Jigsaw

Large Language Models meet Program Synthesis

Presented by: Andras Geiszl

Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram Rajamani, Rahul Sharma
Who uses LLMs for code?

LLM = **Large Language Model** (Copilot, Codex, GPT-4, etc.)

Generate code in python that prints "Hello World!".

Here’s the Python code that prints "Hello World!" to the console:

```python
print("Hello World!")
```

When you run this code, it will output "Hello World!" in the console.
The problem

Can you draw me a flower?

Certainly! Here is a simple flower that I drew for you:

I hope you like it! Is there anything else I can help with?

https://github.com/giuen95/chatgpt-failures
The problem

<table>
<thead>
<tr>
<th>country</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: India</td>
<td>1</td>
</tr>
<tr>
<td>Name: USA</td>
<td>2</td>
</tr>
<tr>
<td>UK</td>
<td>3</td>
</tr>
</tbody>
</table>

Input

Output

Code from LLM

df['c'] = df['c'].str.replace('Name: ', '')
The problem

Code from LLM

```python
dfout = dfin.drop_duplicates(subset=['inputB'])  # Model
```

Post-processing

```python
dfout = dfin.drop_duplicates(subset=['inputB'], keep=False)  # Correct
```

❌ ✅
Previous work: SLANG [Vechev et al.] (2016)

Code completion by predicting **probability of sequences**

First approach that builds **probabilistic models of API calls** extracted via static analysis.

First approach that uses **RNNs for program prediction** tasks
Previous work: SLANG

Probabilistic code completion using the n-gram model and RNNs

Extract symbols using static analysis

Complete code in partial program by predicting sentences
Previous work: SLANG

Accuracy: ~30-40%

<table>
<thead>
<tr>
<th>Analysis</th>
<th>No alias analysis</th>
<th>With alias analysis</th>
<th>With alias analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3-gram</td>
<td>3-gram</td>
<td>RNN + 3-gram</td>
</tr>
<tr>
<td>Language model type</td>
<td>1%</td>
<td>10%</td>
<td>all</td>
</tr>
<tr>
<td>Training dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1 (20 examples)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal in top 16</td>
<td>11</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>Goal in top 3</td>
<td>10</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>Goal at position 1</td>
<td>7</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Task 2 (14 examples)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal in top 16</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Goal in top 3</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Goal at position 1</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Task 3 (50 random ex.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal in top 16</td>
<td>13</td>
<td>27</td>
<td>36</td>
</tr>
<tr>
<td>Goal in top 3</td>
<td>13</td>
<td>23</td>
<td>32</td>
</tr>
<tr>
<td>Goal at position 1</td>
<td>13</td>
<td>16</td>
<td>25</td>
</tr>
</tbody>
</table>
Previous work: AutoPandas [Bavishi et al.] (2019)

Generates programs with 2-3 functions based on I/O examples (DataFrames)

Uses generators for enumerating over the Pandas API

Uses Graph Neural Networks (GNNs) to predict most likely function sequences and arguments.
Previous work: AutoPandas

Generate candidates, then check their output

def synthesize(input, output, max_len):
    generator = generate_candidates(input, output, max_len)
    while (not generator.finished()):
        candidate = next(generator)
        if candidate(input) == output:
            return candidate
Pick a sequence of functions

Select function arguments

Combine functions
Previous work: AutoPandas

Introduces **smart operators** that make neural network queries on the fly.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select(domain)</td>
<td>Returns a single item from domain</td>
</tr>
<tr>
<td>Subset(domain)</td>
<td>Returns an unordered subset, without replacement, of the items in domain</td>
</tr>
<tr>
<td>OrderedSubset(domain)</td>
<td>Returns an ordered subset, without replacement, of the items in domain</td>
</tr>
<tr>
<td>Sequence(len)(domain)</td>
<td>Returns an ordered sequence, with replacement, of the items in domain</td>
</tr>
<tr>
<td></td>
<td>with a maximum length of len</td>
</tr>
</tbody>
</table>

**Rank(Domain, Context)** – per-operator ranking of selected functions/arguments using Graph Neural Networks.
Previous work: AutoPandas

Accuracy: ~65% (?)
Large Language Models (LLMs)

- 12 billion parameters
- 7 billion to 65 billion parameters
- 175 billion parameters
Large Language Models (LLMs)

Take sequence of words as an input and predict the next word

Prompt the model with text

Model outputs text prediction
**Jigsaw: Large Language Models meet Program Synthesis**

Multimodal input: query + I/O examples

Runs code and **checks** if it passes
How does Jigsaw work?

1. Preprocessing
2. Code generation
3. Post-processing
How does Jigsaw work?

Treat language model as a **black box**

Plug in **any language model**
Codex, GPT-3, etc.

Get better performance by **updating the model**
Preprocessing

Process input to be fed into the LLM

```python
# Prime the model with examples
# Examples to train a English to French translator
# Input to the model

import gpt3

gpt3 = GPT(engine="davinci", temperature=0.5, max_tokens=100)
gpt3.add_example(Example("What is your name?", "quel est votre nom?"))
gpt3.add_example(Example("What are you doing?", "Quel faites-vous?"))
gpt3.add_example(Example("How are you?", "Comment allez-vous?"))

# Prompt the model

prompt3 = "where are you?"
output3 = gpt3.submit_request(prompt3)

# Model output

output3.choices.text

Output: Où êtes-vous?
```
Preprocessing

Compare contexts using **tf-idf similarity** or **transformer similarity**
Post-processing

3 types of common errors

**Reference errors**

```r
df2.merge(df1) instead of df1.merge(df2)
```

**Argument errors**

```r
cates(subset=['inputB']) # Model
cates(subset=['inputB'], keep=FALSE) #
```

**Semantic errors**

```r
duplicated() # Model
duplicated().sum() # Correct
```
Reference errors

Model output uses **incorrect variable names**

Developer uses **non-standard** variable names

E.g., \texttt{g1, g2} instead of \texttt{df1, df2} for DataFrames

Model **confuses** variable names

E.g., \texttt{df2.merge(df1)} instead of \texttt{df1.merge(df2)}
Variable transformations

Try permutations and combinations of variable names

\[
\begin{align*}
\text{df1.merge(df1)} & \quad \times \\
\text{df1.merge(df2)} & \quad \checkmark \\
\text{df2.merge(df1)} & \quad \times \\
\text{df2.merge(df2)} & \quad \times
\end{align*}
\]
Argument errors

Model output uses **incorrect arguments**

a.) Query – Drop all the rows that are duplicated in column **‘inputB’**

```
dfout = dfin.drop_duplicates(subset=['inputB'])  # Model
```

```
dfout = dfin.drop_duplicates(subset=['inputB'], keep=False)  # Correct
```

b.) Query – Replace Canada with **CAN** in column country of df

```
df = df.replace({'Canada': 'CAN'})  # Model
```

```
df = df.replace({'country': {'Canada': 'CAN'}})  # Correct
```
Argument transformations

Systematically search through the arguments of an inferred argument space

1. Extract method names

natural language text input

column names from the dataframe schema

arguments in the PTLM output

variables in scope
Argument transformations

Systematically search through the arguments of an inferred argument space

2. Generate program line candidates using the same approach as AutoPandas

**Modifications:**

Instead of using GNNs, extract method names from LLM output

Extend generators to consider complex data types (lists, dictionaries)

Extend set of APIs to those that return Pandas Series types
Semantic errors

Model output is **slightly different** from the correct solution

a.) Query – Select rows of dfin where value in bar is <38 or >60

```python
dfout = dfin[dfin['bar']<38 | dfin['bar'] > 60] # Model
```

```python
# Correct
dfout = dfin[(dfin['bar']<38) & (dfin['bar'] > 60)]
```

*Mistake – missing parentheses change precedence and cause exception*

b.) Query – Count the number of duplicated rows in df

```python
out = df.duplicated() # Model
```

```python
# Correct
out = df.duplicated().sum()
```

*Mistake – missing required summation to get the count*
**Semantic errors**

Model output is *slightly different* from the correct solution

```python
train = data[data.index.isin(test.index))}
```

instead of the following correct code with the bitwise not operator:

```python
train = data[~data.index.isin(test.index)]
```

Same errors are *repeatedly made* by LLM
AST-to-AST transformations

Need to learn general representation, so that it can be repeated with different variables/constants (needs diverse code examples)

1. Collect data from users correcting Jigsaw output
2. Cluster data points (code snippets) by similarity
3. Learn single AST-to-AST transformation for one cluster
AST-to-AST transformations

Greedy, heuristic-based, online clustering

1. For a **new datapoint**, decide if it’s in an **existing cluster** or to **create new**
2. If it’s in an **existing cluster**, try to **relearn transformation** to be more general
3. **Perturb** data points (change variable names) to prevent overfitting

Uses **Prose** framework to learn AST-to-AST transformations
Contributions: data sets

PandasEval1

- 68 Python Pandas tasks
- Single line of code, 2-3 functions
- Created by authors from StackOverflow

Example:
For every row in df1, update ‘common’ column to True if value in column ‘A’ of df1 also lies in column ‘B’ of df2

PandasEval2

- 21 Python Pandas tasks
- Single line of code, 2-3 functions
- Created by 25 users in 2 sessions (725 queries)

Example:
Results

Accuracy: fraction of specifications for which a correct program was synthesized + manual inspection

Run every evaluation three times and report mean accuracy

Report best accuracy using temperatures \{0, 0.2, 0.4, 0.6\}
Results

Context matters!

Pre- and post-processing improves accuracy significantly

Processing time is bottlenecked by the LLM inference (~7 out of 10 seconds)
Learning from user feedback

**Users submit correct code** in cases where Jigsaw is incorrect

**Context bank:** \{ (query 1, code example 1), (query 2, example 2), (query 3, example 3)... \}

**User submission:** (query, code example)

**Jigsaw output:** Jigsaw(query, context bank)

1. Update **context bank**
   1. Is Jigsaw output **correct** or **close to** the submitted code (edit distance)?
   2. Is it **not too similar** to another example in the bank (tf-idf distance)?
   3. If both are true, then **add sample** to the context bank
Learning from user feedback

Users submit correct code in cases where Jigsaw is incorrect

Context bank: { (query 1, code example 1), (query 2, example 2), (query 3, example 3)...

User submission: (query, code example)

Jigsaw output: Jigsaw(query, context bank)

2. Update transformations

1. Find all incorrect code generated by Jigsaw with small edit distance
2. Add them to the clustering
3. Learn incorrect to submitted AST-to-AST transformations
User feedback experiments

Perform evaluation on the PandasEval2 dataset separated to PandasEval2_S1 and PandasEval2_S2

Two experiments: use feedback for first part (PandasEval2_S1) to

update context bank (CS1 -> CS2; 243 seeded + 128 new)
learn AST-to-AST transformations (TS1 -> TS2)
User feedback experiments

Perform evaluation on the PandasEval2 dataset separated to PandasEval2_S1 and PandasEval2_S2

<table>
<thead>
<tr>
<th></th>
<th>PandasEval2_S1</th>
<th>PandasEval2_S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CS1-TS1</td>
<td>CS1-TS1</td>
</tr>
<tr>
<td>GPT-3</td>
<td>45.9 ± 0.4</td>
<td>35.1 ± 0.8</td>
</tr>
<tr>
<td>Codex</td>
<td>75.1 ± 0.5</td>
<td>69.0 ± 0.7</td>
</tr>
</tbody>
</table>

User feedback improves accuracy

Users were able to solve more (82%) tasks in the second experiment than in the first one (71%)
Comparison to AutoPandas

Uses **only I/O examples**, while Jigsaw also uses **natural language input**

```
Jigsaw Query
Filter rows of df where column 'A' mod 4 equals 1

<table>
<thead>
<tr>
<th>Input(s)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>foo</td>
<td></td>
</tr>
<tr>
<td>jig</td>
<td></td>
</tr>
<tr>
<td>bar</td>
<td></td>
</tr>
<tr>
<td>saw</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
```
Comparison to AutoPandas

**Does not support** Series operations, column assignments, dictionary and list generators

PandasEval1: 7/68 solvable  
Jigsaw **outperforms** AutoPandas on these

PandasEval2: 20/21 solvable

<table>
<thead>
<tr>
<th></th>
<th>AutoPandas [9]</th>
<th>PTLM</th>
<th>Jigsaw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset of Jigsaw datasets</td>
<td>16/27</td>
<td>20/27</td>
<td>23/27</td>
</tr>
<tr>
<td>AutoPandas dataset</td>
<td>17/26</td>
<td>15/26</td>
<td>19/26</td>
</tr>
</tbody>
</table>

LLM is worse, but **Jigsaw is better**!

AutoPandas had **3-minute timeout**
**Ablation study**

Evaluate effect of **number of contexts** and the **context selector**

Context selector: **TFIDF** and **TRANSFORMER**

<table>
<thead>
<tr>
<th></th>
<th>PandasEval1</th>
<th>PandasEval2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPT-3</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TFIDF</td>
<td>46.5 ± 4.8</td>
<td>32.4 ± 0.5</td>
</tr>
<tr>
<td>TRANSFORMER</td>
<td>47.1 ± 2.1</td>
<td>35.1 ± 0.7</td>
</tr>
<tr>
<td><strong>Codex</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TFIDF</td>
<td>69.1 ± 2.4</td>
<td>70.1 ± 0.1</td>
</tr>
<tr>
<td>TRANSFORMER</td>
<td>66.7 ± 0.7</td>
<td>72.2 ± 0.5</td>
</tr>
</tbody>
</table>

**Not sensitive** to context selector
Ablation study

Evaluate effect of **number of contexts** and the **context selector**

<table>
<thead>
<tr>
<th># Prompts</th>
<th>PandasEval1</th>
<th>PandasEval2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>47.5 ± 1.8</td>
<td>34.9 ± 0.9</td>
</tr>
<tr>
<td>4</td>
<td>47.1 ± 2.1</td>
<td>35.1 ± 0.7</td>
</tr>
<tr>
<td>8</td>
<td>48.0 ± 2.5</td>
<td>32.9 ± 0.6</td>
</tr>
<tr>
<td>Codex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>62.3 ± 0.7</td>
<td>71.8 ± 0.5</td>
</tr>
<tr>
<td>4</td>
<td>66.7 ± 0.7</td>
<td>72.2 ± 0.5</td>
</tr>
<tr>
<td>8</td>
<td>66.2 ± 1.2</td>
<td>72.4 ± 0.9</td>
</tr>
</tbody>
</table>

*No significant difference* between 4 and 8 prompts

*Both are better* than 1 prompt (and much better than no context)
Beyond pandas

Evaluate performance on TensorFlow tasks

Reuse variable transformations and manually evaluate semantic repair

<table>
<thead>
<tr>
<th>PTLM</th>
<th>Variable Name</th>
<th>Semantic Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/25</td>
<td>15/25</td>
<td>19/25</td>
</tr>
</tbody>
</table>
Evaluation and future work

- Datasets are small and **might not be representative** of all Pandas programs
- Experiments had **only 25 participants**

- Pre- and post-processing **drastically improves quality** of generated code

- In practice, code should have **high performance, be secure, respect licensing**
- Specifications can be **weak or ambiguous**, could be improved with pre-, postconditions, invariants, bounds, etc.
Why not use GPT to correct itself?

BioBootloader
@bio.bootloader

Today I used GPT-4 to make "Wolverine" - it gives your python scripts regenerative healing abilities!

Run your scripts with it and when they crash, GPT-4 edits them and explains what went wrong. Even if you have many bugs it'll repeatedly rerun until everything is fixed.
```python
import fire

def add_numbers(a, b):
    return a + b

def multiply_numbers(a, b):
    return a * b

def divide_numbers(a, b):
    return a / b

def calculate(operation, num1, num2):
    if operation == "add":
        result = add_numbers(num1, num2)
    elif operation == "subtract":
        result = subtract_numbers(num1, num2)
    elif operation == "multiply":
        result = multiply_numbers(num1, num2)
    elif operation == "divide":
        result = divide_numbers(num1, num2)
    else:
        print("Invalid operation")
    return result

if __name__ == "__main__":
    fire.Fire(calculate)
```

Errors:

```
Traceback (most recent call last):
 File "/Users/bio_bootloader/code/gpt-projects/buggy_script.py", line 32, in <module>
     fire.Fire(calculate)
File "/Users/bio_bootloader/code/gpt-projects/venv/lib/python3.10/site-packages/fire/core.py", line 141, in Fire
 component_trace = Fire(component, args, parsed_flag_args, context, name)
File ".../core.py", line 475, in Fire
CallWrapper(Wrapper(fire.Fire(function, **modargs), *args, **kwargs), context)
SyntaxError: \'return\' outside function scope
```

Explanations:

- The 'return' statement is outside of the function scope on line 31. It should be inside the 'calculate' function.

Changes:

```python
---
+++ @ -28,7 +28,7 @@ else:
-     print("Invalid operation")
+     return result
      -return result

if __name__ == "__main__":
    Changes applied. Retrying...
    Script ran successfully.
```

Output:

```
Traceback (most recent call last):
 File "/Users/bio_bootloader/code/gpt-projects/buggy_script.py", line 32, in <module>
     fire.Fire(calculate)
File ".../core.py", line 141, in Fire
 component_trace = Fire(component, args, parsed_flag_args, context, name)
File ".../core.py", line 475, in Fire
CallWrapper(Wrapper(fire.Fire(function, **modargs), *args, **kwargs), context)
SyntaxError: \'return\' outside function scope
```

Output:

```
Changes applied. Retrying...
```

```
Script ran successfully.
```
Why not use GPT to correct itself?

• No guarantees on GPT finding the problem
• No guarantees on time to fix

• Solution is much simpler
• Might work very well for easy fixes
• Gets better as LLM model gets better
Summary

- Generating **correct code** is **hard**

- Even if using LLMs, significant amount of **classical post processing** is required

- **In the future**, ideally model generating the code should **fix itself**
Question time!

Ask me a question about the paper "Jigsaw: Large Language Models meet Program Synthesis".

Sure! What is the main contribution of the paper "Jigsaw: Large Language Models meet Program Synthesis" and how does it differ from previous work in program synthesis?
References