Building Developer Assistants that Think Fast and Slow

Mayur Naik
University of Pennsylvania

Joint work with:

Pardis Pashakhanloo and Aaditya Naik
Hanjun Dai and Petros Maniatis
Thinking, Fast and Slow*

**System 1 (Fast)** "operates automatically and quickly, with little or no effort and no sense of voluntary control"

Examples:
- determine that an object is at a greater distance than another
- localize the source of a specific sound
- display disgust when seeing a gruesome image
- solve 2+2=?
- read text on a billboard
- drive a car on an empty road
- understand simple sentences

**System 2 (Slow)** "allocates attention to the effortful mental activities that demand it, including complex computations"

Examples:
- direct your attention towards someone at a loud party
- count the number of A's in a certain text
- park into a tight parking space
- determine the price/quality ratio of two washing machines
- determine the validity of a complex logical reasoning
- solve 17 × 24

* 2011 book by psychologist and Economics Nobel Laureate, Daniel Kahneman
Two Kinds of Developer Assistants

<table>
<thead>
<tr>
<th>System 1 (Fast)</th>
<th>System 2 (Slow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;operates automatically and quickly, with little or no effort and no sense of voluntary control&quot;</td>
<td>&quot;allocates attention to the effortful mental activities that demand it, including complex computations&quot;</td>
</tr>
</tbody>
</table>

- Based on **Deep Neural Network Models**
- Trained on "Big Code"
- Emergent: ~ 2018 – present
- Example: code completion using Github and OpenAI's **Copilot** (2021)

- Based on **Symbolic Reasoning Systems**
- Written by Human Experts
- Conventional: 1970's – present
- Example: code analysis using Semmle's **CodeQL** (2006; -> Github since 2019)
Two Prototypical Tasks

**System 1**: Detecting Variable Misuses
- Introduced in 2018
- Incorporated into Visual Studio IntelliCode

**System 2**: Detecting SQL Injections
- First appeared ~ 1998
- Remains in Top-3 in OWASP Top-10

Exploits of a Mom https://xkcd.com/327
Many Uses of Developer Assistants

- Code Completion
- Code Generation
- Linting
- Test Generation
- Program Merging
- Bug-Finding
- Fuzzing

- Type Inference
- Code Search
- Program Repair
- Optimization
- Code Captioning
- Code Summarization
- Decompilation

- Deobfuscation
- Verification
- Fault Localization
- Code Review
- Refactoring
- Code Migration
- Clone Detection

... and growing!

Should I use System 1 or System 2 for my task?
System 2: Challenges

• Vague or tedious specifications

• Undecidability of checking
  • "Halting Problem"

• Scalability (time, memory, ...)

• Real-world constraints
  • Missing libraries
  • Program must compile/run/...
Example: Global Taint Analysis

Inter-procedural dataflow analysis for taint tracking is expensive

In CodeQL, 21 queries rely on this analysis:

- SQL Injection
- XPath Injection
- Regex Injection
- Log Injection
- Command-line Injection
- LDAP Injection
- Unsafe Deserialization
- Stack-trace Exposure
- ...

When LoC > 1M, CodeQL timeouts after 6 hours

Applied to top 20 most popular projects in Google's repository on GitHub with 12K-2.5M LoC of Python
Talk Roadmap

- System 1: Overview and Survey
- System 1: Challenges
- The Emerging Role of System 2
- Concluding Remarks
Example: Detecting Variable Misuses

```csharp
public readonly static Thickness multilinePadding = new Thickness(0.0, 1.0);

public static List<Rect> GetRectanglesFromBounds(List<TextBounds> bounds)
{
    var newBounds = new List<Rect>(bounds.Count);
    foreach (var b in bounds)
    {
        double x1 = b.Left - padding.Left;
        double x2 = b.Right + padding.Right;
        if (x1 < x2)
        {
            double y1 = b.TextTop - padding.Top;
            double y2 = b.TextBottom + padding.Bottom;
            newBounds.Add(new Rect(x1, y1, x2 - x1, y2 - y1));
        }
    }
    return newBounds;
}

public static void Main()
{
    if (rectangle != null)
    {
        return;
    }
    // Do something
}
```
How Does it Work?

Training Set

Learning Algorithm

Model Representation

For one or more epochs, or until approx. cost minimum is reached:
For training sample $i$:
For each weight $w_j$:

$$w_j \leftarrow w_j - \eta \frac{\partial J^{(i)}(w)}{\partial w_j}$$

```java
double y1 = ...
double y2 = ...
... new Rect(x1, y1, x2-x1, y2-x1)
```
Model Representation

What program information to capture?

How to embed program information?

"Variable misuse detected on line 32 column 5"

Cancerous

Healthy

misleading

factual
**Program = Sequence of Lexical Tokens**

\[
\text{double } y_1 = \ldots \\
\text{double } y_2 = \ldots \\
\ldots \text{ new Rect}(x_1, y_1, x_2-x_1, y_2-x_1) \ldots
\]

A popular example of sequence-based models is Transformer models for code generation.

If any of the \( k \) generated samples for a problem pass all the unit tests, the problem is considered solved. The fraction of solved problems is PASS\(@k\).

**Pros:**
- Scalable: efficient to train on large code corpuses

**Cons:**
- Fails to capture rich semantic patterns
- Fails to reason about long-range dependencies

<table>
<thead>
<tr>
<th>Model</th>
<th>PASS( @k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-NEO 1.3B</td>
<td>k = 1 7.47% 16.30%</td>
</tr>
<tr>
<td>GPT-NEO 2.7B</td>
<td>k = 1 6.41% 21.37%</td>
</tr>
<tr>
<td>GPT-J 6B</td>
<td>k = 1 11.62% 27.74%</td>
</tr>
<tr>
<td>TabNine</td>
<td>k = 1 2.58% 7.59%</td>
</tr>
<tr>
<td>CodeX-2.5B</td>
<td>21.36% 59.5%</td>
</tr>
<tr>
<td>CodeX-12B</td>
<td>28.81% 72.31%</td>
</tr>
</tbody>
</table>

**HumanEval Dataset**

M. Chen et al. "Evaluating large language models trained on code." arXiv, 2021
Program = Graph of AST + Semantic Edges

Pros:
- Can capture long-range dependencies via additional edges

Cons:
- Bottleneck problem

M. Allamanis, M. Brockschmidt, and M. Khademi. "Learning to represent programs with graphs." ICLR'18.
Program = Set of Paths in AST

double y1 = ...
double y2 = ...
... new Rect(x1, y1, x2-x1, y2-x1) ...

Pros:
• Combines benefits of both previous approaches

Cons:
• Only considers the AST, so:
  • A semantic-preserving code transformation may drastically change several paths
  • Small non-semantic-preserving changes might affect the paths too little

Program = Lexical Sequence + Semantic Edges

```java
double y1 = ...
double y2 = ...
... new Rect(x1, y1, x2-x1, y2-x1) ...
```

Pros:
• Incorporates "message passing" ability of GNNs into Transformers
• Relates elements to each other even if they are lexically far apart

Cons:
• Increased complexity and decreased scalability w.r.t. Transformers

C. Ying et al. "Do Transformers Really Perform Badly for Graph Representation?" Neurips'21.
System 1: Challenges

1. Input Challenge: Data Quality
   - Training Set
   - Learning Algorithm
   - Model Representation

2. Output Challenge: Correctness

3. Overall Challenge: Generalizability/Robustness
Input Challenge: Data Quality

• What does high-quality data mean?

  Amount  Noise

  Coverage  Bias

• Example 1: Naively mining large code corpora

Example 2: Using synthetic instead of real bugs

Trained five models on 700K instances from Github

Synthetic: ETH Py150 Open (378K instances)

Test set: vs.

Real-world: Github commits (398 instances)

Task: Detecting Variable Misuses in Python functions

```
double y1 = ...
double y2 = ...
... new Rect(x1, y1, x2-x1, y2-y1)
```

System 2 for Ensuring Data Quality

Can we use or chain program reasoning tools to generate new data or clean up existing datasets?

Fuzzers, compilers, parsers, type checkers, test generators, decompilers, static analyzers, symbolic execution, theorem provers, program reducers, constraint solvers, ...

Example: Data Generation for Inferring Preconditions in Java

```java
public Stack(int s) {
    maxSize = s;
    stackArray = new long[maxSize];
    top = -1;
}
```

Regression Test Generator

```
boolean Stack_pre(int s) {
    maxSize = s;
    if (maxSize < 0) return false;
    return true;
}
```

Generated dataset:

\{<M_1, S_1>, <M_2, S_2>, ...\}

Output Challenge: Correctness

**Example 1:** Code Completion (Github Copilot)
Correctness Goal: Never introduce security vulnerabilities

```java
// allocate memory for n library members. n is provided by the user.
int n;
printf("Enter the number of library members: ");
scanf("%d", &n);
struct library *lib = malloc(n * sizeof(struct library));
```

CWE-20: Improper Input Validation
Code fails to check user-provided value, e.g., it can be −1 or too large.

**Example 2:** Code Migration (Facebook Transcoder)
Correctness Goal: Generate semantically equivalent program

```python
def pow_func(a, b):
    return a**b
```

TransCoder

```java
public Integer pow_func(Integer a, Integer b) {
    return a.pow(b);
}
```

System 2 for Ensuring Correctness

Example 1: AlphaCode

Input
The first line contains a single integer $q$ ($1 \leq q \leq 10^5$) the number of test cases. The first line of each test case contains the string $s$ ($1 \leq |s| \leq 10^5$). Each character of $s$ is a lowercase English letter.

Output
For each test case, print “YES” if you can obtain the string $t$ by typing the string $s$ and replacing some characters with presses of “Backspace” button, or “NO” if you cannot.

Example Input
4
ababa
ba
ababa
aaaa
ababa
ababa

Example Output
YES
NO
NO
YES

Explanation
In order to obtain “ba” from “ababa”, you may press Backspace instead of typing the first and the fourth characters.

There’s no way to obtain “bb” while typing “ababa”.

There’s no way to obtain “aaaa” while typing “ababa”.

In order to obtain “ababa” while typing “ababa”, you have to press Backspace instead of typing the first character, then type all the remaining characters.

System 2 for Ensuring Correctness

Example 1: AlphaCode

<table>
<thead>
<tr>
<th>Approach</th>
<th>Validation Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10@1k</td>
<td>10@10k</td>
</tr>
<tr>
<td>9B</td>
<td>16.9%</td>
<td>22.6%</td>
</tr>
<tr>
<td>41B</td>
<td>16.9%</td>
<td>23.9%</td>
</tr>
<tr>
<td>41B + clustering</td>
<td>21.0%</td>
<td>26.2%</td>
</tr>
</tbody>
</table>
"... further gains can be realized by generating 100 samples per problem ... by selecting the sample that passes the unit tests (77.5% solved). All samples are generated with temperature 0.8."

"... we use temperature sampling (with temperature 0.5) to generate 80 samples of code and then execute each of them against tests for semantic correctness"
Example 1: Adversarial semantics-preserving perturbations

Overall Challenge: Generalizability/Robustness

Example 2: Skeletal Program Enumeration perturbations

Q. Zhang, C. Sun, and Z. Su. "Skeletal program enumeration for rigorous compiler testing." PLDI'17

Task: Detecting Unused Variables in Python functions
Example: CodeTrek

- CodeTrek represents a program as a relational database.
- CodeTrek leverages the per-language schema defined by Semmle to uniformly store lexical, syntactic, and semantic program information.

CodeTrek

With same neural architecture, **richness of semantic information** has significant impact on accuracy. Representation can be further enriched by providing more CodeQL queries!
Rich program information overwhelms most deep learning techniques!

**Solution:** sampling walks over the relational graph

CodeTrek can **learn to bias** these random walks. The learned biases for bug finding tasks perform as good as or better than biases that the expert domain selects!

P. Pashakhanloo et al. "Learning to Walk over Relational Graphs of Source Code." DL4C@ICLR’22
How to embed a sampled walk?

1. Identify the node types and values:
   - **node types**: access, expr, stmt
   - **node values**: VarUse, StmtList, Sub

2. Construct the edge types:
   - **edge types**: access-expr, expr-stmt, stmt-expr

3. Apply positional encoding to the node and edge values.

4. Use the Transformer Encoder to process the encoded data.

5. Pool the output to obtain the Embedding Vector of the Sampled Walk.
CodeTrek

Advantages:

1. Scaling deep learning on modules instead of single functions.
   - **Example:** Exception Type Prediction
     - | Granularity       | CodeTrek | GGNN | Code2Seq | GREAT | CuBERT |
       |-------------------|----------|------|----------|-------|--------|
       | Single Function   | 65%      | 51%  | 51%      | 68%   | 69%    |
       | Single Module     | **63%**  | 28%  | 30%      | 44%   | 42%    |

   - **Example:** Unused Variable Detection
     - ```python
def foo():
a = 1
b = 1
return a
```
DAMP replaces an original variable from $P$ with an adversarial variable to obtain $P'$.

- $\tau' = \bar{v} - \eta \cdot \nabla_\theta J(\theta, x, y_{bad})$
- $\text{argmax}$

$J$: original loss function

$J_{adv}$: adversarial loss function

Adversarial Training

How to quantify robustness?

Percentage of original samples whose labels are correctly predicted and not changed after applying DAMP.
Central Idea: Expose PL semantics through weak supervision from a static analyzer.

Task: Code Generation

Training Phase:

Each generation step is conditioned on symbolically-derived attributes of the current context.

Percent of Static Checks Passed

<table>
<thead>
<tr>
<th></th>
<th>CODEX</th>
<th>CODEGPT</th>
<th>NSG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage of parsing</td>
<td>96.41%</td>
<td>97.08%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Pass all checks</td>
<td>47.49%</td>
<td>67.73%</td>
<td>86.41%</td>
</tr>
</tbody>
</table>

- No undeclared variable access
- Valid class variable access
- No uninitialized objects
- No variable access error
- Object-method compatibility
- Return statement type
- No unused variables

Contrastive Learning

Example: ContraCode

Programs with the same functionality should have similar representations.

ContraCode learns representations by pre-training an encoder to retrieve equivalent, transformed programs.

Given a program,

```
function (len) {
    for (i = 0; i < len, i++) {
        ...;
    }
}
```

Maximize similarity with equivalent programs

```
function (n) { while (i < n) { ... } }
```

```
function (str, len) { return str.slice(0, len); }
```

```
function f(n) { return n < 2 ? 1 : f(n-1) + f(n-2); }
```

```
function (arr) { for (i of arr) { ... } }
```

Minimize similarity with functionally different programs

```
for (i = 0; i < length; ++i) {
    ...
}
```

```
while (j < length) {
    ...
    j++;
}
```

```
if (i < 10) {
    a = "code";
}
```

Concluding Remarks

Developer assistants of the present:

• Those based on **System 1**:
  - aim to overcome challenges of those based on System 2
  - but ineffective or risky

• Those based on **System 2**:
  - have proven their capabilities and are here to stay
  - but also face many hard challenges

Developer assistants of the future?

• Combining System 1 and 2 for best of both worlds
  - this talk: emerging works showing how System 2 is improving System 1
  - but area is ripe for lots of innovation!
  - also, not covered: how System 1 is improving System 2