Microsoft

# From Language Modeling to Grammar-Guided Code Sketch Generation

Alexey Svyatkovskiy

Microsoft Research, Data & AI

# THE ML4Code LANDSCAPE

**Code Generation**

- Code Completion
- Program Synthesis
- Semantic Parsing

**Program Analysis**

- Specification Tuning
- Specification Inference
- Black-Box Analysis Learning

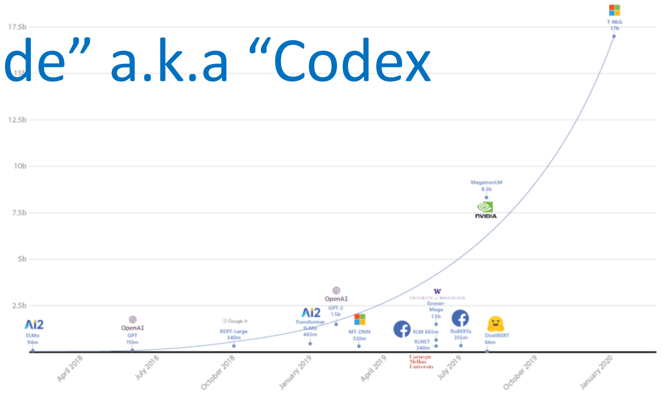
…

# Introduction: Language Models of Code (I)

- Hindle et. al. “On the Naturalness of Software” (2012)
  - Consider programs as sequences of tokens
  - Use N-gram model to estimate how likely tokens are to follow each other
  - Assume Markov property
- Svyatkovskiy et. al. “IntelliCode Compose” (2020)
  - Programs as sequence of BPE subtokens
  - Pretrain a (relatively small capacity) transformer decoder-only model
  - Decode lines of code via beam search, use prefix trie-cache to bridge the perf gap
- Feng et. al, “CodeBERT: A Pretrained Model for Programming and NL” (2020)
  - Pretrain a transformer encoder via MLM+RTD
  - Code understanding tasks, NL-PL

# Introduction: Language Models of Code (II)

- Chen et. al., "Evaluating Large Language Models Trained on Code" a.k.a "Codex paper" (2021)
  - GPT-style model
  - Larger capacity model (up to 12 billion parameters)
  - Longer sequences




- Clement et. al, "Long-range Modeling of Source Code Files with eWASH" (2021)
  - Solve problem of long-range modeling of source code
- Guo et. al., "Learning to Complete Code with Sketches" the "Grammformers" paper (2021)
  - Grammar-guided code generation
  - Avoid hallucinations by predicting sketches with "holes" – non-terminals
- Future?
  - ChatGPT: LLM + RLHF finetuning

# Limitations of Modern LMs of Code

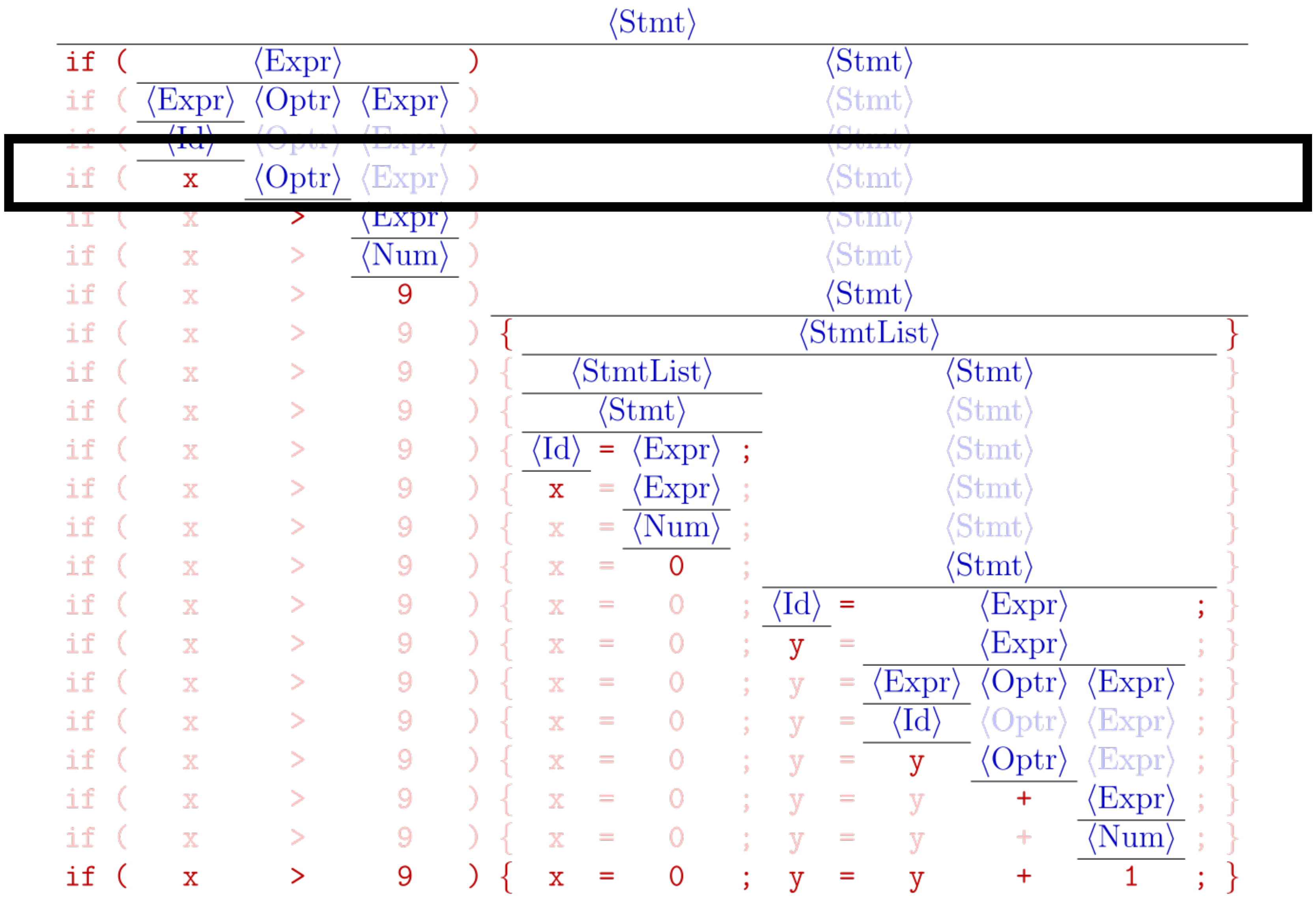
- ***Lack of Interpretability:*** ML models may have issues with interpretability, making it difficult to reason about why a model is making certain predictions
- While LMs generate realistic-looking outputs, they are known to occasionally ***"hallucinate"*** generating plausible but incorrect content
- ***Uncertainty Quantification:*** *specifically, a lack of ability to decline to make predictions when uncertain, outputting a hole, but continuing to generate around holes*
- ***Syntactic correctness:*** *LMs of code are commonly trained on partial code snippets which are not syntactically correct, as such it may generate syntactically incorrectly prediction*

# Programming Language Grammars (I)

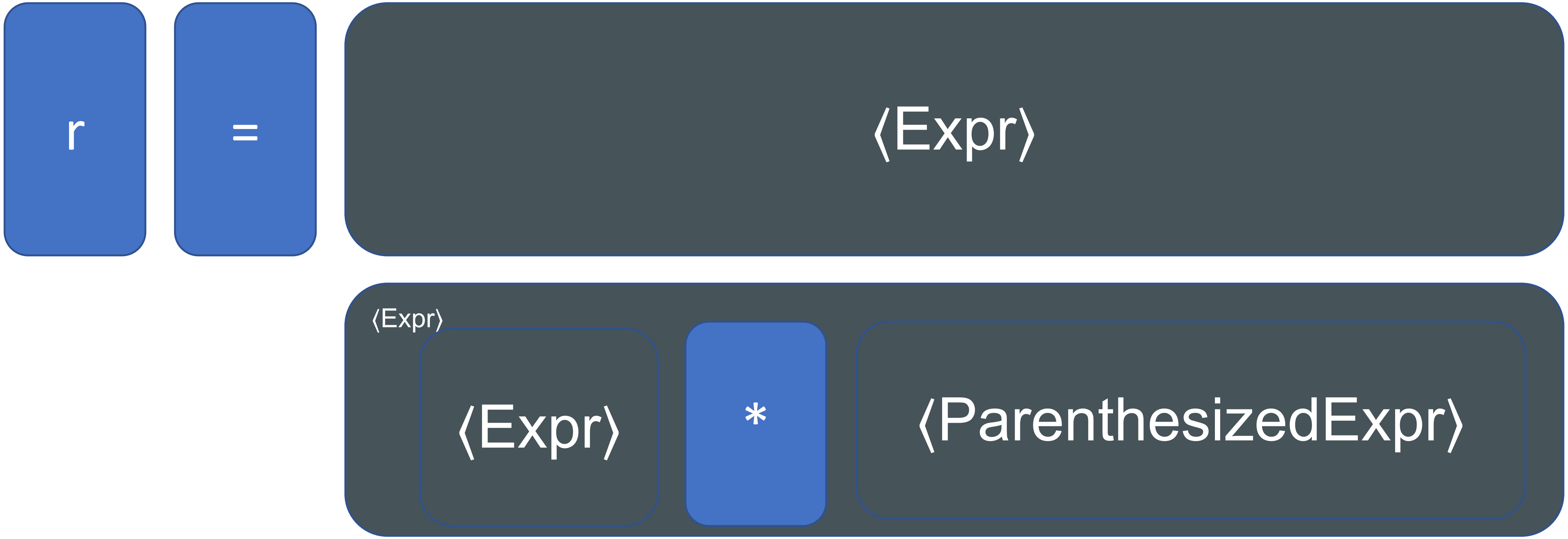
- A grammar of a programming language formally specifies the syntax rules of that language. Grammars are commonly used in compilers, which translate code written in a particular language into executable form. They are also used in code editors and IDEs to enforce syntax rules and assist with code completion.
- A grammar typically consists of tuple (Σ, N, S, R):
  - Σ: a set of terminal symbols, which represent the basic building blocks of the language (e.g. keywords, variable names, operators)
  - N: a set of nonterminal symbols, which are placeholders for sequences of terminal symbols. We denote non-terminals as ⟨NonTerminalName⟩
  - R: a set of productions, which specify the ways that nonterminal symbols can be replaced by sequences of terminal and/or nonterminal symbols.
  - S: a start symbol, which specifies the initial symbol in the grammar from which all derivations begin

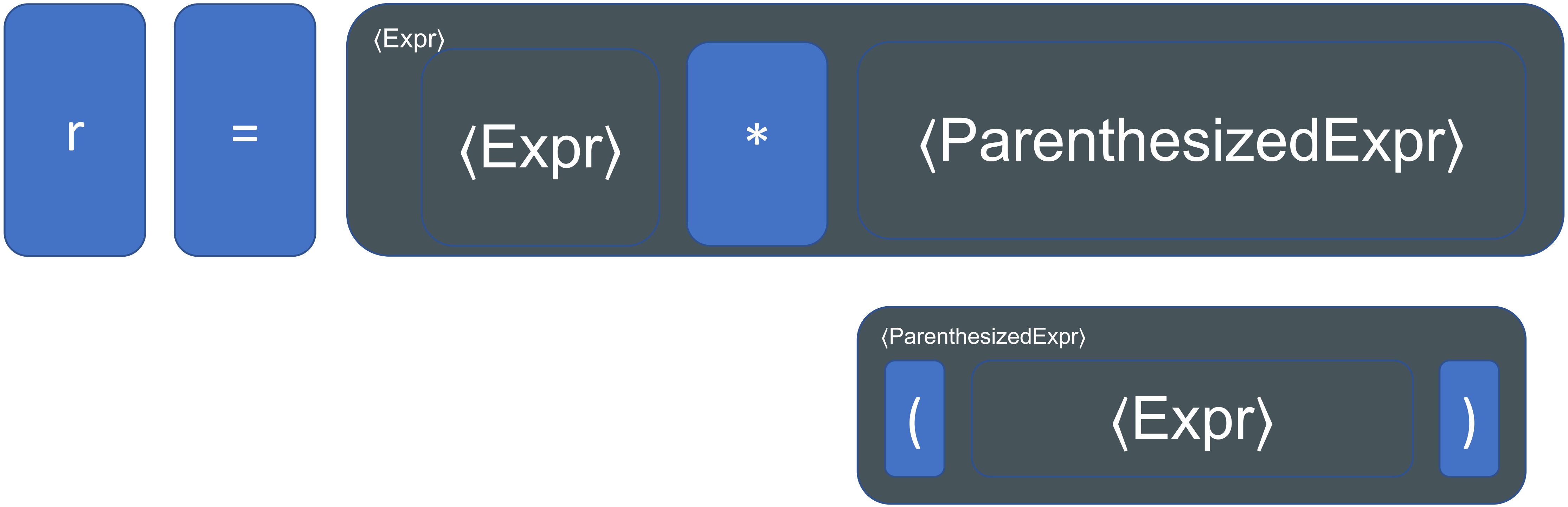
# Programming Language Grammars (II)

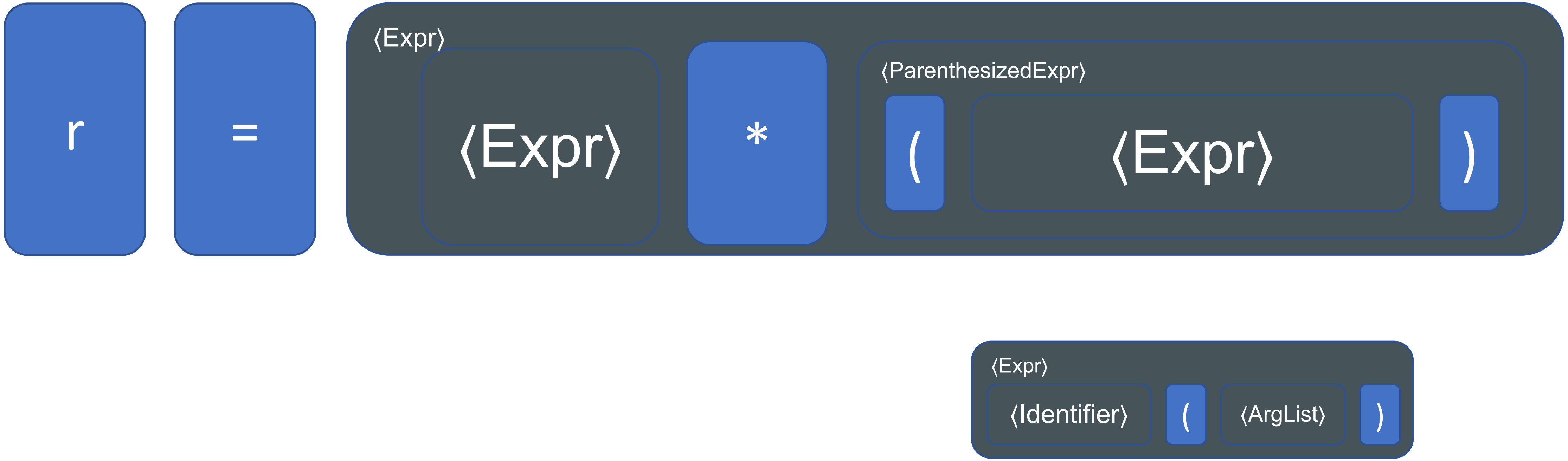
$\langle \text{Stmt} \rangle \rightarrow \langle \text{Id} \rangle$ `=` $\langle \text{Expr} \rangle$ `;`
$\langle \text{Stmt} \rangle \rightarrow$ `{` $\langle \text{StmtList} \rangle$ `}`
$\langle \text{Stmt} \rangle \rightarrow$ `if (` $\langle \text{Expr} \rangle$ `)` $\langle \text{Stmt} \rangle$
$\langle \text{StmtList} \rangle \rightarrow \langle \text{Stmt} \rangle$
$\langle \text{StmtList} \rangle \rightarrow \langle \text{StmtList} \rangle \langle \text{Stmt} \rangle$
$\langle \text{Expr} \rangle \rightarrow \langle \text{Id} \rangle$
$\langle \text{Expr} \rangle \rightarrow \langle \text{Num} \rangle$
$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle$
$\langle \text{Id} \rangle \rightarrow$ `x`
$\langle \text{Id} \rangle \rightarrow$ `y`
$\langle \text{Num} \rangle \rightarrow$ `0`
$\langle \text{Num} \rangle \rightarrow$ `1`
$\langle \text{Num} \rangle \rightarrow$ `9`
$\langle \text{Optr} \rangle \rightarrow$ `>`
$\langle \text{Optr} \rangle \rightarrow$ `+`




https://en.wikipedia.org/wiki/Context-free_grammar


r
=
⟨Expr⟩
⟨Expr⟩
⟨Expr⟩
*
⟨ParenthesizedExpr⟩

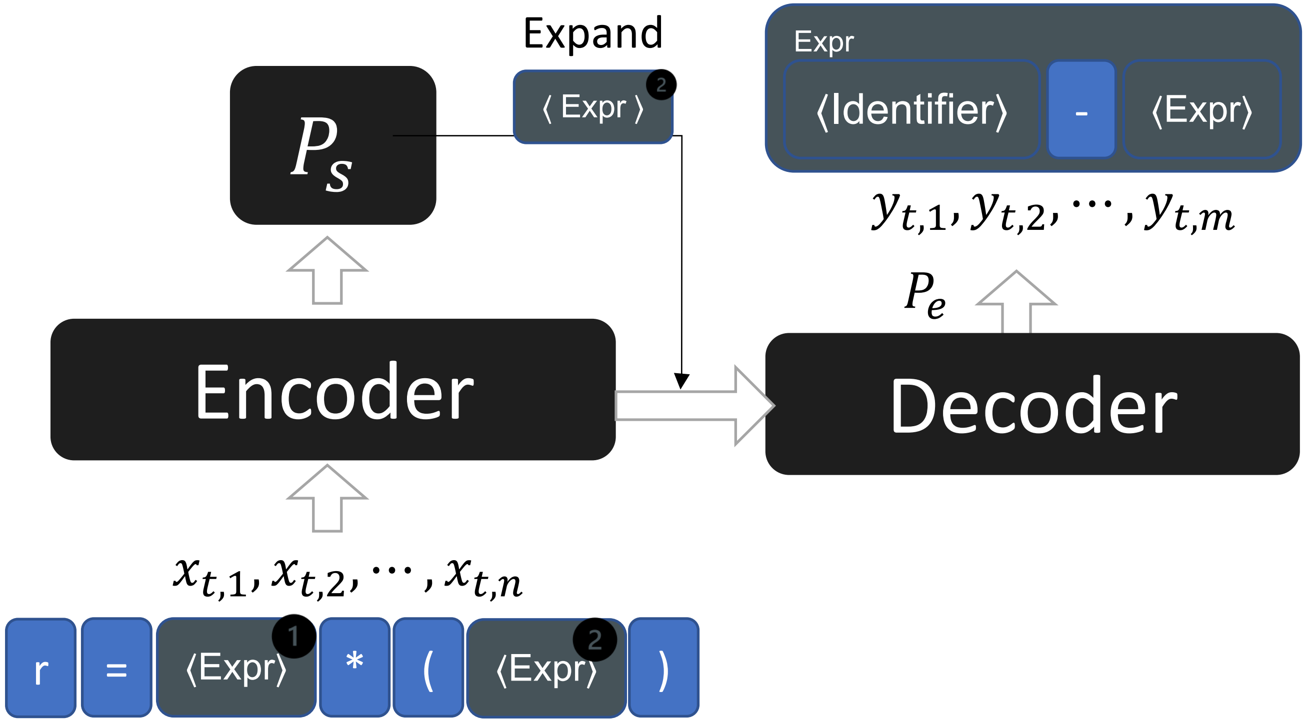

r
=
⟨Expr⟩
⟨Expr⟩
*
⟨ParenthesizedExpr⟩
⟨ParenthesizedExpr⟩
(
⟨Expr⟩
)


r
=
⟨Expr⟩
⟨Expr⟩
*
⟨ParenthesizedExpr⟩
(
⟨Expr⟩
)
⟨Expr⟩
⟨Identifier⟩
(
⟨ArgList⟩
)


r
=
⟨Expr⟩
*
⟨ParenthesizedExpr⟩
(
⟨Expr⟩
foo
(
)
)


```
r = ▧ * (foo(▧))
```

# Grammformers – neural model




**for** $t = 0, 1, 2, \ldots$ **do**

$i_t \sim P_s(i | \mathbf{x}_t, N(\mathbf{x}_t))$

**if** $i_t = \oslash$ **then**

**break**

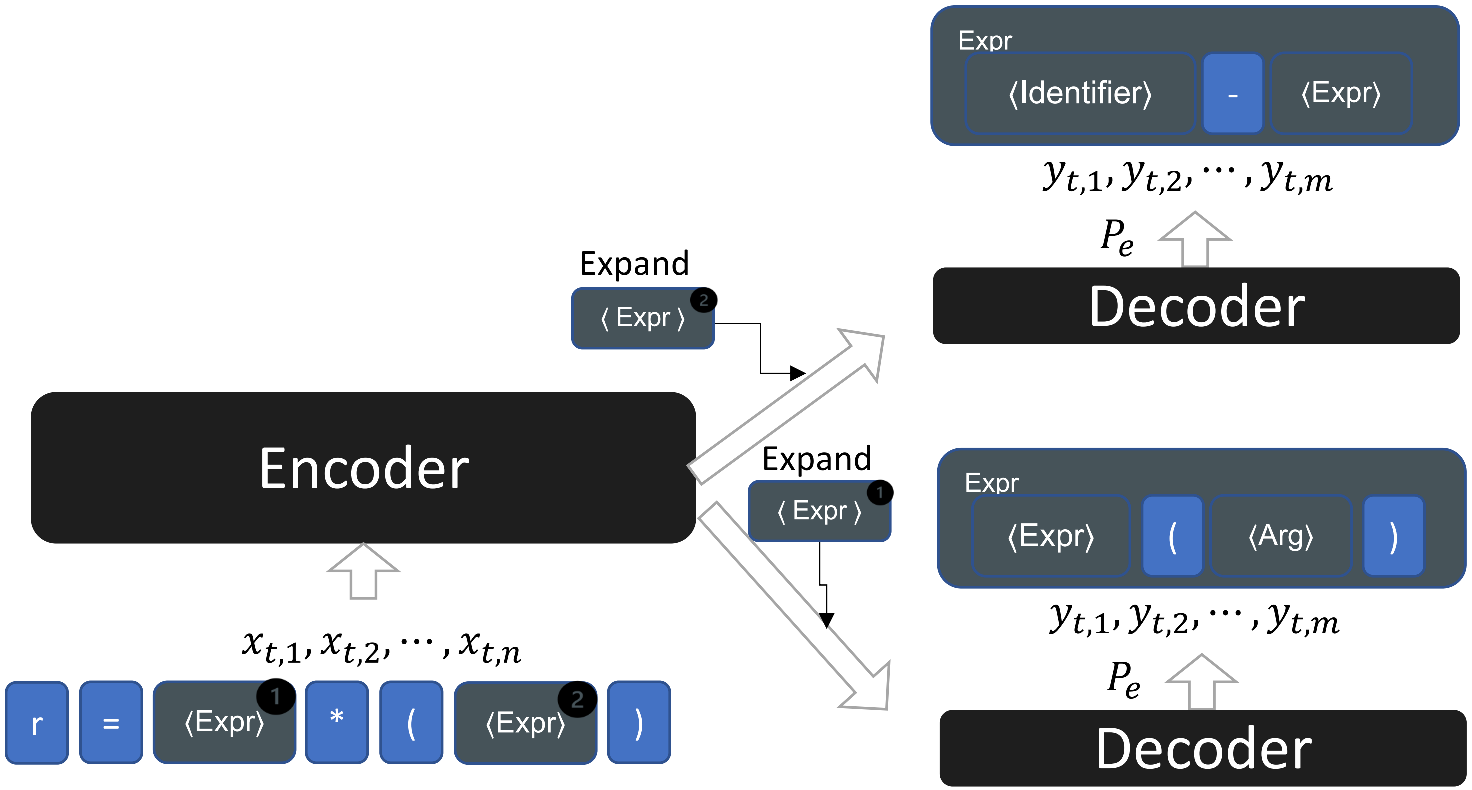
$\hat{\mathbf{y}}_{t \circledcirc i_t} \sim P_e(\mathbf{y} | \mathbf{x}_t, i_t)$

$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_{t,<i_t} :: \hat{\mathbf{y}}_{t \circledcirc i_t} :: \mathbf{x}_{t,>i_t}$

$\mathbf{x}_{\text{out}} \leftarrow \text{NonTerminalsToHoles}(\mathbf{x}_t)$

**return** $\mathbf{x}_{\text{out}}$

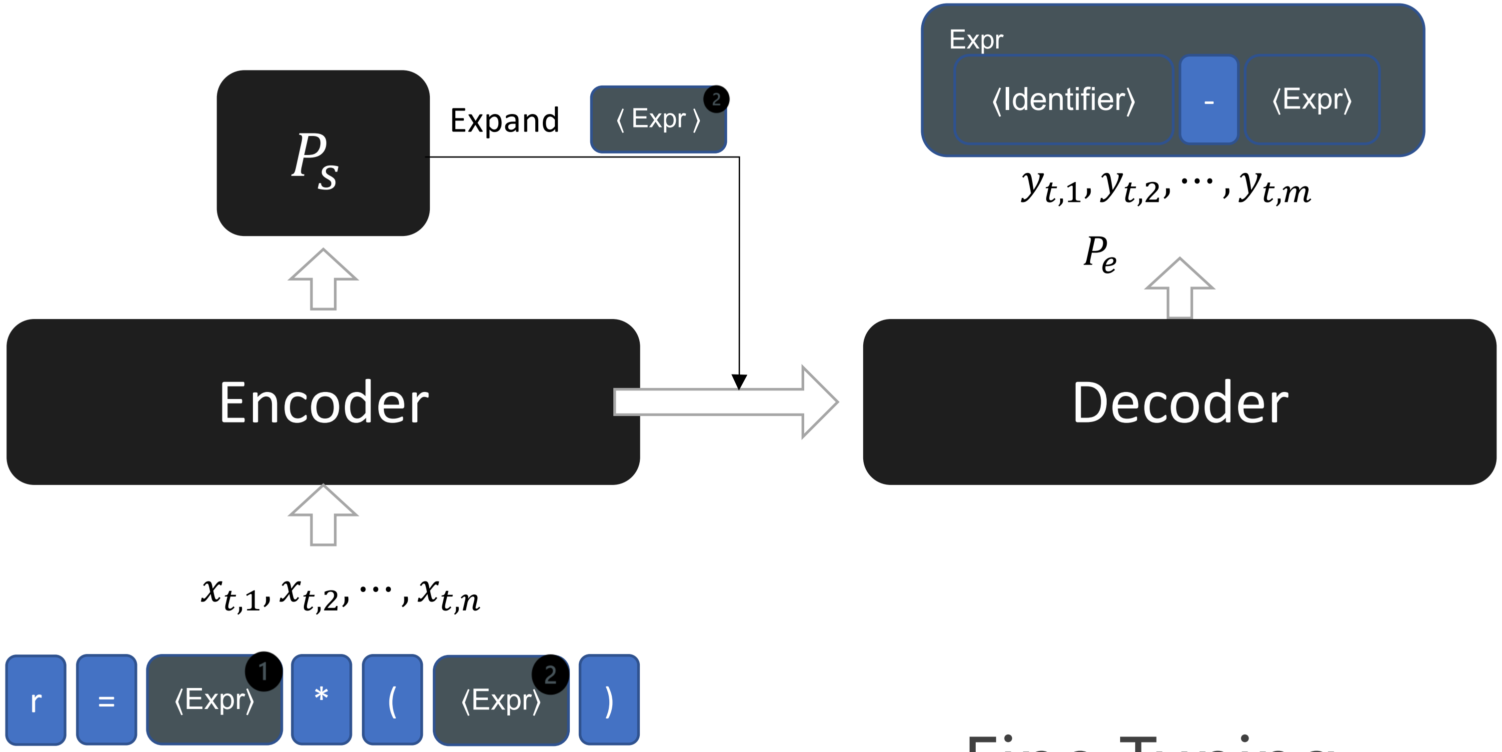
# Training Grammformers




- Pretraining

$$\mathcal{L}_{\text{pre, e}}\left(\mathbf{x}^{(t)}, (\mathbf{u}^{(t)}_{\circledcirc i})^*_{i\in\widetilde{N}(\mathbf{x}^{(t)})}\right) = \frac{1}{|\widetilde{N}(\mathbf{x}^{(t)})|}\cdot\sum_{i\in\widetilde{N}(\mathbf{x}^{(t)})} -\log P_e\left((\mathbf{u}^{(t)}_{\circledcirc i^{(t)}})^* \mid \mathbf{x}^{(t)}, i\right)$$

# Training Grammformers




- Fine Tuning

$$\mathcal{L}_{\text{train}}(\mathbf{x}_0, \mathbf{x}^*) = \left(r(\mathbf{x}_{\text{out}}, \mathbf{x}^*) - \tilde{r}(\mathbf{x}_0)\right) \sum_{t=0}^{T} \left(-\log P_s(i_t|\mathbf{x}_t) - \mathbb{I}(i_t \neq \oslash) \log P_e(\hat{\mathbf{y}}_{t \circledcirc i_t} | \mathbf{x}_t, i_t)\right)$$

Rennie, et al. 2017

# The Trade-offs in Sketch Generation

## Accurate

*Predict a sketch that matches the ground-truth.*

```
ap.add_argument(“--foo”, action=“store_true”)
ap.add_argument(▧, action=“store_false”)
ap.add_argument(▧, required=▧)
```

## Specific

*Predict a sketch that is as concrete as possible.*

```
ap.add_argument(▧, action=“store_true”)
ap.add_argument(▧, action= ▧)
ap.add_argument(▧, ▧)
ap.add_argument(▧)
ap.▧(▧, action=“store_true”)
▧.add_argument(▧, action=“store_true”)
▧.▧(▧)
```

# Evaluation — Regex Accuracy

$$\textsc{RegexAcc}(\hat{s}, s^*) \triangleq \frac{\texttt{nTerm}(\hat{s})}{\texttt{nTerm}(s^*)} \cdot \texttt{matches}(\texttt{toRegex}(\hat{s}), s^*)$$

$s^*$ = `ap.add_argument('--experimental', action="store_true")`

| $\hat{s}$ | | Regex Accuracy |
|---|---|---|
| | `ap.add_argument(▧, action="store_true")` | 9/10 |
| | `ap.add_argument(▧, action= ▧)` | 8/10 |
| | `ap.add_argument(▧, ▧)` | 6/10 |
| | `ap.add_argument(▧, action="store_false")` | 0 |
| | `ap.add_argument(▧, required=▧)` | 0 |

# Evaluation — REWARD

$$r(\hat{\mathbf{y}}, \mathbf{y}^*) = \frac{1}{2}\left(\textsc{RegexAcc}(\hat{\mathbf{y}}, \mathbf{y}^*) + \textsc{Rouge}_{\text{F1}}(\textsc{EraseHoles}(\hat{\mathbf{y}}, \mathbf{y}^*))\right)$$

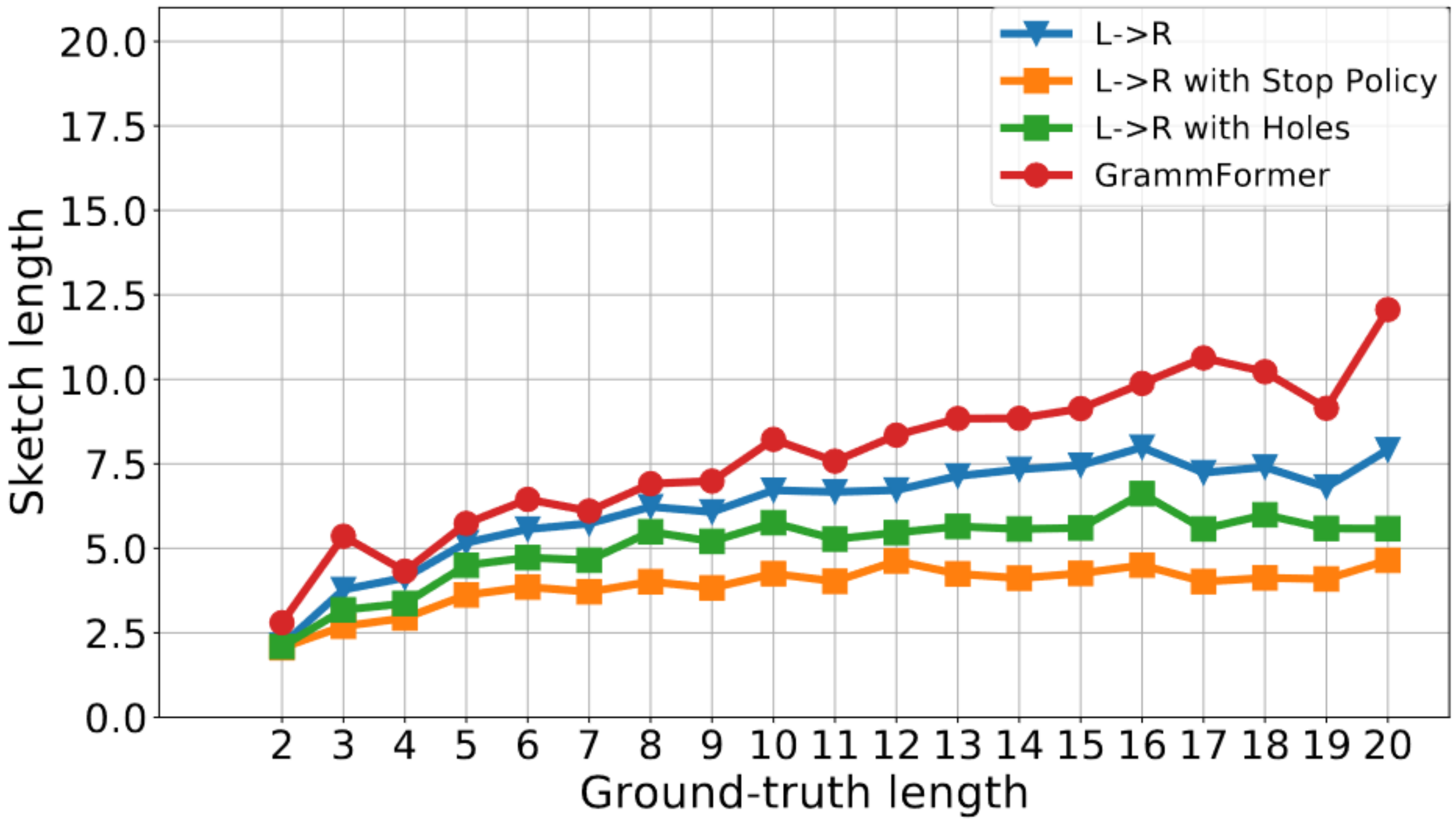
# Evaluation

C#

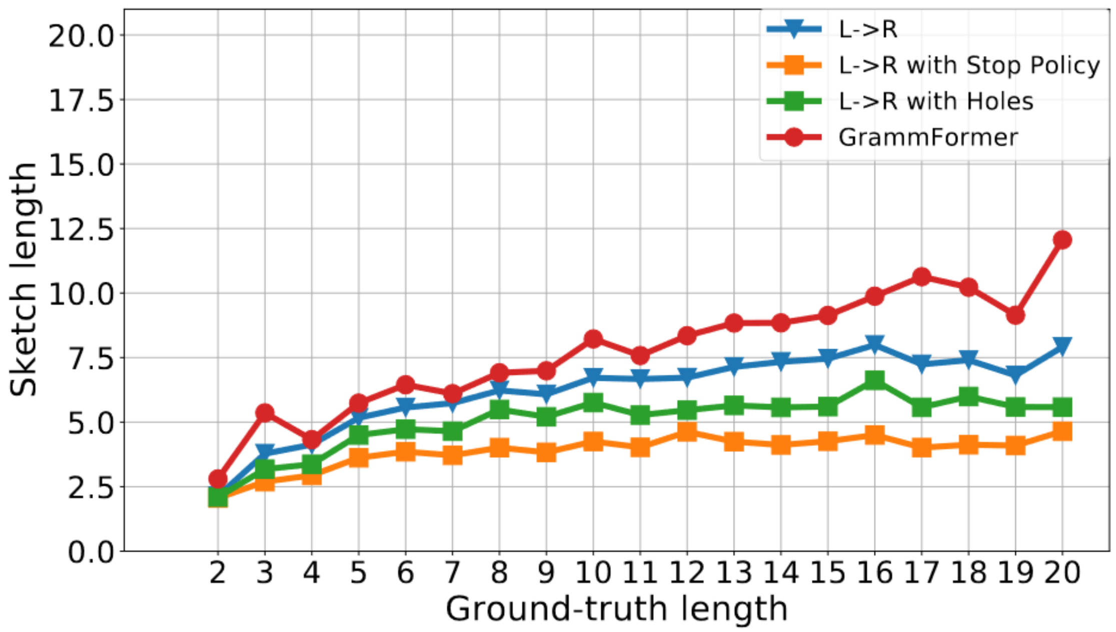
| Model | RegexAcc@1 | RegexAcc@5 | Avg Gen Length |
|---|---|---|---|
| L→R | 42% | 47% | 7.1 |
| L→R+⃠ | 45% | 54% | 5.3 |
| LM+▧ | 44% | 54% | 6.3 |
| **Grammformer** | **47%** | **59%** | **7.5** |

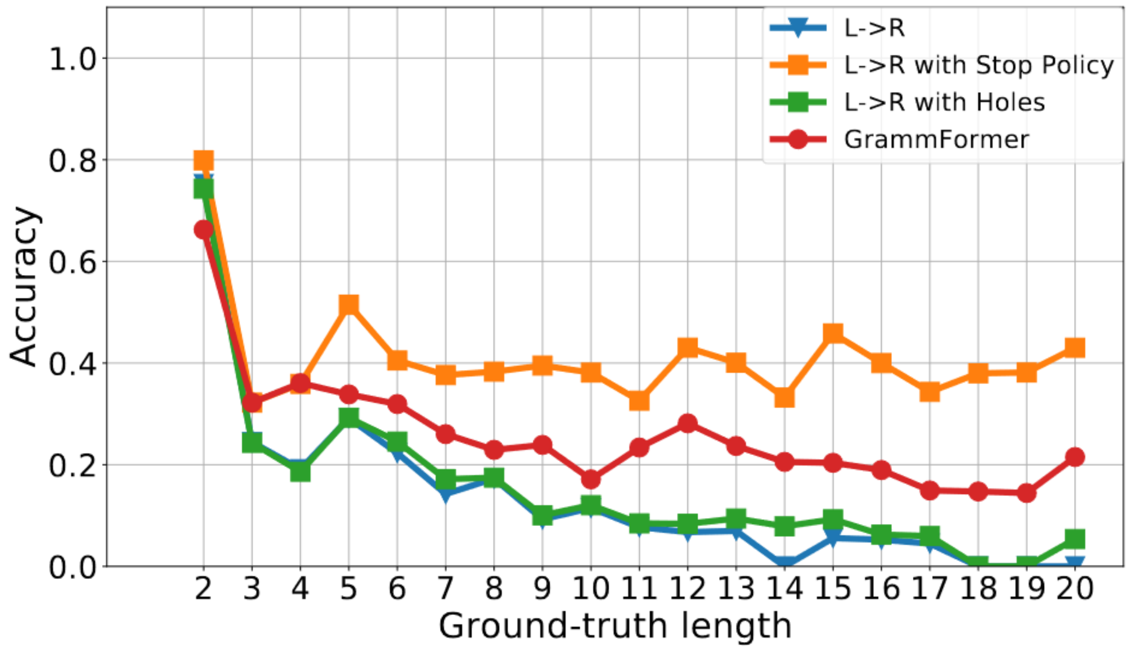
- 100% correct but with 47% of tokens specified.

# Evaluation


L->R
L->R with Stop Policy
L->R with Holes
GrammFormer
Sketch length
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Ground-truth length

# Evaluation


L->R
L->R with Stop Policy
L->R with Holes
GrammFormer
Sketch length
Ground-truth length
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


L->R
L->R with Stop Policy
L->R with Holes
GrammFormer
Accuracy
Ground-truth length
1.0
0.8
0.6
0.4
0.2
0.0
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

# Sketch Completion

```python
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("--release", action="store_true")
ap.add_argument("--prerelease", action="store_true")

ap.add_argument(▧, action="store_true")

```

**Learning to Generate Code Sketches.** *Guo, Svyatkovskiy, Yin, Duan, Brockschmidt, Allamanis. 2021*

# Copilot/OpenAI Codex

```python
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("--release", action="store_true")
ap.add_argument("--prerelease", action="store_true")
ap.add_argument("--version", action="store_true")
```

# Grammformer

```python
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("--release", action="store_true")
ap.add_argument("--prerelease", action="store_true")

ap.add_argument(▧, action="store_true")
```

Ground Truth:
ap.add_argument(“—-experimental”, action=“store_true”)

## Copilot/OpenAI Codex

```python
import sys
import os
import platform

if platform.system() == 'Linux':
    os.system('clear')
elif platform.system() == 'Windows':
    os.system('cls')

target = sys.argv[1]
print "Target: " + target +/-
```

## Grammformer

```python
import sys
import os
import platform

if platform.system() == "Linux":
    os.system('clear')
elif platform.system() == "Windows":
    os.system('cls')

target = sys.argv[1]
▧ = sys.argv[2]
```

Ground Truth:
ID = sys.argv[2]

# Thank You!

Questions

# Collaborators:

- Miltos Allamanis, Mark Brockshmidt, Google Research
- Daya Guo, Nan Duan, Microsoft Research Asia