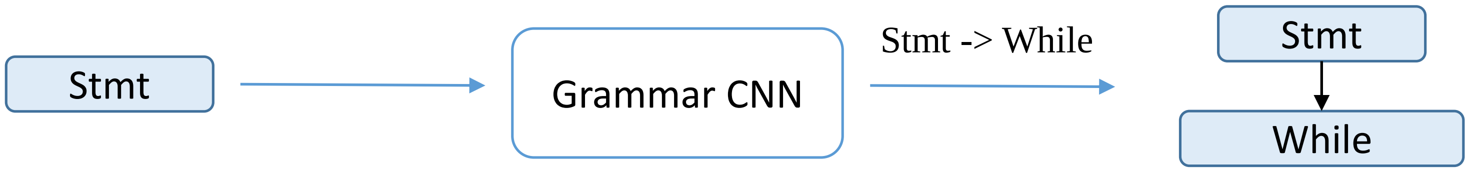
# Background: Learning to Synthesize

- CNN produces a classifier of a fixed number of categories
  - Cannot be applied to program generation: the category is infinite or extremely large

- Yingfei Xiong, Bo Wang, Guirong Fu, Linfei Zang. Learning to Synthesize. GI'18.
  - Decompose a program generation problem into a series of classification problems
  - Guided by the grammar of the program
  - Presented at APLAS-NIER 2017

# Decompose into classification problems

- The key step of generation is to predict the grammar rule, which will be applied to expand the AST.



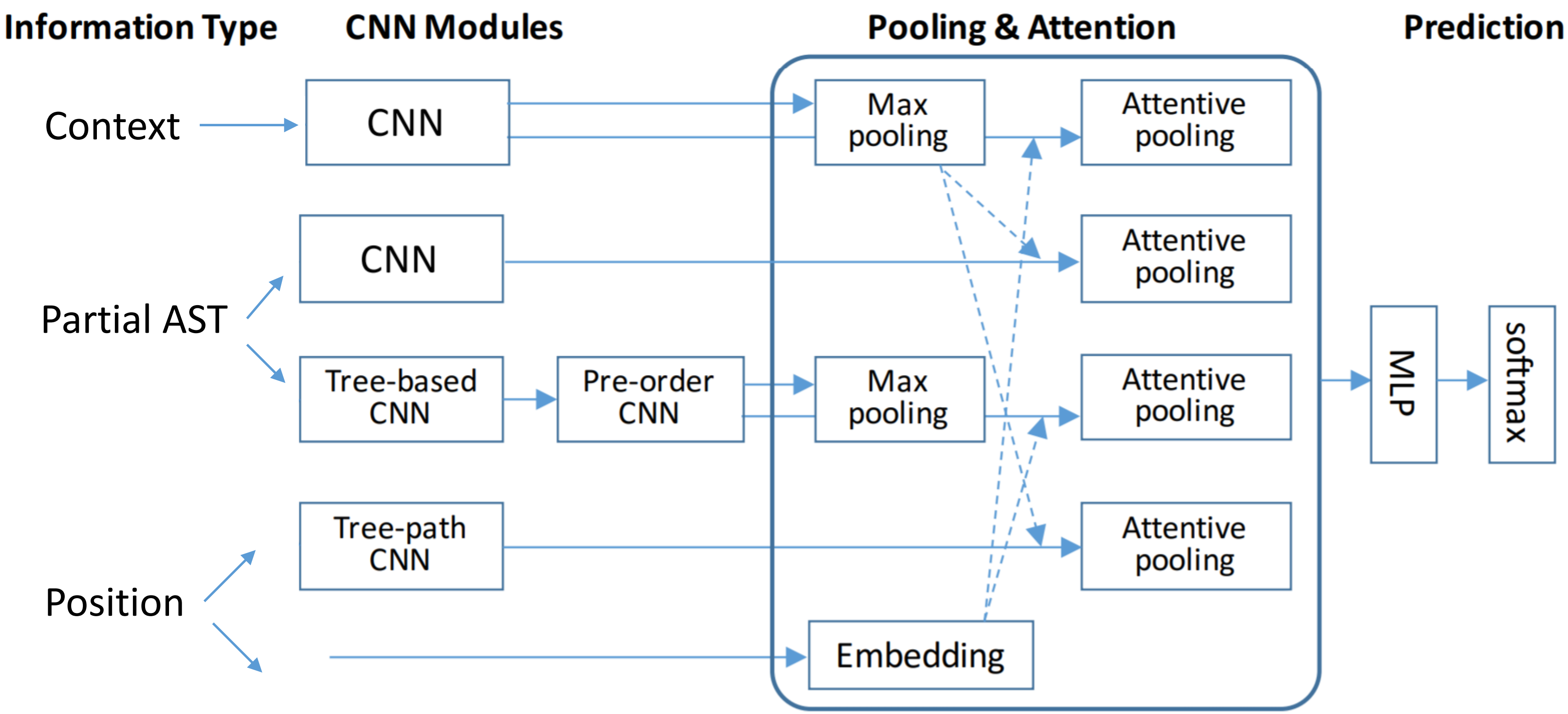

- The probability of an entire code is decomposed as

$$p(\text{program}) = \prod_{n=1}^{N} p(\mathrm{r}_n | \mathrm{r}_1 \cdots, \mathrm{r}_{n-1})$$

# TO PREDICT GRAMMAR RULES

- The prediction is mainly based on three types of information:
  - the context information (e.g. the natural language description)
  - the partial AST that has been generated.
  - the position of the node to be expanded

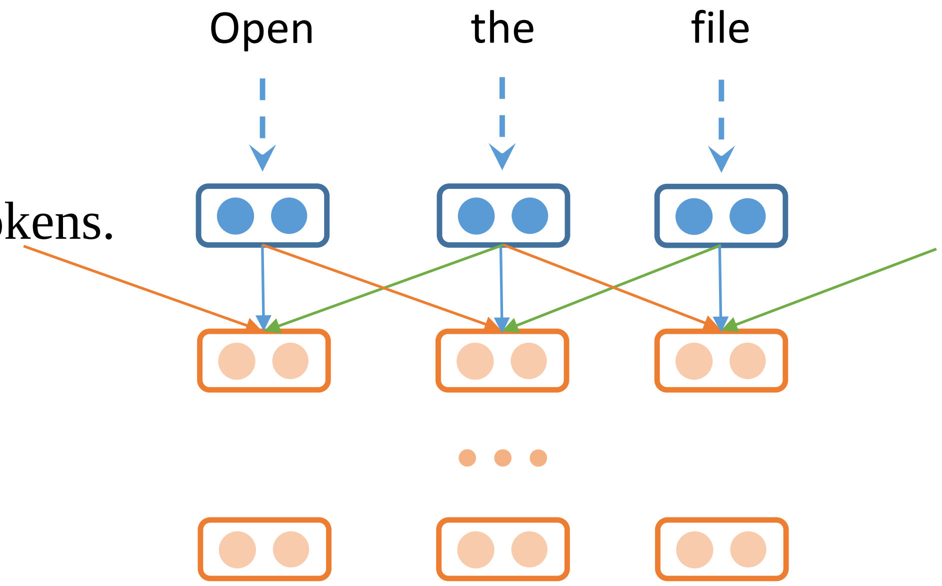
# OVERVIEW



Information Type
CNN Modules
Pooling & Attention
Prediction
Context
CNN
Max pooling
Attentive pooling
CNN
Attentive pooling
Partial AST
Tree-based CNN
Pre-order CNN
Max pooling
Attentive pooling
MLP
softmax
Tree-path CNN
Attentive pooling
Position
Embedding

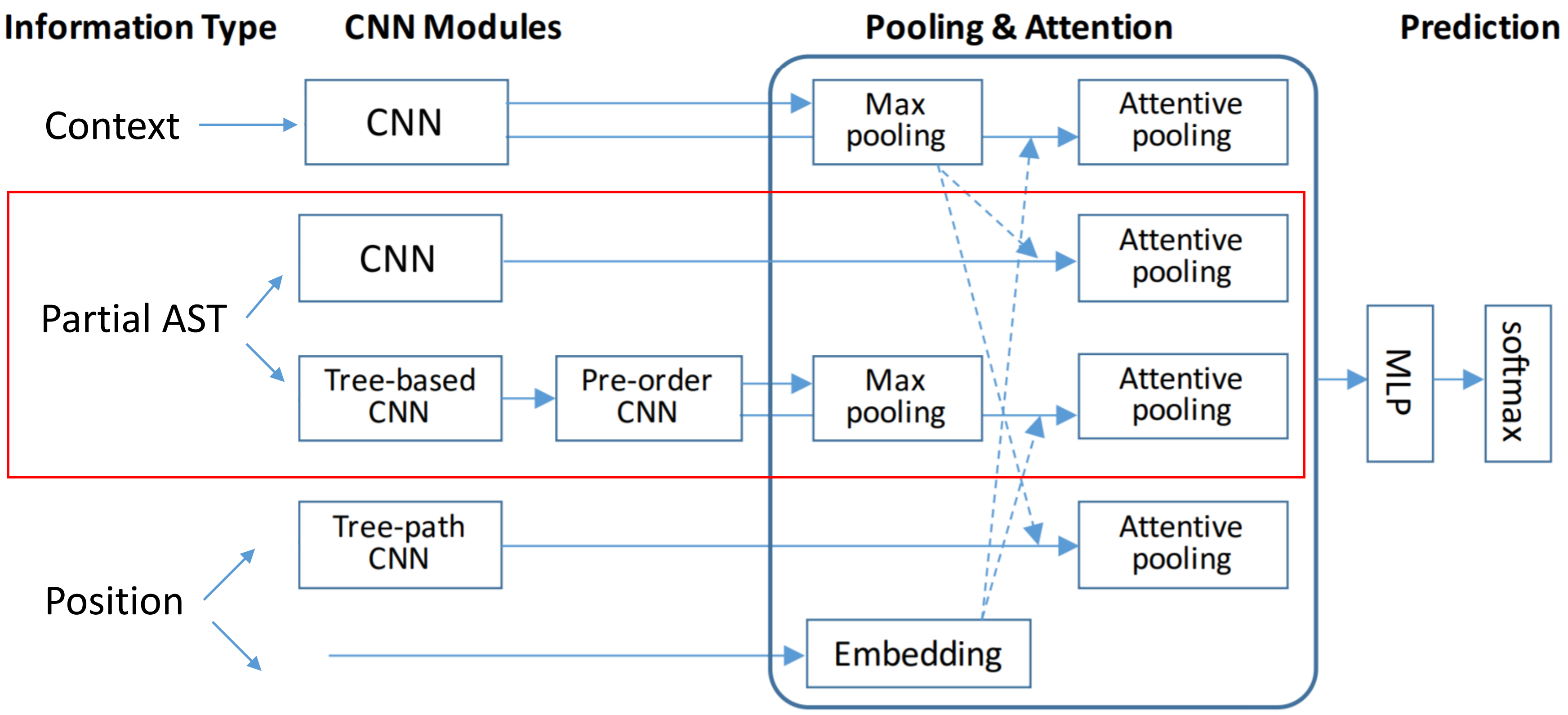
# TO ENCODE THE CONTEXT

- The context of our model is a piece of description.
- We first tokenize the context, and obtain a sequence of tokens.
- Then, a set of convolutional layers are applied.
  - We adopt shortcut connections every other layer parallel to linear transformation, as in ResNet [1].




[1] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In CVPR, 770–778.

# OVERVIEW



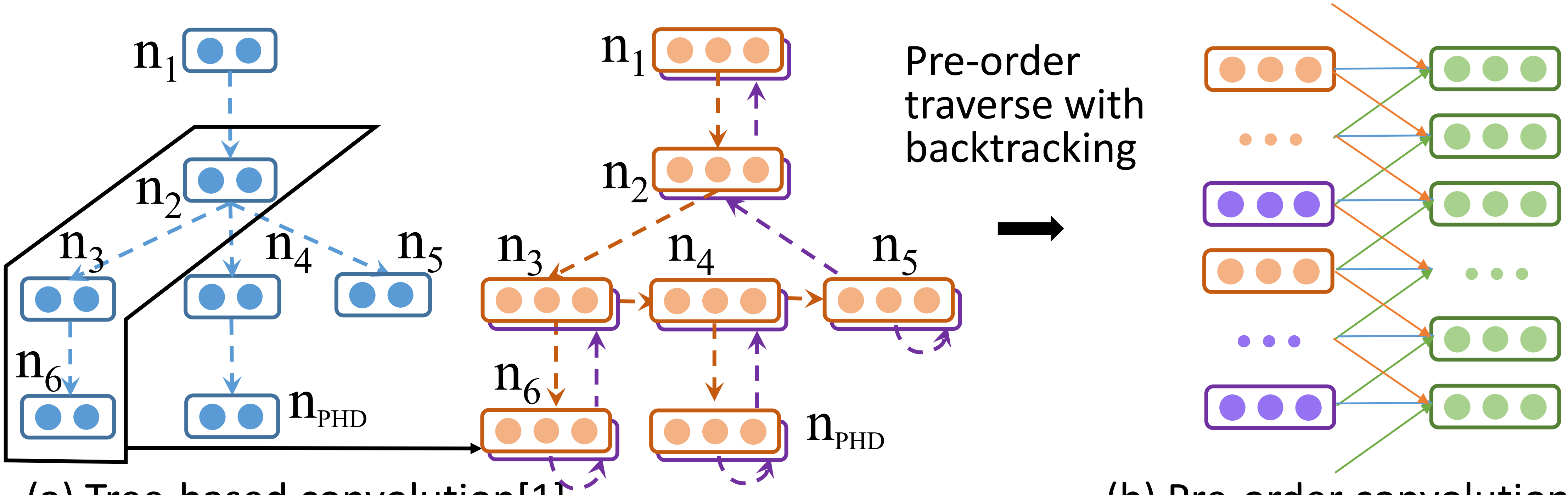
Information Type
CNN Modules
Pooling & Attention
Prediction
Context
CNN
Max pooling
Attentive pooling
Partial AST
CNN
Attentive pooling
Tree-based CNN
Pre-order CNN
Max pooling
Attentive pooling
Position
Tree-path CNN
Attentive pooling
Embedding
MLP
softmax

# TO ENCODE THE PARTIAL AST

- The partial AST is decided by the rules predicted
  - A sequence of rules can be directly encoded by a CNN
- However, it is difficult for the CNN to learn the structure of the AST
  - A tree-based CNN is further applied to capture the structure information

# TO CAPTURE THE STRUCTURAL INFORMATION

- We split each node into two nodes.



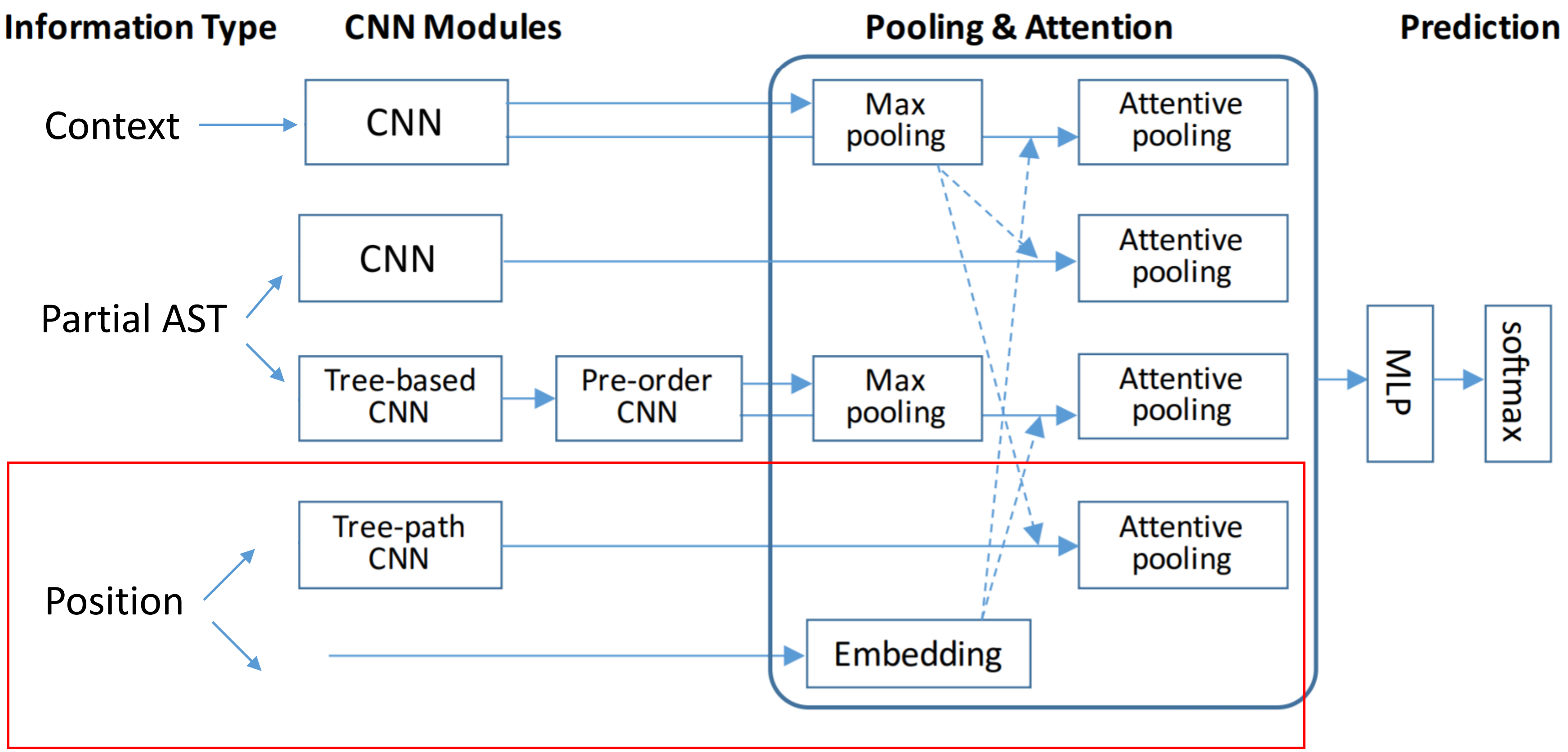

(a) Tree-based convolution[1]

(b) Pre-order convolution

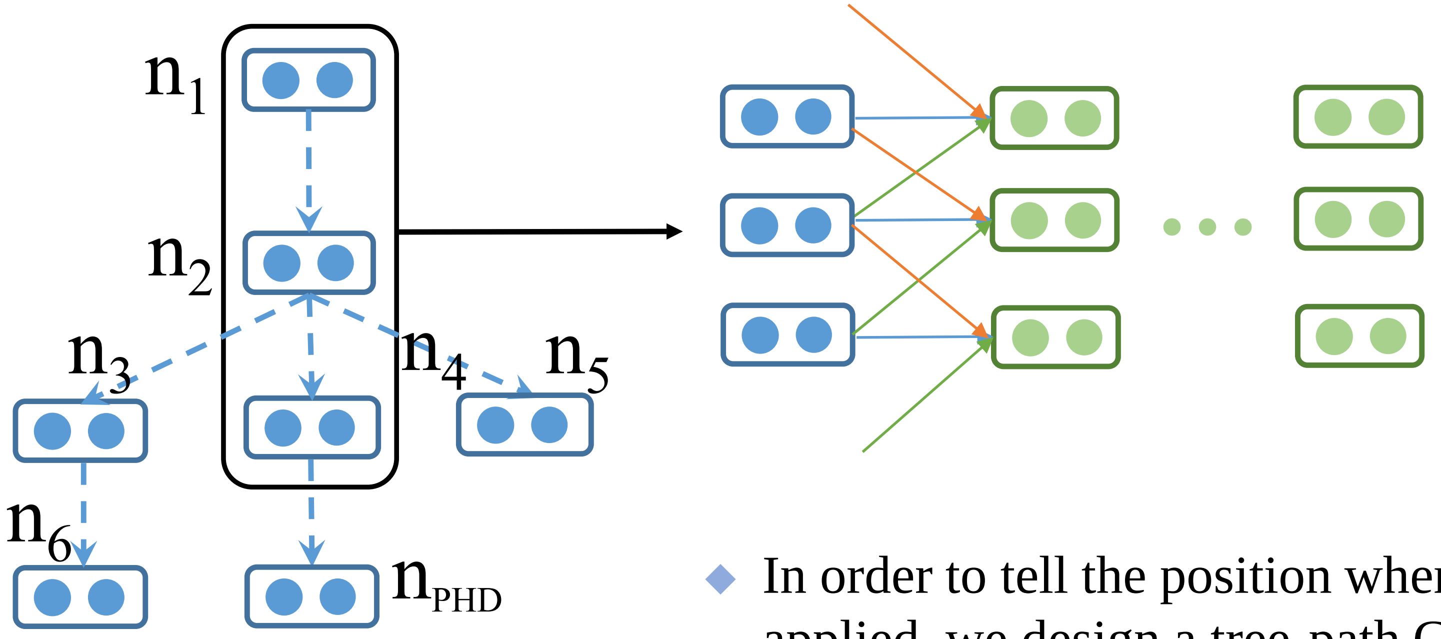
- A local feature detector of a fixed depth, sliding over a tree to extract structural feature.
- We put a placeholder to indicate where the next grammar rule is applied.

[1] Mou, L.; Li, G.; Zhang, L.; Wang, T.; and Jin, Z. 2016. Convolutional neural networks over tree structures for programming language processing. In AAAI, 1287–1293

# OVERVIEW


Information Type
CNN Modules
Pooling & Attention
Prediction
Context
CNN
Max pooling
Attentive pooling
CNN
Attentive pooling
Partial AST
Tree-based CNN
Pre-order CNN
Max pooling
Attentive pooling
MLP
softmax
Tree-path CNN
Attentive pooling
Position
Embedding

# TO ENCODE THE POSITION(1/2)




- In order to tell the position where the next grammar rule is applied, we design a tree-path CNN to catch this information.
- We extract the path from the root to the node to expand.

# TO ENCODE THE POSITION (2/2)

- The scope name is often important for code prediction
  - Class name
  - Method name
- The current local scope name is also encoded

# TO DECODE

- Moreover, we also design several components for code generation.
  - The CNN for predicted.
  - The attentive pooling.

# EXPERIMENT: HEARTHSTONE

- Our main experiment is based on an established benchmark dataset, HearthStone (HS) [1]
- The dataset comprises 665 different cards of the HearthStone game.
- We use StrAcc (exact match), Acc+ and BLEU-4 score as metrics.
    - Acc+ is a human-adjusted accuracy.


```
[NAME]
Acidic Swamp Ooze
[ATK] 3
[DEF] 2
[COST]  2
[DUR]  -1
[TYPE]  Minion
[CLASS] Neutral
[RACE] NIL
[RARITY] Common
[DESCRIPTION]
"Battlecry: Destroy Your Opponent's Weapon"
```

```python
class AcidicSwampOoze(MinionCard):
    def __init__(self):
        super().__init__("Acidic Swamp Ooze", 2,
            CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
            battlecry=Battlecry(Destroy(), WeaponSelector(EnemyPlayer())))

    def create_minion(self, player):
        return Minion(3, 2)
```

[1] Ling, W.; Blunsom, P.; Grefenstette, E.; Hermann, K. M.; Kocisk ˇ y, T.; Wang, F.; and Senior, A. 2016. Latent predictor ` networks for code generation. In ACL, 599–609.

# EXPERIMENT: HEARTHSTONE

- Our model is compared with previous state-of-the-art results.

| Model | StrAcc | Acc+ | BLEU |
|---|---|---|---|
| LPN (Ling et al. 2016) | 6.1 | – | 67.1 |
| SEQ2TREE (Dong and Lapata 2016) | 1.5 | – | 53.4 |
| SNM (Yin and Neubig 2017) | 16.2 | ∼18.2 | 75.8 |
| ASN (Rabinovich, Stern, and Klein 2017) | 18.2 | – | 77.6 |
| ASN+SUPATT (Rabinovich, Stern, and Klein 2017) | 22.7 | – | 79.2 |
| Our system | **27.3** | **30.3** | **79.6** |

- Ablation tests to analyze the contribution of each component.

| Line # | Model Variant | Acc+ | BLEU |
|---|---|---|---|
| 1 | Full model | **30.3** | 79.6 |
| 2 | Pre-order CNN → LSTM | 21.2 | 78.8 |
| 3 | – Predicted rule CNN | 24.2 | 79.2 |
| 4 | – Pre-order CNN | 25.8 | **80.4** |
| 5 | – Tree-based CNN | 25.8 | 79.4 |
| 6 | – Tree-path CNN | 28.8 | **80.4** |
| 7 | – Attentive pooling | 24.2 | 79.3 |
| 8 | – Scope name | 25.8 | 78.6 |

- Our model outperforms all previous results.
- We have designed reasonable components of the neural architecture, suited to the code generation task.

# EXPERIMENT: HEARTHSTONE

```
Generated code:
class Maexxna(MinionCard):
    def __init__(self):
        super().__init__("Maexxna", 6, CHARACTER_CLASS.ALL,
            CARD_RARITY.LEGENDARY, minion_type = MINION_TYPE.BEAST)

    def create_minion(self, player):
        return Minion(2, 8, effects = [Effect(DidDamage(),
            ActionTag(Kill(), TargetSelector(IsMinion())))])

Reference code:
class Maexxna(MinionCard):
    def __init__(self):
        super().__init__("Maexxna", 6, CHARACTER_CLASS.ALL,
            CARD_RARITY.LEGENDARY, minion_type = MINION_TYPE.BEAST)

    def create_minion(self, player):
        return Minion(2, 8, effects = [Effect(DidDamage(),
            ActionTag(Kill(), TargetSelector(IsMinion())))])
```

- The code we successfully generated.

```
Generated Code:
class Gnoll(MinionCard):
    def __init__(self):
        super().__init__("Gnoll", 2, CHARACTER_CLASS.ALL,
                CARD_RARITY.COMMON, False)

    def create_minion(self, p):
        return Minion(2, 2, taunt = True)

Reference Code:
class Gnoll(MinionCard):
    def __init__(self):
        super().__init__("Gnoll", 2, CHARACTER_CLASS.ALL,
                CARD_RARITY.COMMON, False)

    def create_minion(self, player):
        return Minion(2, 2, taunt = True)

Reference Code For Anthor Card:
class DefenderMinion(MinionCard):
    def __init__(self):
        super().__init__("Defender", 1, CHARACTER_CLASS.PALADIN,
                CARD_RARITY.COMMON)

    def create_minion(self, p):
        return Minion(2, 1)
```

- Our model used a different argument name, but implements a correct functionality.

# EXPERIMENT: SEMANTIC PARSING

- Semantic parsing aims to generate logical forms given a natural language description.

**Input description:** list airport in ci0
**Output $\lambda$-calculus:**

```
lambda $0 e ( and ( airport $0 )
                  ( loc:t $0 ci0 ) )
```

- We evaluated our model on two semantic parsing datasets (ATIS and JOBS) used in Dong and Lapata (2016) [1] with Accuracy.

[1] Dong, L., and Lapata, M. 2016. Language to logical form with neural attention. In ACL, 33–43.

# EXPERIMENT: SEMANTIC PARSING

- The logic form for semantic parsing is usually short, containing only 1/4–1/3 tokens as in HS.

| | ATIS | | JOBS | |
|---|---|---|---|---|
| | **System** | **Accuracy** | **System** | **Accuracy** |
| Traditional | ZH15 | 84.2 | ZH15 | 85.0 |
| | ZC07 | 84.6 | PEK03 | 88.0 |
| | WKZ14 | **91.3** | LJK13 | 90.7 |
| Neural | SEQ2TREE | 84.6 | SEQ2TREE | 90.0 |
| | ASN | 85.3 | ASN | 91.4 |
| | ASN-SUPATT | 85.9 | ASN-SUPATT | **92.9** |
| | Our System | 85.0 | Our System | 89.3 |

- Neural models are generally worse than the WKZ14 system (based on CCG parser).
- Our model achieves results similar to the state-of-the-art neural models.

# CONCLUSION

- We propose a grammar-based structural CNN for code generation.
- Our model makes use of the abstract syntax tree (AST) of a program, and generates code by predicting the grammar rules.
- We address the problem that traditional RNN-based approaches may not be suitable to program generation.

北京大学软件工程研究所
Software Engineering Institute, Peking University

adeptmind

# Thank you!

A Grammar-Based Structural CNN Decoder
for Code Generation