Jigsaw

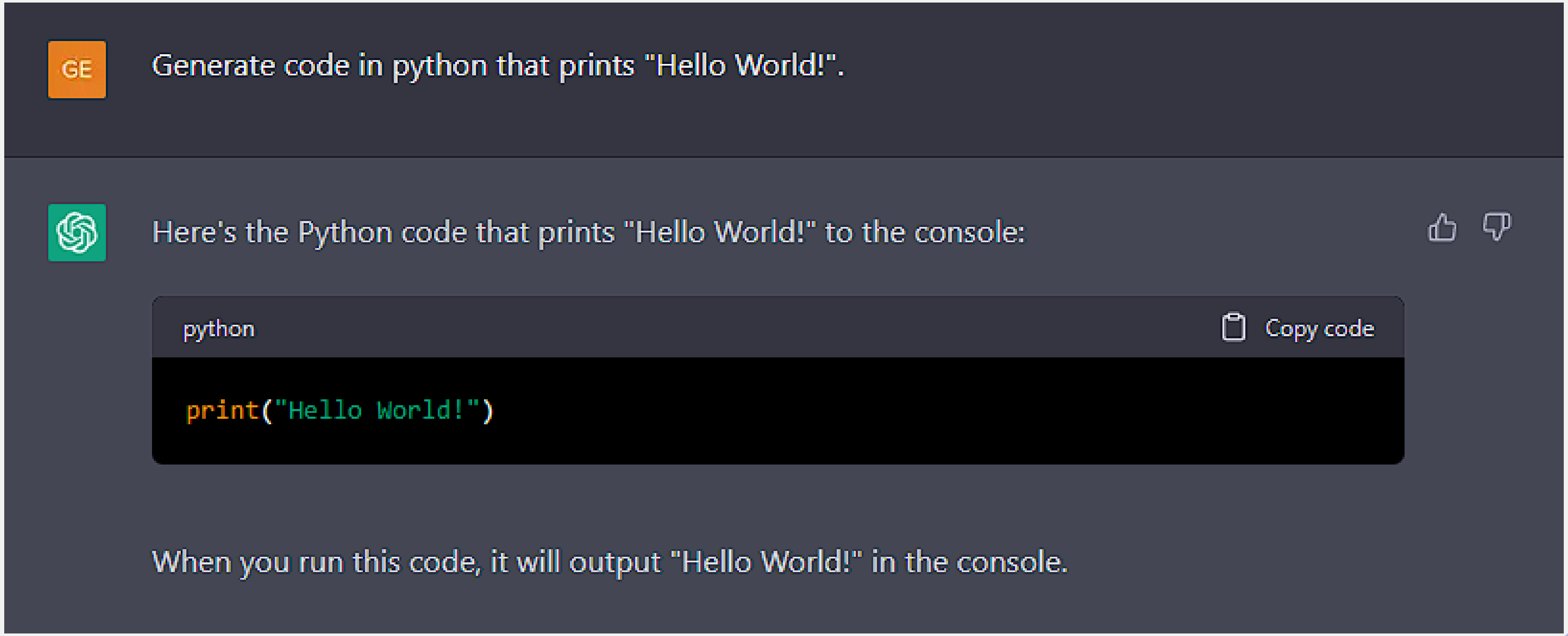
Large Language Models meet Program Synthesis

Presented by: Andras Geiszl

Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram Rajamani, Rahul Sharma

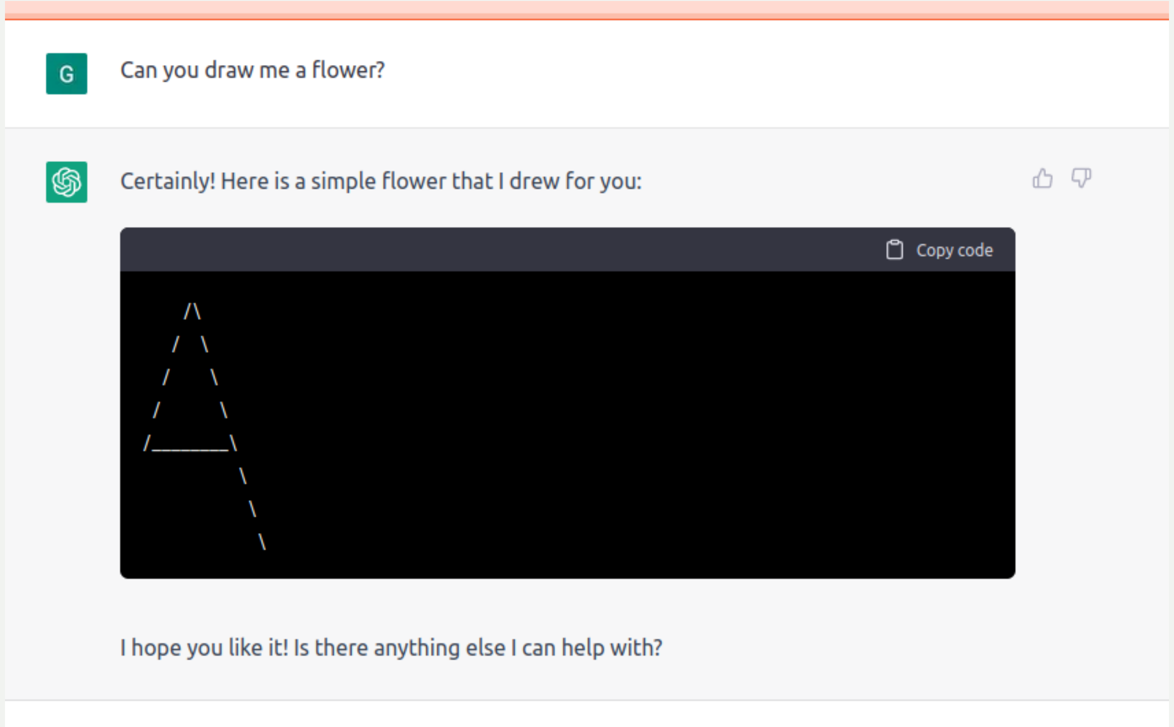
Who uses LLMs for code?

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



https://chat.openai.com/chat

The problem



https://github.com/giuven95/chatgpt-failures

The problem

Input

country	val
Name: India	1
Name: USA	2
UK	3

Output

country	val
India	1
USA	2
UK	3

Code from LLM

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

The problem

Code from LLM

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

Post-processing

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

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

Code completion by predicting **probability of sequences**

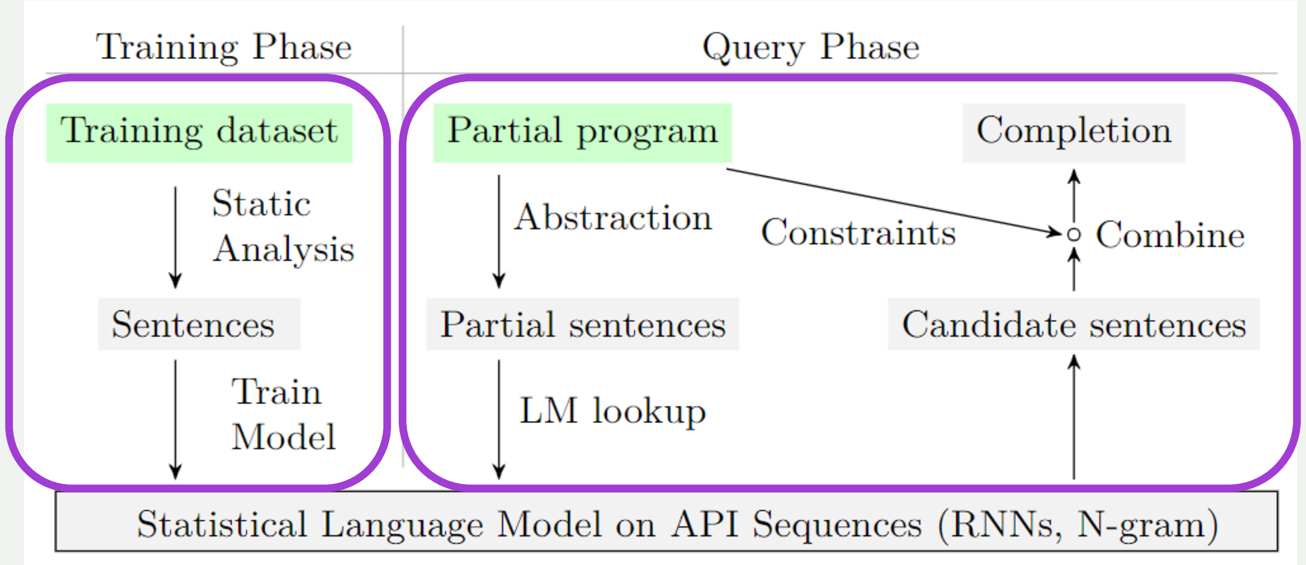
First approach that builds **probabilistic models of API calls** extracted via static analysis.

First approach that uses **RNNs for program prediction** tasks

[5]

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%**

Analysis	No alias analysis			With alias analysis			With alias analysis	
Language model type	3-gram			3-gram			RNN	RNN + 3-gram
Training dataset	1%	10%	all	1%	10%	all	all	all
Task 1 (20 examples)								
Goal in top 16	11	16	18	12	18	**20**	**20**	**20**
Goal in top 3	10	12	16	11	15	**18**	**18**	**18**
Goal at position 1	7	8	12	7	10	**15**	**14**	**15**
Task 2 (14 examples)								
Goal in top 16	3	5	7	10	10	**13**	**13**	**13**
Goal in top 3	3	4	6	8	8	**13**	**12**	**13**
Goal at position 1	3	3	5	6	6	**11**	**11**	**12**
Task 3 (50 random ex.)								
Goal in top 16	13	27	36	21	43	**48**	**48**	**48**
Goal in top 3	13	23	32	18	34	**44**	**40**	**45**
Goal at position 1	13	16	25	14	25	**31**	**27**	**31**

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.

[4]

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
```

```python
@generator
def generate_candidates(input, output, max_len):
    functions = [pivot, drop, merge, ...]
    function_sequence = Sequence(max_len)(functions, context=[input, output], id=1)
    intermediates = []
    for function in function_sequence:
        c = [input, *intermediates, output]
        if function == pivot:
            df = Select(input + intermediates, context=c, id=2)
            arg_col = Select(df.columns, context=[df, output], id=3)
            arg_idx = Select(df.columns - {arg_col}, context=[df, output], id=4)
            if isinstance(df.index, pandas.MultiIndex) and arg_idx is None:
                arg_val = None
            else:
                arg_val = Select(df.columns - {arg_col, arg_idx},
                                 context=[df, output], id=5)
            args = (df, arg_col, arg_idx, arg_val)

        elif function == merge:
            df1 = Select(input + intermediates, context=c, id=6)
            df2 = Select(input + intermediates, context=c, id=7)
            common_cols = set(df1.columns) & set(df2.columns)
            arg_on = OrderedSubset(common_cols, context=[df1, df2, output], id=8)
            args = (df1, df2, arg_on)
        # Omitted code: case for each function

        intermediates.append(function.run(*args))

    return function_sequence
```

Pick a sequence of **functions**

Select function **arguments**

Combine functions

Previous work: AutoPandas

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

Operator	Description
Select(domain)	Returns a single item from domain
Subset(domain)	Returns an unordered subset, without replacement, of the items in domain
OrderedSubset(domain)	Returns an ordered subset, without replacement, of the items in domain
Sequence(len)(domain)	Returns an ordered sequence, with replacement, of the items in domain with a maximum length of len

Rank(Domain, Context) – per-operator ranking of selected functions/arguments using

Graph Neural Networks

Previous work: AutoPandas

Accuracy: **~65% (?)**

Benchmark	Depth	Candidates Explored		Sequences Explored		Solved		Time(s)	
		AutoPandas	Baseline	AutoPandas	Baseline	AutoPandas	Baseline	AutoPandas	Baseline
SO_11881165	1	**15**	64	1	1	Y	Y	**0.54**	1.46
SO_11941492	1	783	**441**	8	8	Y	Y	12.55	**2.38**
SO_13647222	1	**5**	15696	1	1	Y	Y	**3.32**	53.07
SO_18172851	1	-	-	-	-	N	N	-	-
SO_49583055	1	-	-	-	-	N	N	-	-
SO_49592930	1	**2**	4	1	1	Y	Y	**1.1**	1.43
SO_49572546	1	**3**	4	1	1	Y	Y	**1.1**	1.44
…									
SO_13576164	3	**22966**	-	5	-	**Y**	N	**339.25**	-
SO_14023037	3	-	-	-	-	N	N	-	-
SO_53762029	3	**27**	115	1	1	Y	Y	1.90	**1.50**
SO_21982987	3	8385	**8278**	10	10	Y	Y	30.80	**13.91**
SO_39656670	3	-	-	-	-	N	N	-	-
SO_23321300	3	-	-	-	-	N	N	-	-
Total						17/26	14/26		

Large Language Models (LLMs)

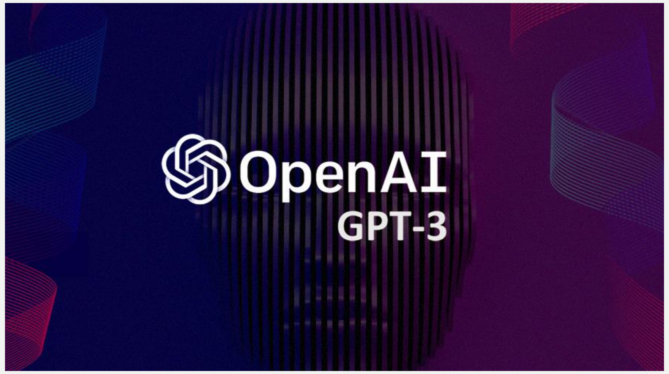
12 billion parameters

[1]

7 billion to 65 billion parameters

[2]

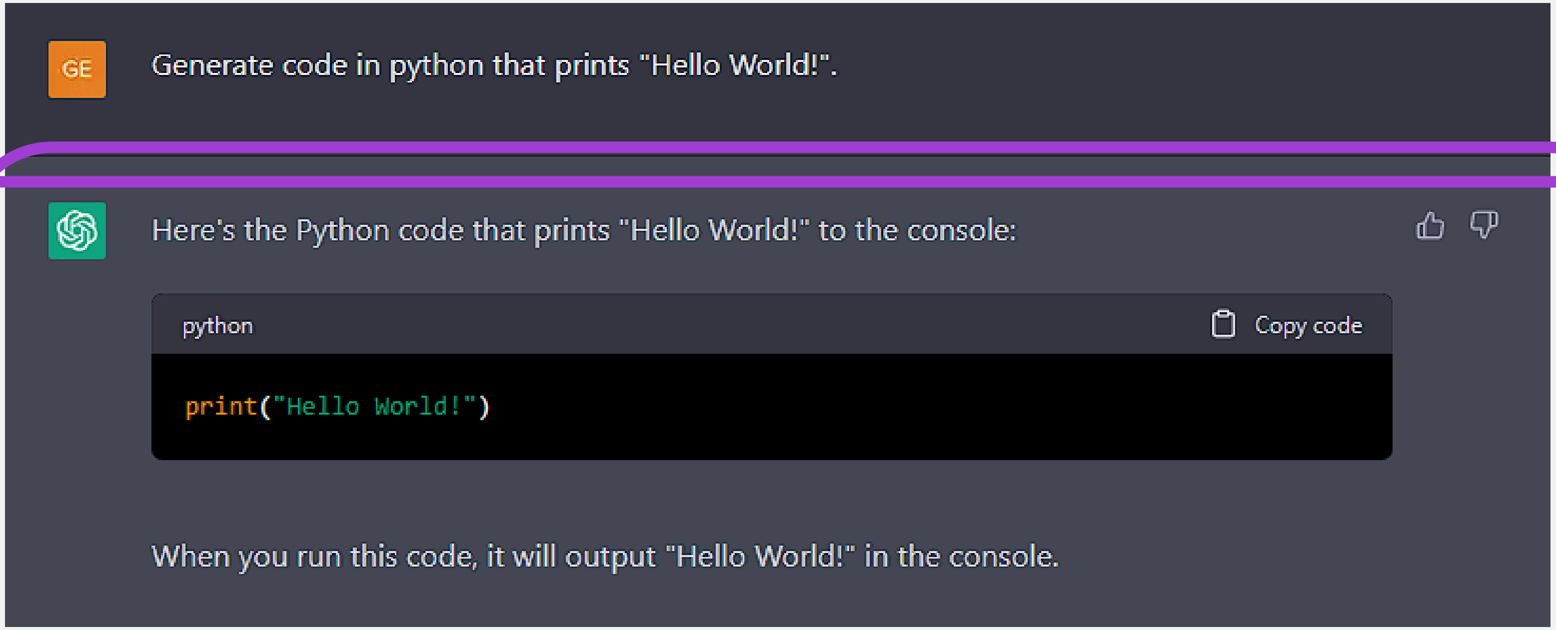
175 billion parameters



[3]

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*

Jigsaw Query

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

Input(s)

	A	B
0	84	foo
1	33	jig
2	22	bar
3	41	saw

Output

	A	B
1	33	jig
3	41	saw

Multimodal input: **query + I/O examples**

Solution 1 (status - PASSED)

```
1  dfout = df[df['A'] % 4 == 1]
```

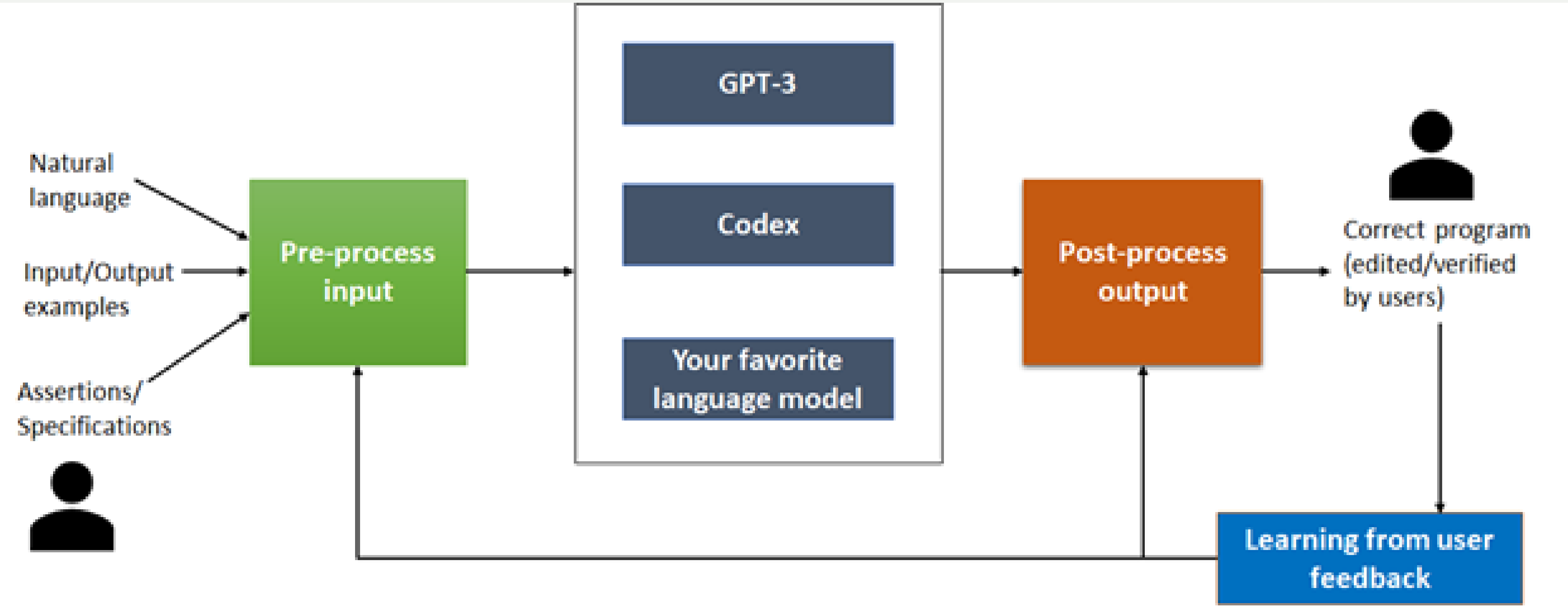
Solution 2 (status - DIDN'T PASS)

```
1  dfout = df.loc[df['A'] % 4 != 1]
```

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

```
gpt3 = GPT(engine="davinci", temperature=0.5, max_tokens=100)
# Examples to train a English to French translator
gpt3.add_example(Example('What is your name?', 'quel est votre nom?'))
gpt3.add_example(Example('What are you doing?', 'Que faites -vois?'))
gpt3.add_example(Example('How are you?', 'Comment allex-vous?'))
```

Prime the model with examples

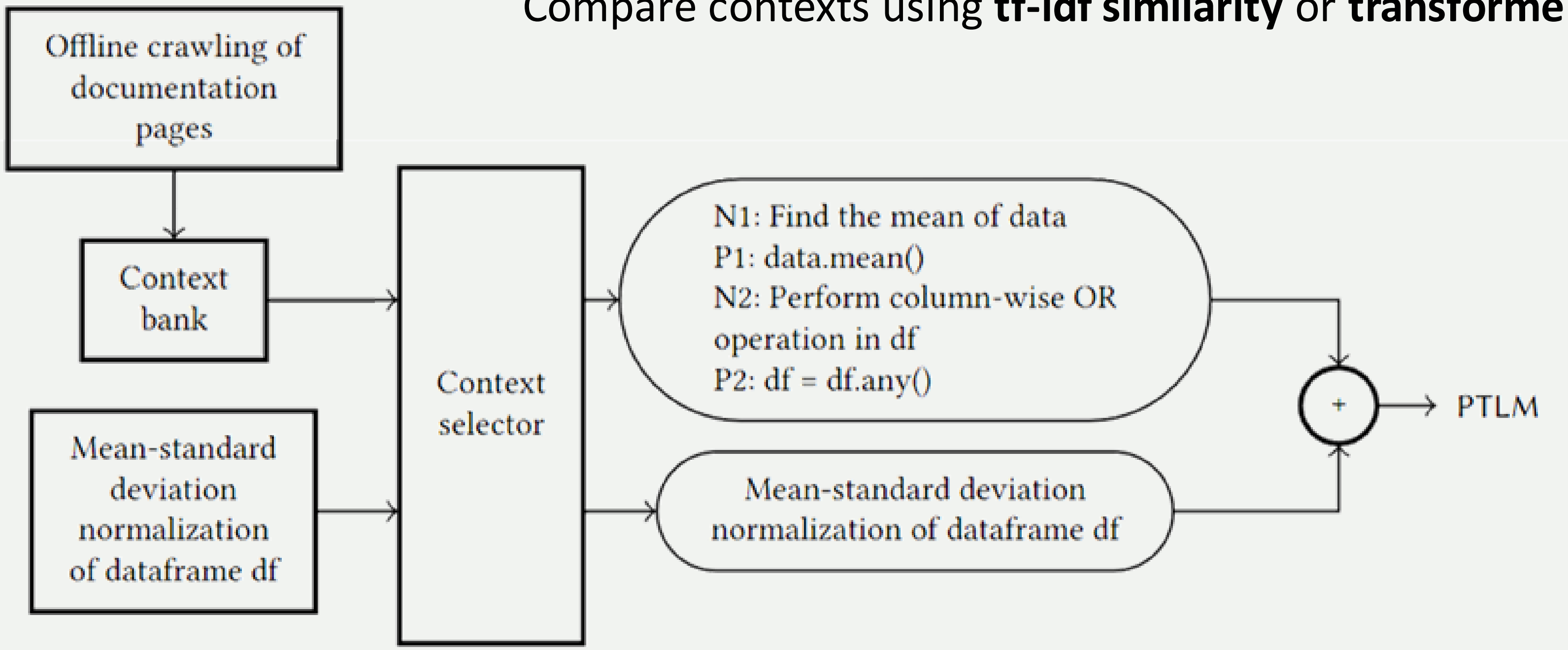
```
# Input to the model
prompt3 = "where are you?"
output3 = gpt3.submit_request(prompt3)
# Model output
output3.choices.text
```

Prompt the model

Output: Où êtes-vous?

Preprocessing

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



Post-processing

3 types of common errors

Reference errors

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

Argument errors

```
.cates(subset=['inputB']) # Model
cates(subset=['inputB'],keep=False) #
```

Semantic errors

```
duplicated() # Model
duplicated().sum() # Correct
```

Reference errors

Model output uses **incorrect variable names**

Developer uses **non-standard** variable names

E.g., **g1**, **g2** instead of **df1**, **df2** for DataFrames

Model **confuses** variable names

E.g., **df2**.merge(**df1**) instead of **df1**.merge(**df2**)

Variable transformations

Try **permutations and combinations** of variable names

df1.merge(**df1**)	✕
df1.merge(**df2**)	☑
df2.merge(**df1**)	✕
df2.merge(**df2**)	✕

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
```

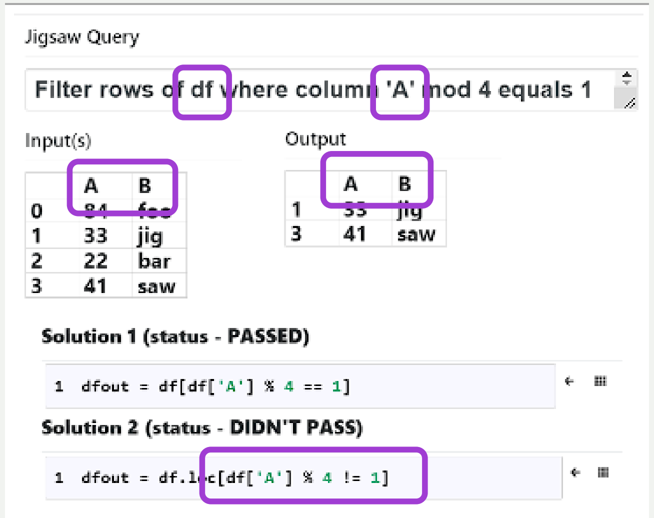
b.) Query – Replace Canada with `CAN` in column country of df

```
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`

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

Mistake – missing parentheses change precedence and cause exception

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

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

Mistake – missing required summation to get the count

Semantic errors

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

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

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

```
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

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

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:

country	city	IATA
USA	LA	France
France	Paris	Paris
UK	LON	DU
France	LYS	London
IN	DEL	IN

country	city	IATA
USA	LA	France
FR	PAR	Paris
UK	LON	DU
FR	LYS	London
IN	DEL	IN

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

		PandasEval1			PandasEval2		
		PTLM	Variable Name	Semantic Repair	PTLM	Variable Name	Semantic Repair
GPT-3	NO-CONTEXT	30.9 ± 1.2	38.2 ± 2.4	44.6 ± 3.9	8.9 ± 0.6	24.8 ± 0.9	33.6 ± 0.5
	TRANSFORMER	33.8 ± 2.4	41.7 ± 2.5	47.1 ± 2.1	6.6 ± 0.2	24.3 ± 0.8	35.1 ± 0.7
Codex	NO-CONTEXT	45.6 ± 1.2	54.9 ± 0.7	59.8 ± 3.5	26.8 ± 1.2	51.0 ± 0.6	56.8 ± 0.3
	TRANSFORMER	52.0 ± 0.7	63.7 ± 0.7	66.7 ± 0.7	31.2 ± 0.2	67.5 ± 0.5	72.2 ± 0.5

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**

	PandasEval2_S1	PandasEval2_S2	
	CS1-TS1	CS1-TS1	CS2-TS2
GPT-3	45.9 ± 0.4	35.1 ± 0.8	67.2 ± 0.3
Codex	75.1 ± 0.5	69.0 ± 0.7	84.4 ± 0.8

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

Input(s)

	A	B
~~0~~	~~84~~	~~foo~~
1	33	jig
2	22	bar
~~3~~	~~41~~	~~saw~~

Output

	A	B
1	33	jig
3	41	saw

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

	AutoPandas [9]	PTLM	Jigsaw
Subset of Jigsaw datasets	16/27	20/27	23/27
AutoPandas dataset	17/26	15/26	19/26

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**

	Context	PandasEval1	PandasEval2
GPT-3	TFIDF	46.5 ± 4.8	32.4 ± 0.5
	TRANSFORMER	47.1 ± 2.1	35.1 ± 0.7
Codex	TFIDF	69.1 ± 2.4	70.1 ± 0.1
	TRANSFORMER	66.7 ± 0.7	72.2 ± 0.5

Not sensitive to context selector

Ablation study

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

	# Prompts	PandasEval1	PandasEval2
GPT-3	1	47.5 ± 1.8	34.9 ± 0.9
	4	47.1 ± 2.1	35.1 ± 0.7
	8	48.0 ± 2.5	32.9 ± 0.6
Codex	1	62.3 ± 0.7	71.8 ± 0.5
	4	66.7 ± 0.7	72.2 ± 0.5
	8	66.2 ± 1.2	72.4 ± 0.9

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**

PTLM	Variable Name	Semantic Repair
8/25	15/25	19/25

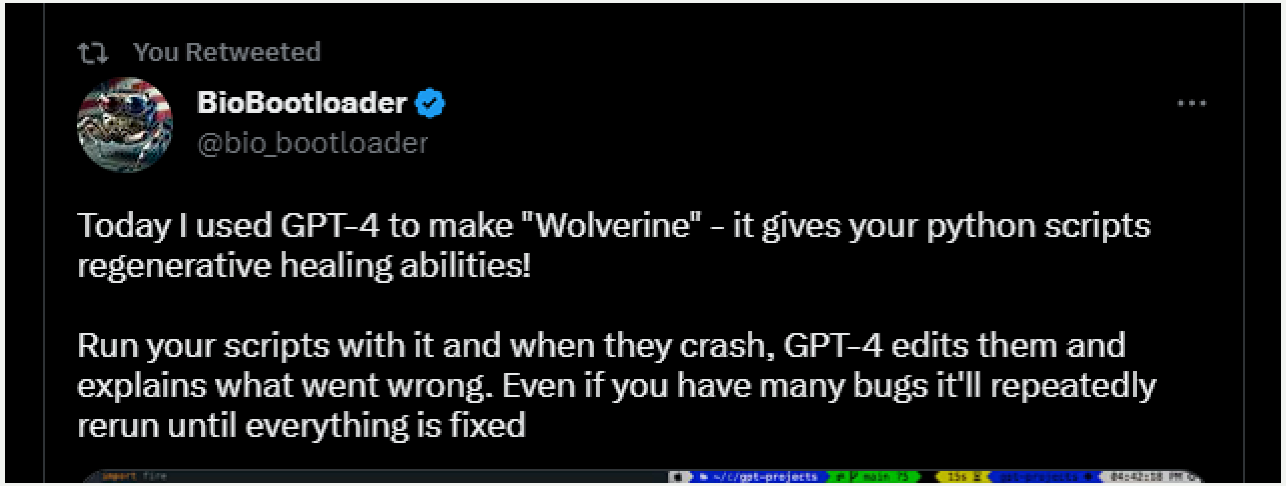
Evaluation and future work

- 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

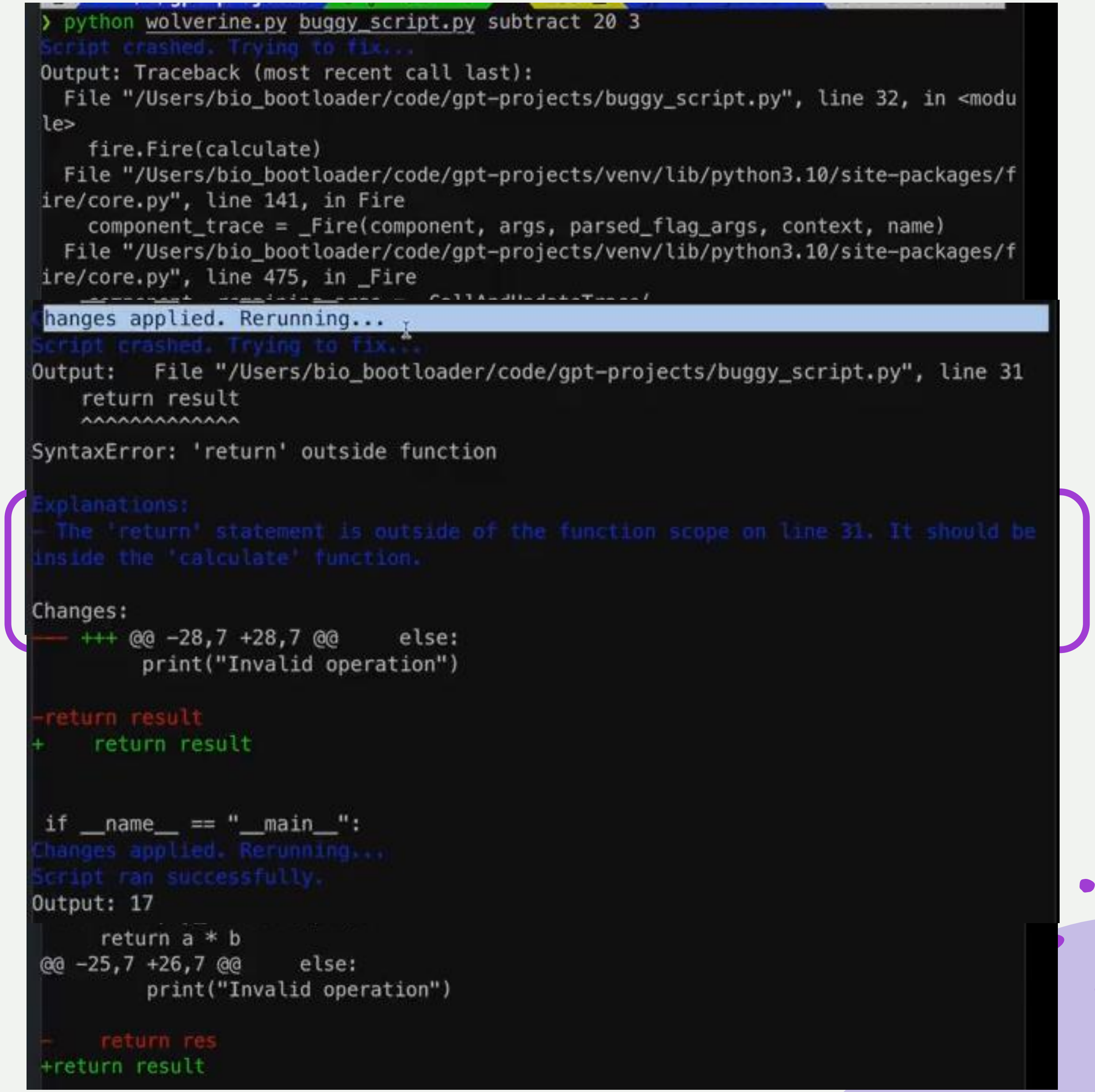
- In practice, code should have **high performance**, **be secure**, **respect licensing**
- Specifications can be **weak or ambiguous**, could be improved with pre-, postconditions, invariants, bounds, etc.

Why not use GPT to correct itself?



https://twitter.com/bio_bootloader/status/1636880208304431104


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


python wolverine.py buggy_script.py subtract 20 3
Script crashed. Trying to fix...
Output: 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 "/Users/bio_bootloader/code/gpt-projects/venv/lib/python3.10/site-packages/fire/core.py", line 475, in _Fire
hanges applied. Rerunning...
Script crashed. Trying to fix...
Output: File "/Users/bio_bootloader/code/gpt-projects/buggy_script.py", line 31
return result
SyntaxError: 'return' outside function
Explanations:
- The 'return' statement is outside of the function scope on line 31. It should be inside the 'calculate' function.
Changes:
--- +++ @@ -28,7 +28,7 @@ else:
print("Invalid operation")
-return result
+ return result
if __name__ == "__main__":
Changes applied. Rerunning...
Script ran successfully.
Output: 17
return a * b
@@ -25,7 +26,7 @@ else:
print("Invalid operation")
- return res
+return result

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!

GE

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?

https://chat.openai.com/chat

References

1. https://codeandhack.com/openai-codex-can-now-write-code-from-natural-language/
2. https://next14.com/en/nextnews-7-march-a-new-language-model-for-meta-bing-ai-on-windows-and-the-first-tokenized-real-estate-sales/
3. https://www.analyticsinsight.net/new-version-of-gpt-3-a-game-changing-language-model-by-open-ai/
4. Rohan Bavishi, Caroline Lemieux, Roy Fox, Koushik Sen, and Ion Stoica. 2019.AutoPandas: neural-backed generators for program synthesis.Proc. ACM Program.Lang.3, OOPSLA (2019), 168:1–168:27.

References

5. Vechev, Martin, and Eran Yahav. "Programming with “big code”." Foundations and Trends® in Programming Languages 3.4 (2016): 231-284.