Some existing program synthesizers consider many "obviously bad" candidate programs:

```java
String uppercase(String str) {
    int var1 = 0;
    String var2 = "";
    str = var2;
    return "";
}
```

**Objective:** generate more natural candidate programs (methods) for the synthesizer

### Task Setup

**Input:** types in method signature

**Output:** sequence of tokens

**Datasets:**
1. **GitHub:** ~10,000 methods scraped
2. **Synthesizer:** ~500 solutions plus ~3,000 "helpful" methods, with weights, for 90 different tasks. Train/dev/test split by task.
3. **Solutions:** subset of only the ~500 solutions

Variable names are canonicalized (e.g., `arg1`, `var2`). Some types of tokens are grouped, e.g., `230` becomes `<NumberLit>`. Vocab size of 100.

Example training pair:

- **Input:** `long <Class>`
- **Output:** `return ( arg1 == null ) ? <NumberLit> : arg1 . <Field> ;`

### Architecture:

- (previous token, function signature)
- 64-dimensional encodings of tokens
- 2-layer LSTM (512 hidden units per layer)
- FC softmax layer (outputs token probs)

### Loss Function:

Negative weighted LL of the dataset, normalized by the sequence length

\[
\frac{1}{\sum_{i=1}^{m} y^{(i)}} \sum_{i=1}^{m} \sum_{j=1}^{n} \log p \left( y^{(i)}(j) \right)
\]

### Transfer Learning:

- **GitHub** dataset is large but doesn't contain many interesting control structures
- **Synthesizer** dataset is small but is exactly the "style" of code we want to generate
- **Transfer** from **GitHub** to **Synthesizer**

### Results:

<table>
<thead>
<tr>
<th>Model</th>
<th>Syn-Train</th>
<th>Syn-Test</th>
<th>Sol-Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub</td>
<td>1.4 (69%)</td>
<td>1.3 (71%)</td>
<td>1.3 (72.4%)</td>
</tr>
<tr>
<td>Synthesizer</td>
<td>0.5 (84%)</td>
<td>1.1 (68%)</td>
<td>1.1 (69.4%)</td>
</tr>
<tr>
<td>Solutions</td>
<td>0.9 (73%)</td>
<td>1.3 (61%)</td>
<td>1.3 (62.6%)</td>
</tr>
<tr>
<td>Transfer-Syn</td>
<td>0.3 (90%)</td>
<td>1.0 (74%)</td>
<td>0.9 (75.1%)</td>
</tr>
<tr>
<td>Transfer-Sol</td>
<td>0.5 (86%)</td>
<td>1.1 (73%)</td>
<td>1.0 (74.6%)</td>
</tr>
</tbody>
</table>

### Example Generated Programs:

- **Signature:** `<Class>[] <Class> <Class>[]`
  
  ```java
  for (int i1 = <NumberLit>; i1 < Ineq arg2.<Field>; i1++) {
    arg2[i1] = arg1.<Method>(arg2);
  }
  return arg2;
  ```

- **Signature:** `int Object String`
  
  ```java
  if (arg1 == null) { arg2 = <NumberLit> ; }
  return <Class> . <Method>(arg1.<Method>());
  ```

### Main Conclusions:

- **Transfer learning** results in the best models
- Training on full **Synthesizer** dataset boosts performance even when tested on **Solutions**
- The model generates natural-looking code
- The generated code doesn't always compile.
- Most common errors:
  - Not understanding types
  - Incorrect variable names
  - Extraneous or unmatched parens/braces
- Token encodings help the model generalize

### Future Work:

- Generate a tree instead of a sequence
- Force the model to follow **language rules** by only sampling from allowable options at each step, possibly with **beam search**