

Contextual Cues for Deep Learning Models of Code

Disha Srivastava



Outline

On-the-Fly Adaptation of Source Code Models (*CAP Workshop, NeurIPS 2020*)

RepoFusion: Training Code Models to Understand Your Repository *(under review at NeurIPS 2023)*



<p>Learning to Combine Per-Example Solutions for Neural Program Synthesis</p> <hr/> <p>Hilke Hartmann Mitsuru Ito Liang Zhang</p> <p>Yann LeCun Facebook AI Research</p> <p>Shai Fine Mitsubishi Electric Research Research</p>
<p>Abstract</p> <p>The goal of program synthesis is to find a target program that satisfies a set of constraints. In this paper, we propose a novel learning-based approach to program synthesis. Our system, <i>Aggregator</i>, can learn to predict which blocks to invoke from a pool of pre-trained blocks to satisfy a set of constraints. By contrast, existing approaches that predict blocks to invoke must do so sequentially, one at a time, until all constraints are satisfied. We show that <i>Aggregator</i> can learn to invoke multiple blocks simultaneously, and thus significantly reduce the number of blocks required to satisfy a set of constraints. We also show that <i>Aggregator</i> can learn to invoke different blocks in parallel, and thus significantly reduce the number of steps required to satisfy a set of constraints. Finally, we show that <i>Aggregator</i> can learn to invoke different blocks in sequence, and thus significantly reduce the number of blocks required to satisfy a set of constraints.</p>
<p>1 Introduction</p> <p>Program synthesis is the problem of finding a program that satisfies a set of input requirements. There are two types of program synthesis: <i>symbolic</i> and <i>neural</i>. In symbolic synthesis, a user provides a set of requirements, and a solver finds a program that satisfies them. In neural synthesis, a user provides a set of requirements, and a neural network finds a program that satisfies them. In this paper, we propose a novel learning-based approach to program synthesis. Our system, <i>Aggregator</i>, can learn to predict which blocks to invoke from a pool of pre-trained blocks to satisfy a set of constraints. By contrast, existing approaches that predict blocks to invoke must do so sequentially, one at a time, until all constraints are satisfied. We show that <i>Aggregator</i> can learn to invoke multiple blocks simultaneously, and thus significantly reduce the number of blocks required to satisfy a set of constraints. We also show that <i>Aggregator</i> can learn to invoke different blocks in parallel, and thus significantly reduce the number of steps required to satisfy a set of constraints. Finally, we show that <i>Aggregator</i> can learn to invoke different blocks in sequence, and thus significantly reduce the number of blocks required to satisfy a set of constraints.</p>
<p>https://arxiv.org/abs/1905.07052</p>

Learning to Combine Per-Example Solutions for Neural Program Synthesis (NeurIPS 2021)

Repository-Level Prompt Generation for Large Language Models of Code

Dimitris Sideris^{1,2}, Yigitcan Hascal^{1,2,3,4}, Dafna Tsurk^{1,5}

¹University of Michigan, ²Microsoft Research, ³Microsoft, ⁴University of Michigan, ⁵University of Illinois Urbana-Champaign

Abstract

With the rise of large language models (LLMs), there has been a significant interest in generating code snippets using LLMs. One challenge in this task is how to generate code that is both correct and useful. In this work, we propose a repository-level prompt generation approach that is capable of generating snippets that are both correct and useful. We evaluate our approach against state-of-the-art baselines and show that it outperforms them in terms of both correctness and usefulness. We also show that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines. Finally, we show that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines. We believe that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines.

1. Introduction

Large language models (LLMs) have become increasingly popular as a source of knowledge for generating code snippets. In this paper, we propose a repository-level prompt generation approach that is capable of generating code snippets that are both correct and useful. In this work, we propose a repository-level prompt generation approach that is capable of generating code snippets that are both correct and useful. We evaluate our approach against state-of-the-art baselines and show that it outperforms them in terms of both correctness and usefulness. We also show that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines. Finally, we show that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines. We believe that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines.

2. Related Work

There has been a significant amount of research on generating code snippets using LLMs. One challenge in this task is how to generate code that is both correct and useful. In this work, we propose a repository-level prompt generation approach that is capable of generating snippets that are both correct and useful. We evaluate our approach against state-of-the-art baselines and show that it outperforms them in terms of both correctness and usefulness. We also show that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines. Finally, we show that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines. We believe that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines.

3. Methodology

In this section, we describe the proposed methodology for generating code snippets using LLMs. The methodology consists of three main steps: (1) collecting a dataset of code snippets, (2) generating a repository-level prompt, and (3) generating the final code snippet.

3.1. Dataset Collection

We collected a dataset of code snippets from GitHub. The dataset contains over 1 million code snippets, each with a corresponding commit message. The commits are from various repositories, including open-source projects and commercial software. The dataset is annotated with labels indicating whether the code is correct or incorrect, and whether it is useful or not useful.

3.2. Repository-Level Prompt Generation

The repository-level prompt generation approach consists of two main components: (1) a template for generating prompts, and (2) a repository-level prompt generator.

3.2.1. Template

The template for generating prompts is defined as follows:

```
template = "Given the following code snippet, answer the question. If the code is correct, return 'True'. If the code is incorrect, return 'False'. If the code is correct but not useful, return 'None'. If the code is useful but not correct, return 'None'. If the code is neither correct nor useful, return 'None'. The code snippet is: {code_snippet}. The question is: {question}."
```

3.2.2. Repository-Level Prompt Generator

The repository-level prompt generator takes a repository as input and generates a repository-level prompt. The repository-level prompt is generated by concatenating the template with the repository's name and a list of its files. The repository-level prompt is then passed to the LLM for generation.

3.3. Code Generation

The LLM generates the final code snippet based on the repository-level prompt. The generated code snippet is then evaluated for correctness and usefulness. If the code is correct and useful, it is returned. If the code is incorrect or not useful, it is discarded.

4. Evaluation

We evaluated our approach against state-of-the-art baselines. The baselines include a template-based approach and a repository-level prompt generation approach. The evaluation results show that our approach outperforms the baselines in terms of both correctness and usefulness.

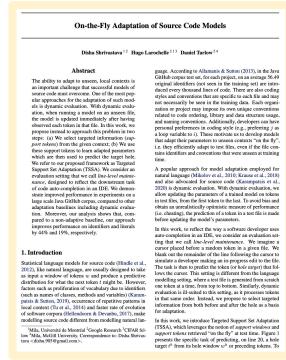
5. Conclusion

In this paper, we proposed a repository-level prompt generation approach for generating code snippets using LLMs. The approach is capable of generating snippets that are both correct and useful. We evaluated our approach against state-of-the-art baselines and showed that it outperforms them in terms of both correctness and usefulness. We believe that our approach can be used to generate snippets that are more useful than those generated by state-of-the-art baselines.

Repository Level Prompt Generation for Large Language Models of Code (ICML 2023)

Outline

On-the-Fly Adaptation of Source Code Models (CAP Workshop, NeurIPS 2020)



RepoFusion: Training Code Models to Understand Your Repository (under review at NeurIPS 2023)



Introduction

Article 1



Article 2



Article 3

Final Thoughts

Learning to Combine Per-Example Solutions for Neural Program Synthesis
(NeurIPS 2021)

Repository Level Prompt Generation for Large Language Models of Code
(ICML 2023)

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.

```
# Given a string representing musical
notes in a special ASCII format,
# parse this string and compute a total
length of the music in beats.
# Legend:
#   'o' - whole note, Last four beats
#   'o|' - half note, Last two beats
#   '.|' - quarter note, Last one beat

notes = 'o o| .| o| o| .| .| .| o o'

# 1. Split the string into a list of
# notes.
# 2. Iterate over the list and compute the
# total length of the music.
# 3. Print the result.

# 1.
notes = notes.split()

# 2.
total_length = 0
for note in notes:
    if note == 'o':
        total_length += 4
    if note == 'o|':
        total_length += 2
    if note == '.|':
        total_length += 1

# 3.
print(total_length)
```

Code Generation

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.

```
# Given a string representing musical
notes in a special ASCII format,
# parse this string and compute a total
length of the music in beats.
# Legend:
#   'o' - whole note, Last four beats
#   'o|' - half note, Last two beats
#   '.' - quarter note, Last one beat

notes = 'o o| .| o| o| .| .| .| .| o o'
```

Code Generation

```
# 1. Split the string into a list of
notes.
# 2. Iterate over the list and compute the
total length of the music.
# 3. Print the result.

# 1.
notes = notes.split()

# 2.
total_length = 0
For note in notes:
    if note == 'o':
        total_length += 4
    if note == 'o|':
        total_length += 2
    if note == '.':
        total_length += 1

# 3.
print(total_length)
```

```
1 #include <stdio.h>
2 int main() {
3     int a[10];
4     int i;
5     scanf("%d", &n);
6     scanf("%d", &a[1]);
7     int count = 0;
8     for (i = 0; i < n; i++) {
9         for (j = 0; j < n; j++) {
10             if (a[i] == a[j])
11                 count = count + 1;
12         }
13     }
14     if (count == a[i])
15         printf("Yes");
16     else
17         printf("No");
18 }
```

```
1 #include <stdio.h>
2 int main() {
3     int a[100], i, n, count = 0;
4     scanf("%d", &n);
5     for (i = 0; i < n; i++)
6         scanf("%d", &a[i]);
7     for (i = 0; i < n; i++) {
8         for (int j = 0; j < n; j++) {
9             if (a[i] == a[j])
10                 count = count + 1;
11         }
12     }
13     if (count == n)
14         printf("Yes");
15     else
16         printf("No");
17 }
```

Code Repair

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.

```
# Given a string representing musical
notes in a special ASCII format,
# parse this string and compute a total
length of the music in beats.
# Legend:
#   'o' - whole note, Last four beats
#   'o|' - half note, Last two beats
#   '.' - quarter note, Last one beat

notes = 'o o| .| o| o| .| .| .| .| o o'
```

Code Generation

```
# 1. Split the string into a list of
notes.
# 2. Iterate over the list and compute the
total length of the music.
# 3. Print the result.

# 1.
notes = notes.split()

# 2.
total_length = 0
for note in notes:
    if note == 'o':
        total_length += 4
    if note == 'o|':
        total_length += 2
    if note == '.':
        total_length += 1

# 3.
print(total_length)
```

```
1 #include <stdio.h>
2 int main() {
3     int a[10];
4     int i;
5     scanf("%d", &n);
6     scanf("%d", &a[1]);
7     int count = 0;
8     for (i = 0; i < n; i++) {
9         for (j = 0; j < n; j++) {
10             if (a[i] == a[j])
11                 count = count + 1;
12         }
13     }
14     if (count == a[i])
15         printf("Yes");
16     else
17         printf("No");
18     return 0;
19 }
```

```
1 #include <stdio.h>
2 int main() {
3     int a[100], i, n, count = 0;
4     scanf("%d", &n);
5     for (i = 0; i < n; i++)
6         scanf("%d", &a[i]);
7     for (i = 0; i < n; i++) {
8         for (int j = 0; j < n; j++) {
9             if (a[i] == a[j])
10                 count = count + 1;
11         }
12     }
13     if (count == n)
14         printf("Yes");
15     else
16         printf("No");
17     return 0;
18 }
```

Code Repair

```
// Translate from C to Python
int add_one ( int x ) {
    int m = 1;
    while ( x & m ) {
        x = x ^ m;
        m <<= 1;
    }
    x = x ^ m;
    return x; }
```

```
def add_one(x: int):
    m = 1
    while (x & m):
        x = (x ^ m)
        m <= 1
    x = (x ^ m)
    return x
```

Code Translation

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.

```
# Given a string representing musical notes in a special ASCII format, # parse this string and compute a total length of the music in beats. # Legend: #   'o' - whole note, Last four beats #   'o|' - half note, Last two beats #   '.' - quarter note, Last one beat

notes = 'o o| .| o| o| .| .| .| .| o o'
```

Code Generation

```
# 1. Split the string into a list of notes.
# 2. Iterate over the list and compute the total length of the music.
# 3. Print the result.

# 1.
notes = notes.split()

# 2.
total_length = 0
For note in notes:
    if note == 'o':
        total_length += 4
    if note == 'o|':
        total_length += 2
    if note == '.':
        total_length += 1

# 3.
print(total_length)
```

Slow Version(Runtime=4.18s)

```
t = int(input())
for i in range(1, t+1):
    n, b = [int(s) for s in input().split(" ")]
    houses = [int(s) for s in input().split(" ")]
    houses.sort()

    result = 0
    for h in houses:
        if b >= h:
            result += 1
            b -= h
    print("Case #{}: {}".format(i, result))
```

Algorithmic Difference: Using a heap instead of pre-sorting the list once allows early termination of the main loop.

Fast Version(Runtime=2.52s: ~1.66x Speedup)

Fast Version(Runtime=2.52s: ~1.66x Speedup)

```
import heapq

# input() reads a string with a line of input, stripping the '\n'(newline) at the end.
# This is all you need for most Kickstart problems.
# All data
t = int(input()) # read a line with a single integer
for i in range(1, t+1):
    n, b = [int(s) for s in input().split(" ")] # read a list of integers, 2 in this case
    prices = [int(s) for s in input().split(" ")]
    all_data.append([n, b, prices])
    heapq.heapify(prices)
    houses = []
    while prices and b > 0:
        new_house = heapq.heappop(prices)
        b -= new_house
        if b >= 0:
            houses += 1
    print("Case #{}: {}".format(i, houses))
```

Code Analysis

"Learning to Improve Code Efficiency". Chen, Binhong, et al. (2022)

```
1 #include <stdio.h>
2 int main() {
3     int a[10];
4     int i;
5     scanf("%d", &n);
6     scanf("%d", &a[1]);
7     int count = 0;
8     for (i = 0; i < n; i++) {
9         for (j = 0; j < n; j++) {
10             if (a[i] == a[j]) {
11                 count = count + 1;
12             }
13         }
14         if (count == a[i])
15             printf("Yes");
16         else
17             printf("No");
18     }
19 }
```

```
1 #include <stdio.h>
2 int main() {
3     int a[100], i, n, count = 0;
4     scanf("%d", &n);
5     for (i = 0; i < n; i++)
6         scanf("%d", &a[i]);
7     for (i = 0; i < n; i++) {
8         for (int j = 0; j < n; j++) {
9             if (a[i] == a[j])
10                 count = count + 1;
11         }
12     }
13     if (count == n)
14         printf("Yes");
15     else
16         printf("No");
17     return 0;
18 }
```

Code Repair

```
// Translate from C to Python
int add_one ( int x ) {
    int m = 1;
    while ( x & m ) {
        x = x ^ m;
        m <<= 1;
    }
    x = x ^ m;
    return x;
}
```

```
def add_one(x: int):
    m = 1
    while (x & m):
        x = (x ^ m)
        m <= 1
    x = (x ^ m)
    return x
```

Code Translation

<https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html>

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.

```
# Given a string representing musical notes in a special ASCII format, # parse this string and compute a total length of the music in beats. # Legend: #   'o' - whole note, last four beats #   'o/' - half note, last two beats #   './' - quarter note, last one beat notes = 'o o| .| o| o| .| .| .| o|
```

Code Generation

Slow Version(Runtime=4.18s)

```
t = int(input())
for i in range(1, t+1):
    n, b = [int(s) for s in input().split(" ")]
    houses = [int(s) for s in input().split(" ")]
    houses.sort()

    result = 0
    for h in houses:
        if b >= h:
            result += 1
        b -= h
    print("Case #{}: {}".format(i, result))
```

Algorithmic Difference: Instead of pre-sorting the list once allows early termination of the main loop.

```
n, b = [int(s) for s in input().split(" ")] # read a list of integers, 2 in this case
prices = [int(s) for s in input().split(" ")]
wall_data.append([n,b,prices])
heapq.heapify(prices)
houses = []
while prices and b > 0:
    new_house = heapq.heappop(prices)
    b -= new_house
    if b >= 0:
        houses += 1
print("Case #{}: {}".format(i, houses))
```

Code Analysis

```
# 1. Split the string into a list of notes.
# 2. Iterate over the list and compute the total length of the music.
# 3. Print the result.
```

```
import tensorflow as tf

@wrap_objective
def l2(batch=None):
    def inner(T):
        return -tf.reduce_mean((T("input") - 0.5)**2)
    return inner
```

Show Explanation

- The code is a function that takes in an input tensor and returns the squared difference between it and 0.5.
- The code is written to take in a batch size, which defaults to None.
- The code is used to calculate the L2 norm of a batch of data.
- This code calculates the L2 norm for each sample in a batch, and then returns the average value across all samples.

Code Explanation

```
1 #include <stdio.h>
2 int main() {
3     int a[10];
4     int i;
5     scanf("%d", &n);
6     scanf("%d", &a[i]);
```

```
1 #include <stdio.h>
2 int main() {
3     int a[10], i, n, count = 0;
4     scanf("%d", &n);
5     for (i = 0; i < n; i++) {
6         scanf("%d", &a[i]);
7         for (i = 0; i < n; i++) {
8             for (int j = 0; j < n; j++) {
9                 if (a[i] == a[j]) {
10                     count = count + 1;
11                 }
12             }
13         }
14         if (count == n)
15             printf("Yes");
16         else
17             printf("No");
18     }
```

Code Repair

Python

```
def add_one(x: int):
    m = 1
    while (x & m):
        x = (x ^ m)
        m <= 1
    x = (x ^ m)
    return x
```

```
while (x & m) {
    x = x ^ m;
    m <= 1;
}
x = x ^ m;
return x;
```

Code Translation

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.
- Motivation

Helping Non-Programmers

Enable non-expert users to solve problems in an automated fashion.

- Programming requires technical skills.
- Generate programs from user's intent expressed in forms that are natural to them such as NL.

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.
- Motivation

Helping Non-Programmers

Enable non-expert users to solve problems in an automated fashion.

- Programming requires technical skills.
- Generate programs from user's intent expressed in forms that are natural to them such as NL.

Helping Programmers

Boost productivity of software developers.

- Divert attention from mundane tasks.
- Focus on tasks that require creative thinking.
- Code completion to avoid typing boilerplate code.

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.
- Motivation

Helping Non-Programmers

Enable non-expert users to solve problems in an automated fashion.

- Programming requires technical skills.
- Generate programs from user's intent expressed in forms that are natural to them such as NL.

Helping Programmers

Boost productivity of software developers.

- Divert attention from mundane tasks.
- Focus on tasks that require creative thinking.
- Code completion to avoid typing boilerplate code.

Advancing ML Research

Several challenges with modeling source code.

- Rigid Syntax
- Structure
- Discrete nature
- Multiple symbolic representation forms
- Continuously evolving

Deep Learning for Code

- Source Code: Exciting application domain for deep learning methods.
- Motivation
- LLMs of code have potential for huge impact.

```
1 import datetime
2
3 def parse_expenses(expenses_string):
4     """Parse the list of expenses and return the list of triples (date, value, currency).
5     Ignore lines starting with #.
6     Parse the date using datetime.
7     Example expenses_string:
8         2016-01-02 -34.01 USD
9         2016-01-03 2.39 DKK
10        2016-01-03 -2.72 EUR
11    """
12    expenses = []
13    for line in expenses_string.splitlines():
14        if line.startswith("#"):
15            continue
16        date, value, currency = line.split(" ")
17        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
18                         float(value),
19                         currency))
20
21    return expenses
```



GitHub
Copilot

The screenshot shows a user interface for GitHub Copilot. At the top, there is a list of three numbered steps:

1. Check that the file 'data.csv' actually exists. You can do this by using the 'os.path.isfile()' function.
2. Check that the file 'data.csv' is in the same directory as your Python script. You can do this by using the 'os.getcwd()' function to get the current working directory, and then comparing it to the directory where the file 'data.csv' is located.
3. Check that the file 'data.csv' has the correct name. You can do this by using the 'os.path.basename()' function to get the base name of the file, and then comparing it to the name you are expecting.

Below the steps, there is a note: "If you have checked all of these things and the error is still occurring, you may need to contact the person who created the file 'data.csv' for help."

At the bottom of the interface, there are several interaction buttons: thumbs up, thumbs down, a refresh icon, and a "Google it" button. There is also a search bar with the placeholder "Enter a prompt here".

The screenshot shows a dark-themed code editor window for Tabnine Pro. The code in the editor is:

```
1 module.exports = function allowedAdminAccessMiddleware(req, res, next) {
2     if (req.user.isAdmin) {
3         return next();
4     } else {
5         return res.status(403).json({
6             message: 'Unauthorized access'
7         });
8     }
9 }
10
```

A purple sidebar on the right side of the editor displays the text "Tabnine Pro".

The Tabnine logo consists of a purple hexagonal icon followed by the word "tabnine" in lowercase.

Already part of consumer-facing products

Central Theme

Effectively Harness Contextual Cues

**Identify and select
relevant contextual
cues from a given task.**



**Leverage these contextual
cues **effectively** in deep
learning models of code.**

Central Theme

Effectively Harness Contextual Cues

Identify and select relevant contextual cues from a given task.



Leverage these contextual cues effectively in deep learning models of code.

Improves Generalization

- Adding information that the model wouldn't normally have access to.
- Directing model's attention to specific information.

More Context-Aware Predictions

- Adapt to unseen tasks
- Improve performance on existing tasks.

Our General Framework

Given

X = Input Context (*code in the current file before the cursor*)

Y = Actual Target (*tokens following the cursor till the end of the line*)

W = Context Meta-information (*content in other files in the repository*)

Goal: Effectively harness contextual cues based on **X** and **W** such that the predicted target \hat{Y} is close to the actual target **Y**.

Our General Framework

Given

X = Input Context (*code in the current file before the cursor*)

Y = Actual Target (*tokens following the cursor till the end of the line*)

W = Context Meta-information (*content in other files in the repository*)

Goal: Effectively harness contextual cues based on **X** and **W** such that the predicted target \hat{Y} is close to the actual target **Y**.

Context Enhancement

$$Z = Enhance (X , W)$$

Support Context

(*method names and bodies
from the imported file*)

Our General Framework

Given

X = Input Context (*code in the current file before the cursor*)

Y = Actual Target (*tokens following the cursor till the end of the line*)

W = Context Meta-information (*content in other files in the repository*)

Goal: Effectively harness contextual cues based on **X** and **W** such that the predicted target \hat{Y} is close to the actual target **Y**.

Context Enhancement

$$Z = \text{Enhance} (X , W)$$

Prediction using the Enhanced Context

$$\hat{Y} = \text{Predict} (X , Z)$$

Support Context

(*method names and bodies from the imported file*)

Predicted Target
(*tokens generated by the model*)

Our General Framework

Given

X = Input Context (*code in the current file before the cursor*)

Y = Actual Target (*tokens following the cursor till the end of the line*)

W = Context Meta-information (*content in other files in the repository*)

Goal: Effectively harness contextual cues based on X and W such that the predicted target \hat{Y} is close to the actual target Y .

Context Enhancement

$$Z = \text{Enhance} (X , W)$$

Prediction using the Enhanced Context

$$\hat{Y} = \text{Predict} (X , Z)$$

Support Context
(*method names and bodies from the imported file*)

Without Context Enhancement

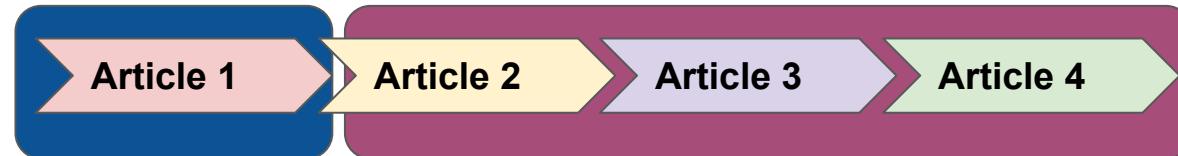
$$\hat{Y} = Q (X)$$

Predicted Target
(*tokens generated by the model*)

Thesis Overview

All articles in this thesis are based on our general
Enhance-Predict framework.

- We propose novel approaches for Enhance and Predict stages.
- We focus on two main tasks.



Program Synthesis
by Examples

Code Completion in an IDE

Outline

On-the-Fly Adaptation of Source Code Models (CAP Workshop, NeurIPS 2020)



^{1,2}McGill University, Quebec, Canada. Correspondence to: Dhruv Shrivastava (dhruv.shrivastava@mcgill.ca).

RepoFusion: Training Code Models to Understand Your Repository (under review at NeurIPS 2023)



Introduction

Article 1



Learning to Combine Per-Example Solutions for Neural Program Synthesis (NeurIPS 2021)

Article 2



Repository Level Prompt Generation for Large Language Models of Code (ICML 2023)

Article 3

Final Thoughts

Learning to Combine Per-Example Solutions for Neural Program Synthesis

NeurIPS 2021



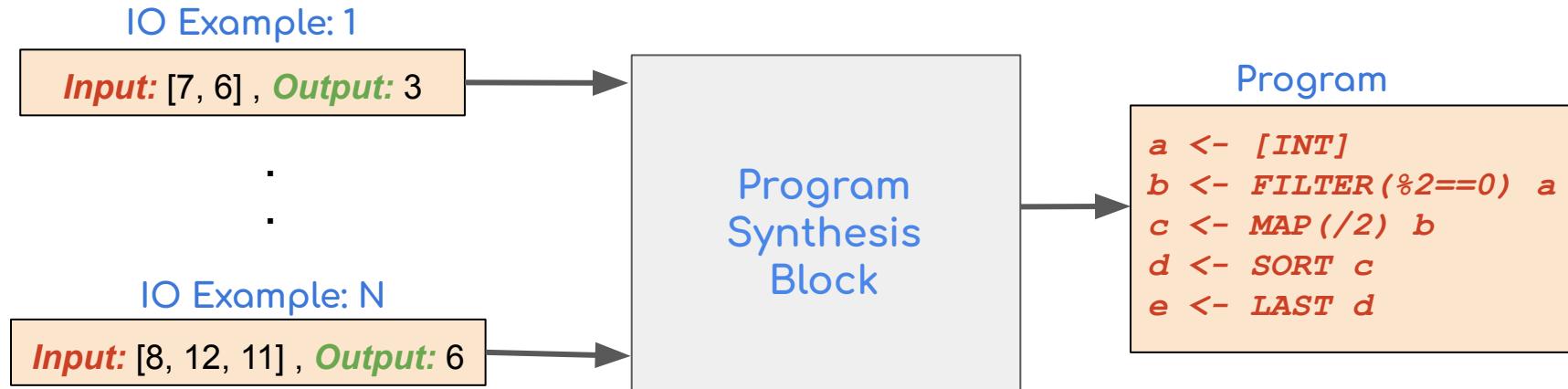
Université 
de Montréal

 Mila

Google Research

Code, data and trained checkpoints: <https://github.com/shrivastavadisha/N-PEPS>

Task: Program Synthesis by Examples



- Given a set of N IO examples, find a program that satisfies those examples.

Task: Program Synthesis by Examples

DSL = governs the syntax and semantics of the program

```
SORT :: [INT] -> [INT],  
MAP :: (INT -> INT) -> [INT] -> [INT], ...  
..
```

IO Example: 1

Input: [7, 6] , **Output:** 3

.

.

IO Example: N

Input: [8, 12, 11] , **Output:** 6

Program
Synthesis
Block

Program

```
a <- [INT]  
b <- FILTER(%2==0) a  
c <- MAP(/2) b  
d <- SORT c  
e <- LAST d
```

- Given a set of N IO examples, find a program that satisfies those examples.
- Given a **timeout** value to be practically meaningful.

Neural Per-Example Program Synthesis (N-PEPS)

```
#1: [154, -252, -228, -85, -136], [109, 65, -3, 71, 189] -> []
#2: [-113, 240, -59, 66], [-197, 150] -> [-240, -66]
#3: [-7, 106, -138], [225, 97, 17] -> []
#4: [-140, -51, 155, 74, -21], [35, 82, -103] -> [-155, -74]
#5: [87, -115, 52], [177, 193, -17] -> [-52]
```

p_g

```
a <- LIST
b <- LIST
1: c <- COUNT (>0) b
2: d <- DROP c a
3: e <- MAP (*-1) d
4: f <- FILTER (<0) e
```

Global Program Synthesis (GPS)

- Find **global solution** p_g that satisfies all IO examples *simultaneously*
- Can be hard

Neural Per-Example Program Synthesis (N-PEPS)

#1: [154, -252, -228, -85, -136], [109, 65, -3, 71, 189] -> []
#2: [-113, 240, -59, 66], [-197, 150] -> [-240, -66]
#3: [-7, 106, -138], [225, 97, 17] -> []
#4: [-140, -51, 155, 74, -21], [35, 82, -103] -> [155, -74]
#5: [87, -115, 52], [177, 193]

p_g

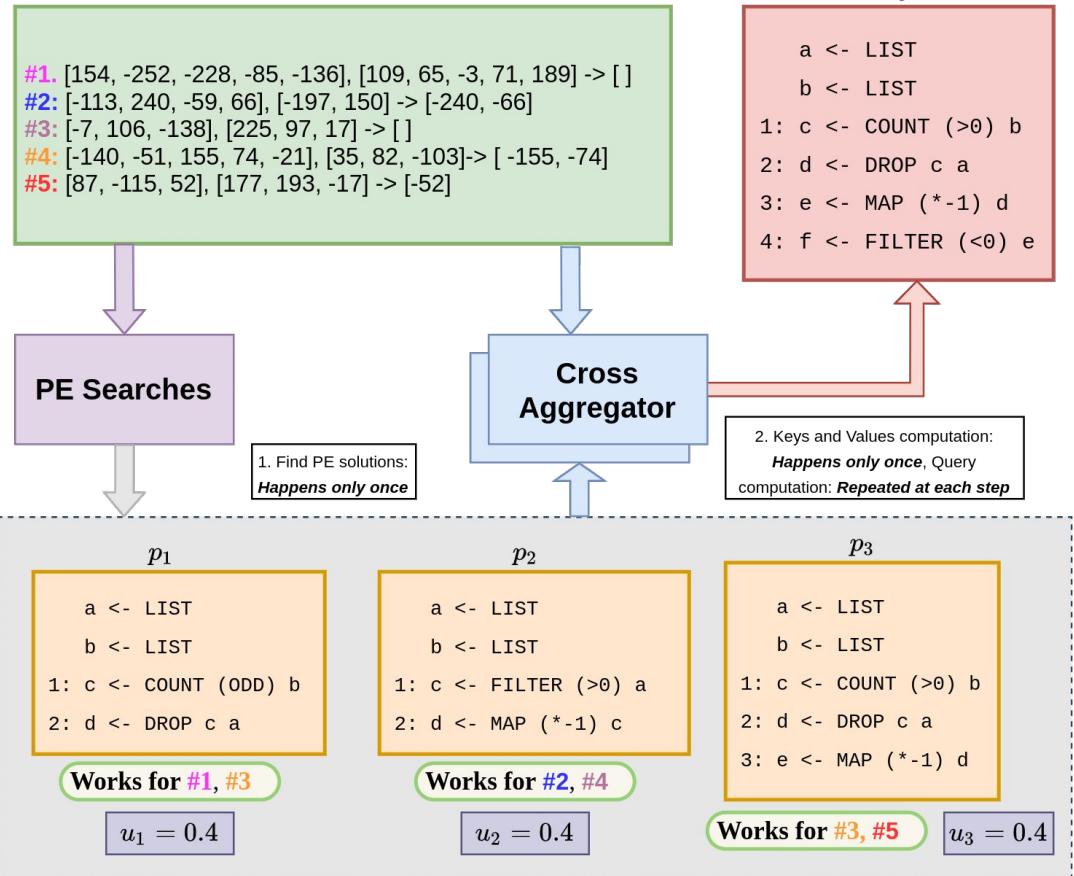
```
a <- LIST  
b <- LIST  
1: c <- COUNT (>0) b
```

Global Program Synthesis (GPS)

- Find global solution p_g that satisfies all IO examples *simultaneously*

- Break a hard problem into smaller, easy to solve subproblems
- Learn to combine the solutions of the sub-problems such that the harder problem is solved

Neural Per-Example Program Synthesis (N-PEPS)



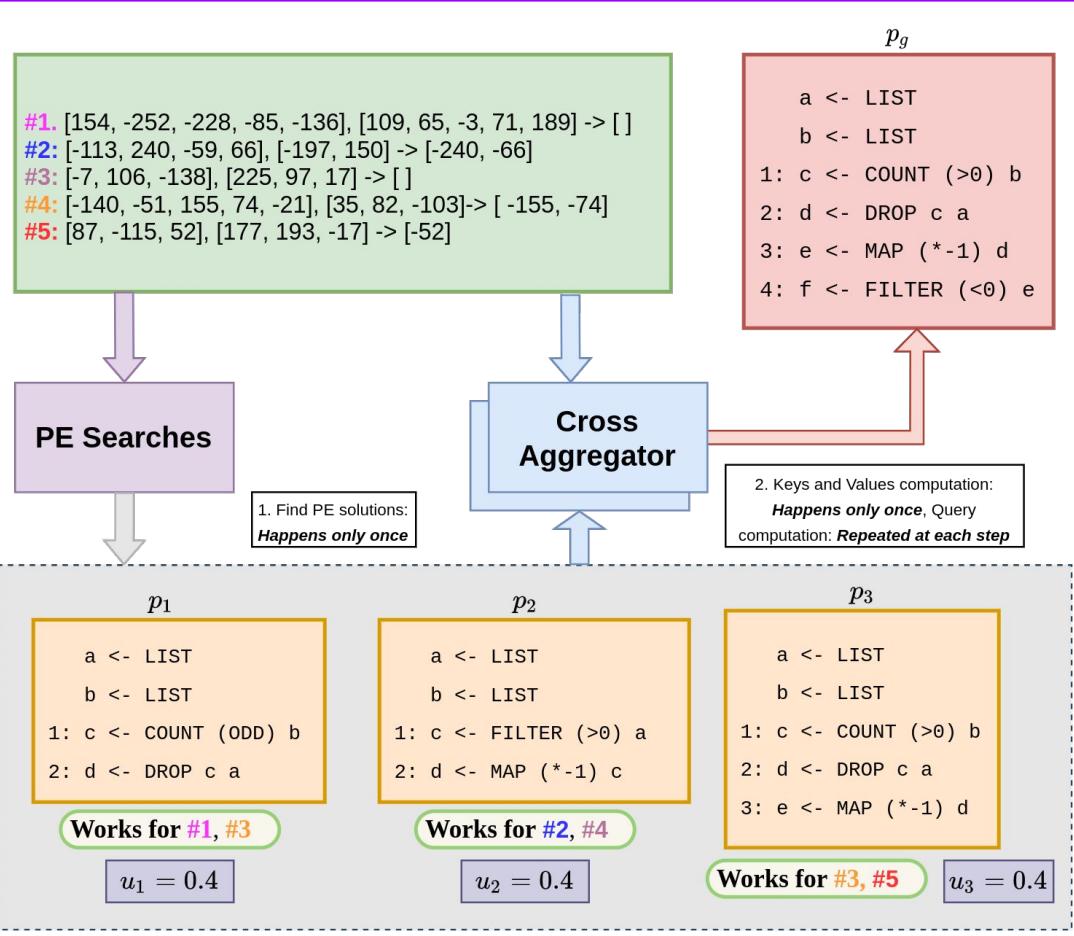
Global Program Synthesis (GPS)

- Find global solution p_g that satisfies all IO examples *simultaneously*
- Can be hard

Per Example Program Synthesis (PEPS). Break into two stages:

- **Enhance:** Find programs that satisfy a single example (PE solutions) - fast
- **Predict:** Combine the PE solutions such that it leads to the global solution

Neural Per-Example Program Synthesis (N-PEPS)



Global Program Synthesis (GPS)

- Find global solution p_g that satisfies all IO examples *simultaneously*
- Can be hard

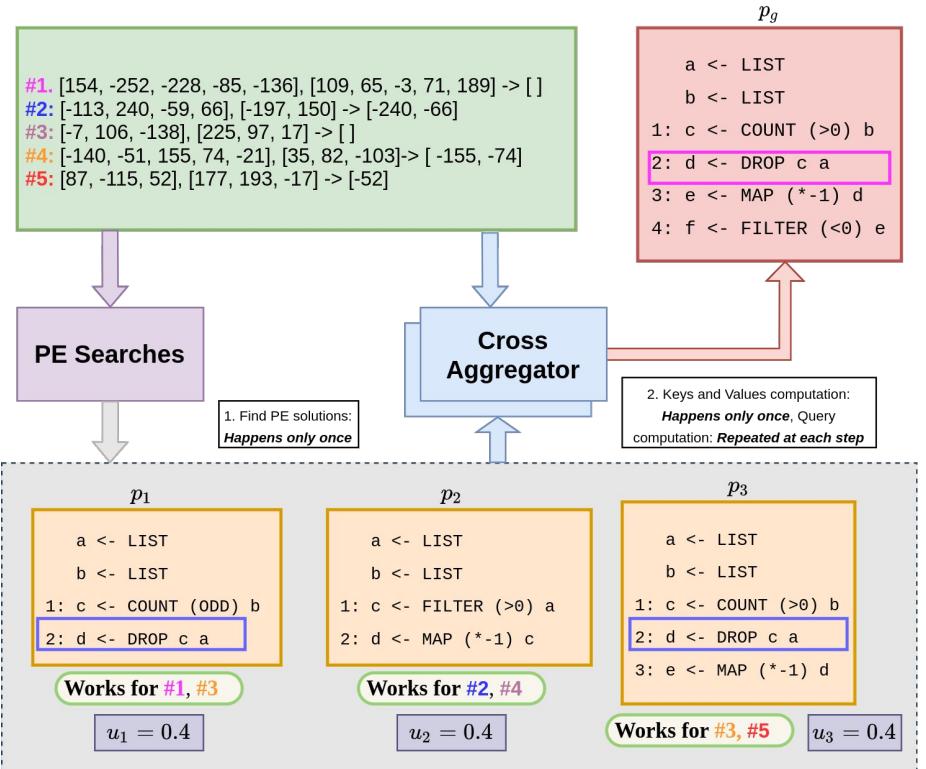
Per Example Program Synthesis (PEPS): Break into two stages:

- **Enhance:** Find programs that satisfy a single example (**PE solutions**) - fast
- **Predict:** Combine the PE solutions such that it leads to the global solution
- We propose an architecture called **Cross Aggregator (CA)** that *learns* to combine the PE solutions.

We use neural networks for both these stages (PE Searches and CA): **N-PEPS**

Cross Aggregator (CA)

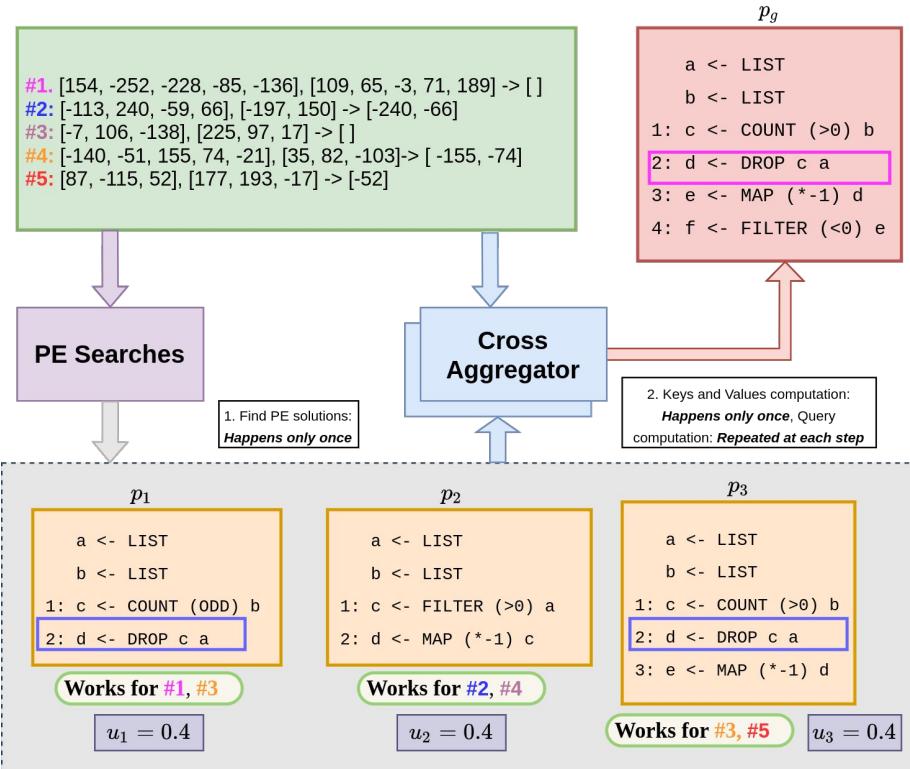
Idea: If a PE program state* has high relevance with the global program state at a given step, then the following PE program line is likely to be useful for synthesizing the next line of p_g .



* Program state at step $t = \text{Vector}$ representing the values of variables obtained by executing t lines of the program.

Cross Aggregator (CA)

Idea: If a PE program state* has high relevance with the global program state at a given step, then the following PE program line is likely to be useful for synthesizing the next line of p_g .



Model: Multi-head cross-attention mechanism

Query = Global program state at step t

Key = PE program state at step t

Value = PE program line t+1

*Automatic program synthesis of long programs with a learned garbage collector". Zohar & Wolf, NeurIPS 2018

Results

Timeout for all methods = 5s

	Model	Success Ratio
GPS* Use aggregation mechanisms other than CA	PCCoder [29]	77.75 ± 0.38
	Sum-PEPS	82.71 ± 0.32
	Mean-PEPS	82.68 ± 0.33
	Mean-PEPS+ \mathcal{U}	82.70 ± 0.32
	N-PEPS	86.22 ± 0.25
	N-PEPS+ \mathcal{U}	87.07 ± 0.28



Leading neural program synthesis technique
for the space of programs we work on

Train: programs until length 4
Test: programs of length 4

*Automatic program synthesis of long programs with a learned garbage collector". Zohar & Wolf, NeurIPS 2018

Results

Timeout for all methods = 5s

	Model	Success Ratio
GPS	PCCoder [29]	77.75 ± 0.38
Use aggregation mechanisms other than CA	Sum-PEPS	82.71 ± 0.32
	Mean-PEPS	82.68 ± 0.33
	Mean-PEPS+ \mathcal{U}	82.70 ± 0.32
	N-PEPS	86.22 ± 0.25
	N-PEPS+ \mathcal{U}	87.07 ± 0.28

Train: programs until length 4
Test: programs of length 4

N-PEPS significantly improves the success rate over GPS and other ablation baselines.

Model	Length = 5	Length = 8	Length = 10	Length = 12	Length=14
PCCoder [29]	70.91 ± 0.35	44.17 ± 0.45	28.18 ± 0.33	19.69 ± 0.34	14.71 ± 0.23
Sum-PEPS	76.45 ± 0.33	43.4 ± 0.56	28.96 ± 0.27	20.94 ± 0.32	15.67 ± 0.32
Mean-PEPS	75.79 ± 0.31	44.42 ± 0.51	29.55 ± 0.29	21.45 ± 0.27	16.35 ± 0.27
Mean-PEPS+ \mathcal{U}	75.99 ± 0.32	44.49 ± 0.52	29.75 ± 0.25	21.74 ± 0.30	16.45 ± 0.33
N-PEPS	79.18 ± 0.31	47.23 ± 0.49	32.3 ± 0.34	23.34 ± 0.28	17.35 ± 0.31
N-PEPS+ \mathcal{U}	79.19 ± 0.30	46.31 ± 0.61	31.84 ± 0.36	22.71 ± 0.28	16.68 ± 0.21

Train: programs until length 12

Test: programs of lengths 5, 8, 10, 12 and 14

Takeaways

Connection to our Framework

- **Input X** = set of given IO examples, **Target Y** = step t of the global solution
- **Context Meta-Info W** = Same as X
- **Support Context Z** = PE solutions (values) + step-wise PE execution states (keys) + execution state of step t - 1 of the global solution (query)
- **Enhance** = PE model (for PE solutions) + code interpreter (for execution states)
- **Predict** = Cross Aggregator (CA)

Future Work

- Generalize to programs with loops and conditionals.
- Extend the idea to LLMs.

Outline

On-the-Fly Adaptation of Source Code Models (CAP Workshop, NeurIPS 2020)



RepoFusion: Training Code Models to Understand Your Repository (under review at NeurIPS 2023)



Introduction

Article 1

Article 2

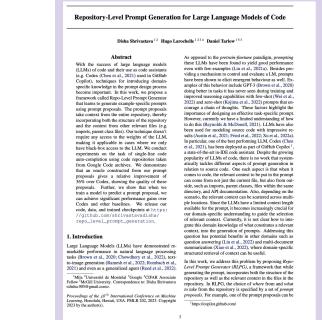
Article 3

Article 4

Final Thoughts



Learning to Combine Per-Example Solutions for Neural Program Synthesis (NeurIPS 2021)



Repository Level Prompt Generation for Large Language Models of Code (ICML 2023)

On-the-Fly Adaptation of Source Code Models

Workshop on Computer Assisted Programming

NeurIPS 2020



Université  de Montréal



Mila

 Google Research

Task: Code Completion in an IDE

Our setting simulates editing a file in an IDE

- Objective: **Complete the first token following the cursor (*target hole*)**
- There can be code following the completion line.
- Rest of the line is blanked.

```
1. package com.asakusafw.windgate.retryable;
2. import java.io.IOException;
3. import java.text.MessageFormat;
4.
5. public class RetryableProcessProfile {
6.     static final WindGateLogger WGLOG = new RetryableProcessLogger(.....);
7.     private static final char SEPARATOR = '.';
8.
9.     } catch (Exception e) {
10.         WGLOG.error(e, "E00001",
11.         profile.getName(),
12.         .....)
```

Code following the line

Blanked-out portion

Cursor Position

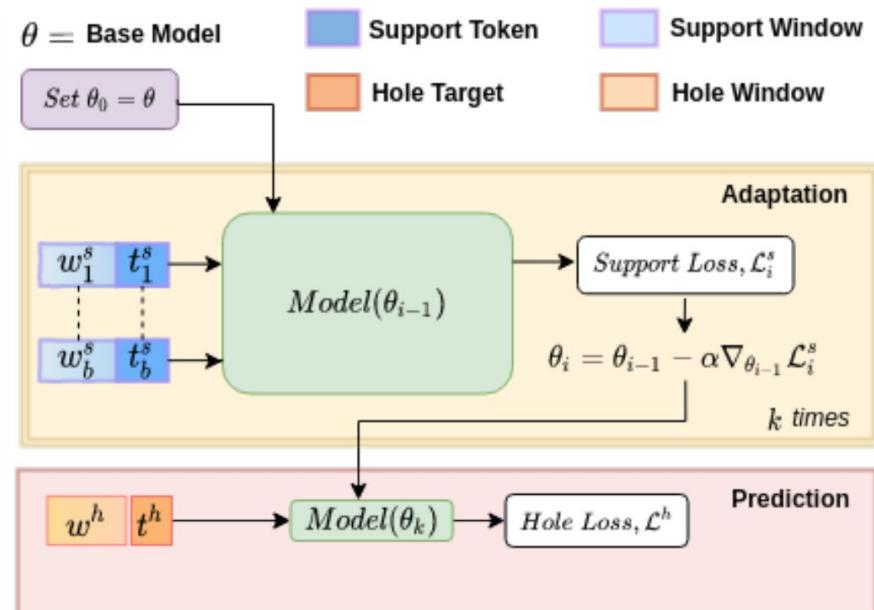
Motivation: Why Adaptation of Source Code Models?

- Models struggle when encountered with code not seen during training.
- Models need to adapt to local, unseen context
 - New Identifiers
 - Organization or project specific coding constructs
 - Variable naming conventions (`get_access` vs `getAccess`)
 - Data structures/ libraries used (`from google3 import b`)
 - Developer-specific preferences
 - `for (int i = 0, ...)` vs `for (int j = 0, ...)`
 - Comments before each line or each method

Targeted Support Set Adaptation (TSSA)

- **Enhance:** Obtain support tokens, e.g. frequent in current file but rare overall.
- **Predict:** Adapt the model based on the support context.
 - *Inner update:* support window → support token (k steps of gradient update)
 - *Outer update:* hole window → hole target (using updated parameters)

```
1. package org.oddjob;
2. import java.util.Properties;
3. import org.oddjob.arooa.ArooaDescriptor;
4.
5. import org.oddjob.arooa.standard.
6.     StandardArooaDescriptor;
7. import org.oddjob.arooa.standard.
8.     StandardPropertyManager;
9.
10. switch(inherit){
11. case NONE:
12.     propertyManager = new StandardPropertyManager(
13.         properties, propertySourceName);
14.     break;
15.
16. }
17.
18. }
```



Results

Test Performance on Target Hole Prediction

Model	Cross Entropy	MRR@10 (All)(%)	MRR@10 (Identifiers)(%)	Recall@10 (All)(%)	Recall@10 (Identifiers) (%)
Base Model	5.222 ± 0.10	65.20 ± 0.42	24.90 ± 0.64	75.74 ± 0.42	36.20 ± 0.78
Dynamic Evaluation	3.540 ± 0.08	68.95 ± 0.41	34.44 ± 0.70	80.39 ± 0.39	48.86 ± 0.82
TSSA-1	3.461 ± 0.07	66.94 ± 0.40	35.76 ± 0.70	81.00 ± 0.38	52.04 ± 0.82
TSSA-8	3.383 ± 0.06	67.52 ± 0.40	35.14 ± 0.70	80.65 ± 0.38	50.27 ± 0.82
TSSA-16	3.240 ± 0.06	68.63 ± 0.40	36.74 ± 0.70	81.51 ± 0.38	52.34 ± 0.82

- **Model architecture:** Seq2seq encoder decoder network with single-layer GRU.
- **Base Model:** no adaptation
- **Dynamic Evaluation***: Support tokens consist of tokens from context before the target hole.
- **TSSA-k:** TSSA with k updates with support tokens from both before and after the target hole.
- We set k = avg. # of updates performed by dynamic evaluation = 16 for our test data.

TSSA improves upon adaptation (dynamic evaluation) and non-adaptation baselines, even with half the #steps on some metrics.

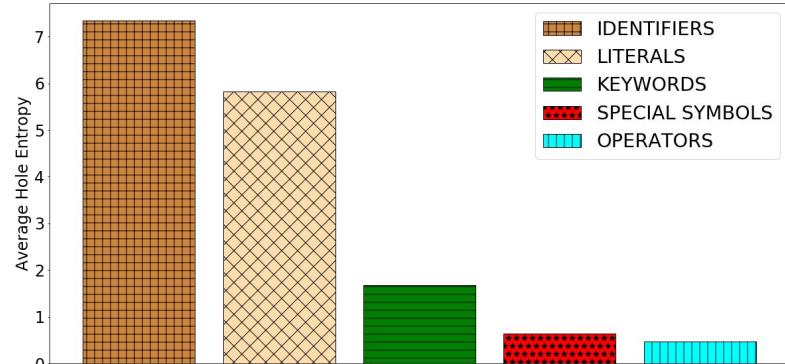
Results

Test Performance on Target Hole Prediction

Model	Cross Entropy	MRR@10 (All)(%)	MRR@10 (Identifiers)(%)	Recall@10 (All)(%)	Recall@10 (Identifiers) (%)
Base Model	5.222 ± 0.10	65.20 ± 0.42	24.90 ± 0.64	75.74 ± 0.42	36.20 ± 0.78
Dynamic Evaluation	3.540 ± 0.08	68.95 ± 0.41	34.44 ± 0.70	80.39 ± 0.39	48.86 ± 0.82
TSSA-1	3.461 ± 0.07	66.94 ± 0.40	35.76 ± 0.70	81.00 ± 0.38	52.04 ± 0.82
TSSA-8	3.383 ± 0.06	67.52 ± 0.40	35.14 ± 0.70	80.65 ± 0.38	50.27 ± 0.82
TSSA-16	3.240 ± 0.06	68.63 ± 0.40	36.74 ± 0.70	81.51 ± 0.38	52.34 ± 0.82

Most of the improvement comes from identifiers and literals.

Test Performance across different token-types



Token Type	Base model	TSSA-16	% Improvement
Identifiers	13.16	7.35	44.15
Literals	7.18	5.82	18.94

Takeaways

Connection to our Framework

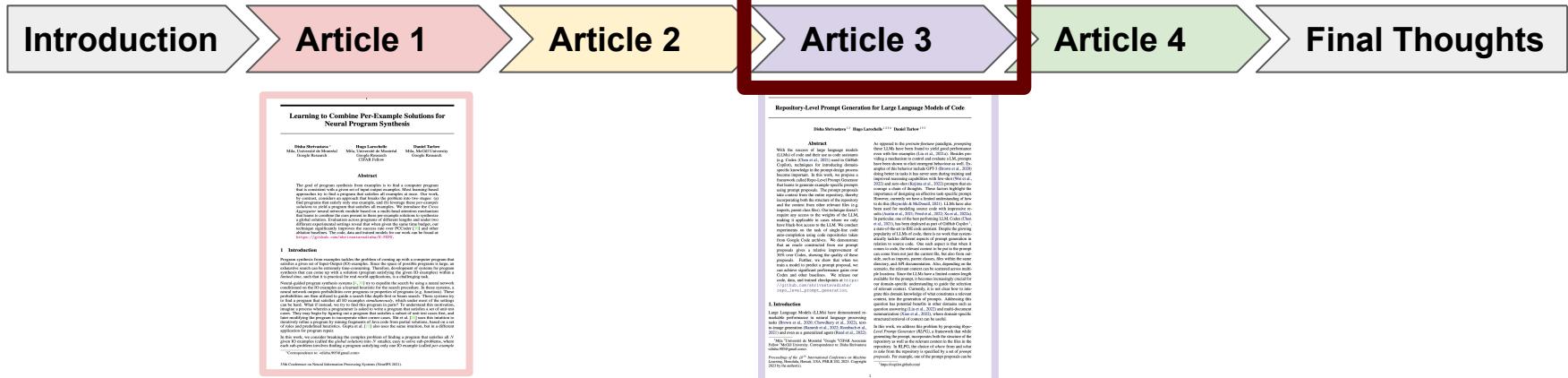
- **Input X** = hole window, **Target Y** = target hole (next token after the cursor)
- **Context Meta-Info W** = position of the cursor + current file
- **Support Context Z** = support tokens + support windows from the current file
- **Enhance** = targeted selection of support context, e.g. strategies based on frequency of occurrence of tokens
- **Predict** = TSSA

Future Work

- Better ways of obtaining the support context
 - Extend the scope from current file to the entire repository.
 - Automated, Example-specific selection
- Leverage the power of pretrained LLMs
 - Expensive to perform gradient updates

Outline

On-the-Fly Adaptation of Source Code Models (*CAP Workshop, NeurIPS 2020*)



Learning to Combine Per-Example Solutions for Neural Program Synthesis (NeurIPS 2021)

Repository-Level Prompt Generation for Large Language Models of Code

ICML 2023



Université  de Montréal

 Mila

Google Research

Code, data and trained checkpoints: https://github.com/shrivastavadisha/repo_level_prompt_generation

Motivation: Large Language Models (LLMs) of Code

- Used in code-assistants (e.g. GitHub Copilot, Bard).
- Struggle when encountered with code not seen during training.
 - Proprietary Software
 - WIP Code Project
- Finetuning on code from the local repository is often impractical
 - Black-box access to strong code LLMs.
 - Computationally expensive as well as challenging to update frequently.
- Building upon previous work, leverage relevant context from other files in the repository (e.g. imports, parent classes), but only during inference.

Motivation: Large Language Models (LLMs) of Code

- Used in code-assistants (e.g. GitHub Copilot, Bard).
- Struggle when encountered with code not seen during training.
 - Proprietary Software
 - WIP Code Project
- Finetuning on code from the local repository is often impractical
 - Black-box access to strong code LLMs.
 - Computationally expensive as well as challenging to update frequently.
- Building upon previous work, leverage relevant context from other files in the repository (e.g. imports, parent classes), but only during inference.

Select relevant repository context in a way that doesn't require access to the weights of the LLM.

Task: Single-line Code Completion in an IDE

Our setting simulates editing a file in an IDE

- Objective: Complete the line following the cursor (*target hole*)
- There can be code after the cursor line.

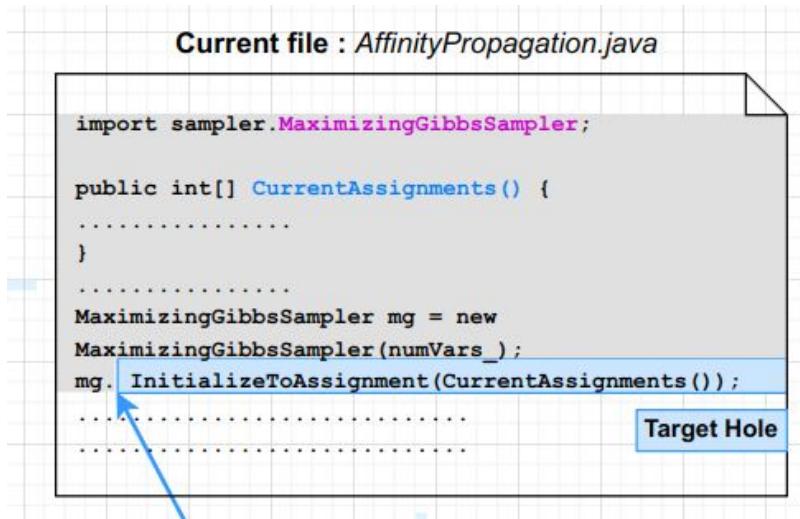
Current file : *AffinityPropagation.java*

```
import sampler.MaximizingGibbsSampler;

public int[] CurrentAssignments() {
    .....
}

MaximizingGibbsSampler mg = new
MaximizingGibbsSampler(numVars_);
mg. InitializeToAssignment(CurrentAssignments());
.....
.....
```

Target Hole



Cursor Position

Task: Single-line Code Completion in an IDE

Our setting simulates editing a file in an IDE

- Objective: Complete the line following the cursor (**target hole**)
- There can be code after the cursor line.

Vanilla Training: given a prefix of code, predict the next tokens.

Vanilla Inference (to match the training): take context prior to the cursor in the current file and predict the **target hole**.

Current file : AffinityPropagation.java

```
import sampler.MaximizingGibbsSampler;

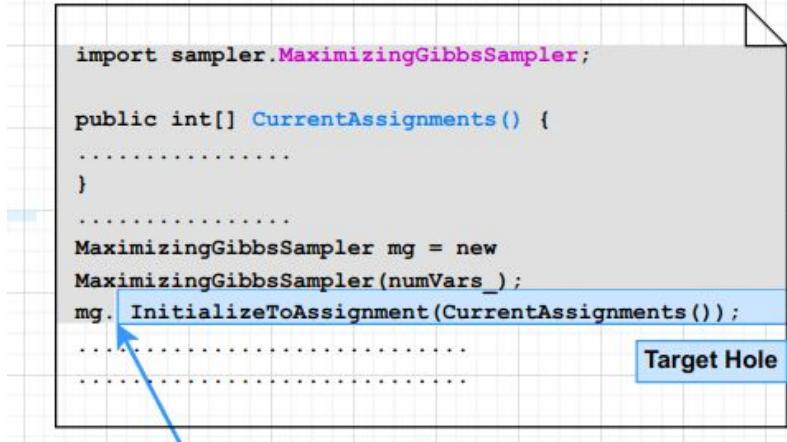
public int[] CurrentAssignments() {
    .....
}

MaximizingGibbsSampler mg = new
MaximizingGibbsSampler(numVars_);
mg. InitializeToAssignment(CurrentAssignments());
.....

```

Target Hole

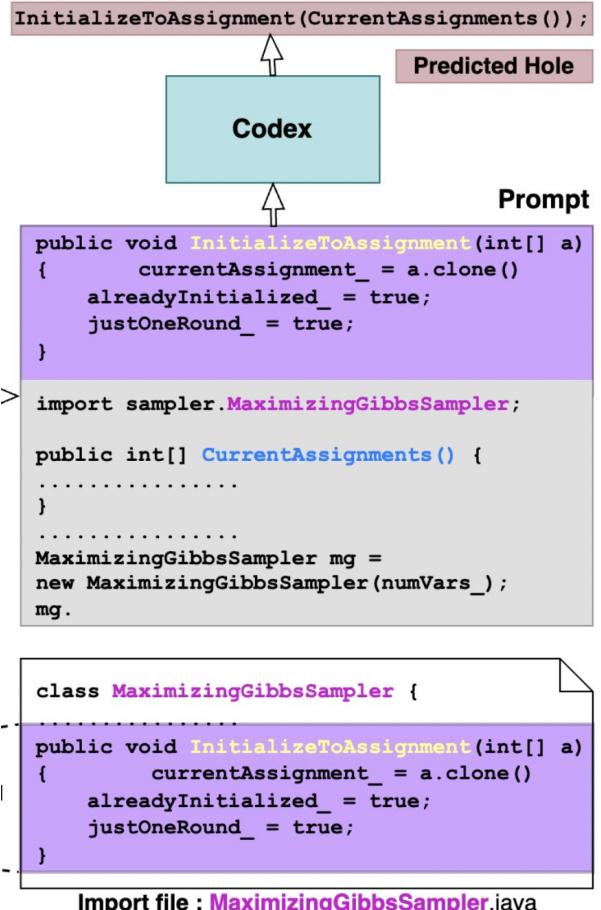
Cursor Position



Repository Context in the Prompt

Take an LLM trained in the usual way, but use it differently during inference.

During inference, in addition to the prior context in the current file, we add **relevant context from the repository** in the prompt.



Repository Context in the Prompt

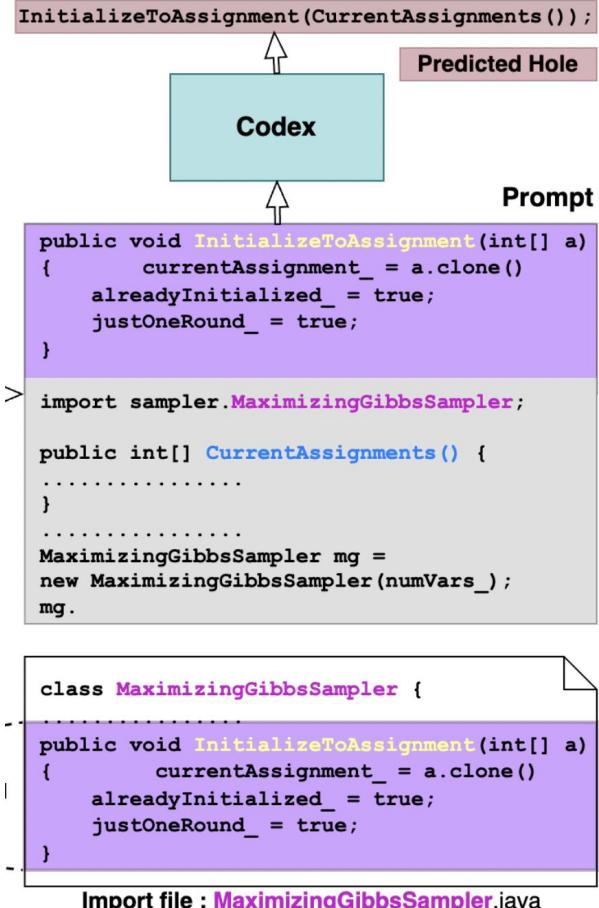
Take an LLM trained in the usual way, but use it differently during inference.

During inference, in addition to the prior context in the current file, we add **relevant context from the repository** in the prompt.

To select relevant context, we want a method that

- Utilizes Structure of the repository
- Utilizes Context in relevant files

Solution: Use domain knowledge to guide the selection of relevant context via a set of **prompt proposals**.



Prompt Proposals

- **Prompt Source**: where to take the context from?
- **Prompt Context Type**: what to take from the prompt source?

Prompt Proposals

- **Prompt Source**: where to take the context from?
- **Prompt Context Type**: what to take from the prompt source?

10 Prompt Sources

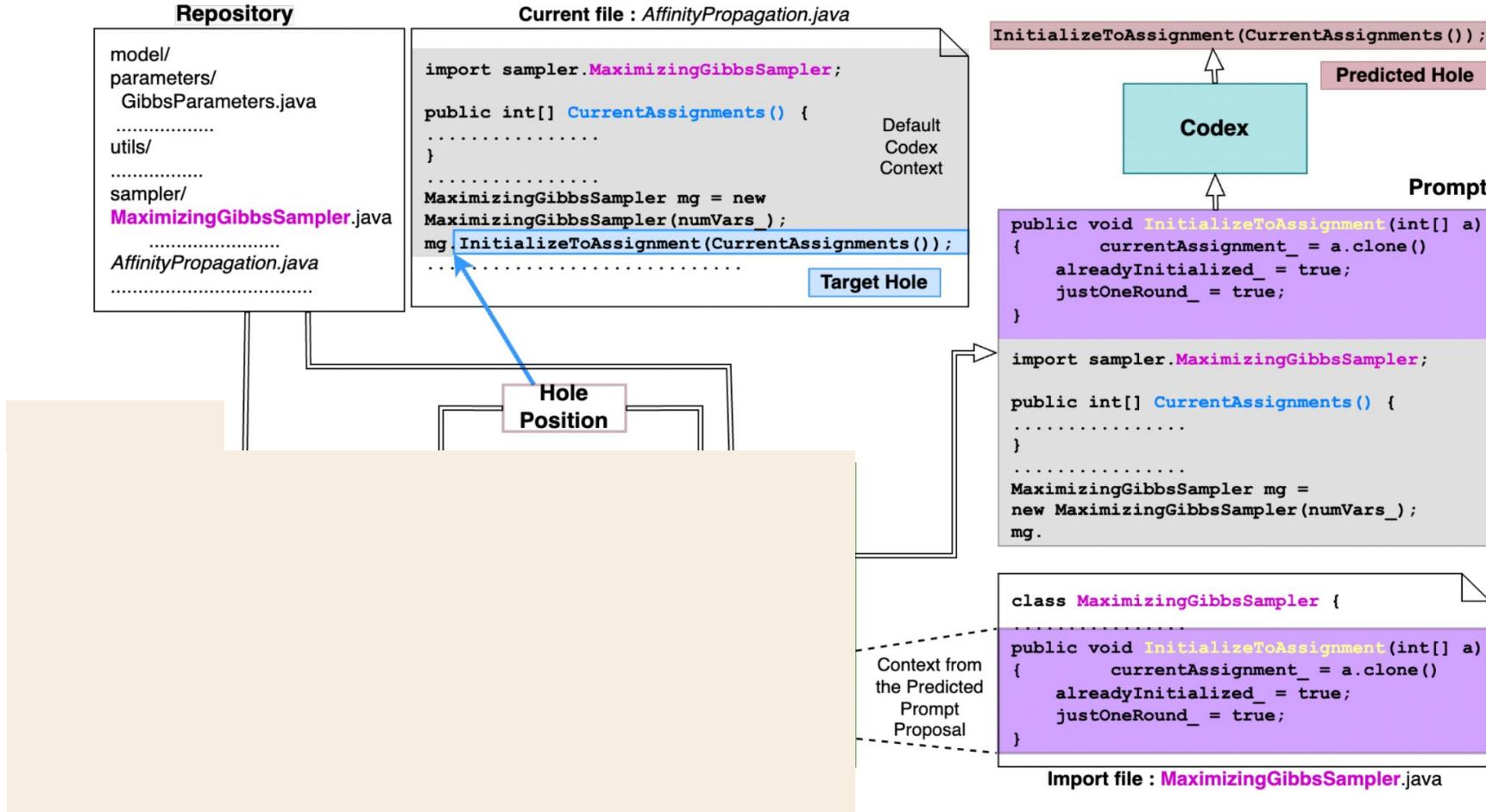
- Current file
- Parent Class file
- Sibling file
- Similar name file
- Child Class file
- Import of the above

In total, we propose a list of **63** prompt proposals

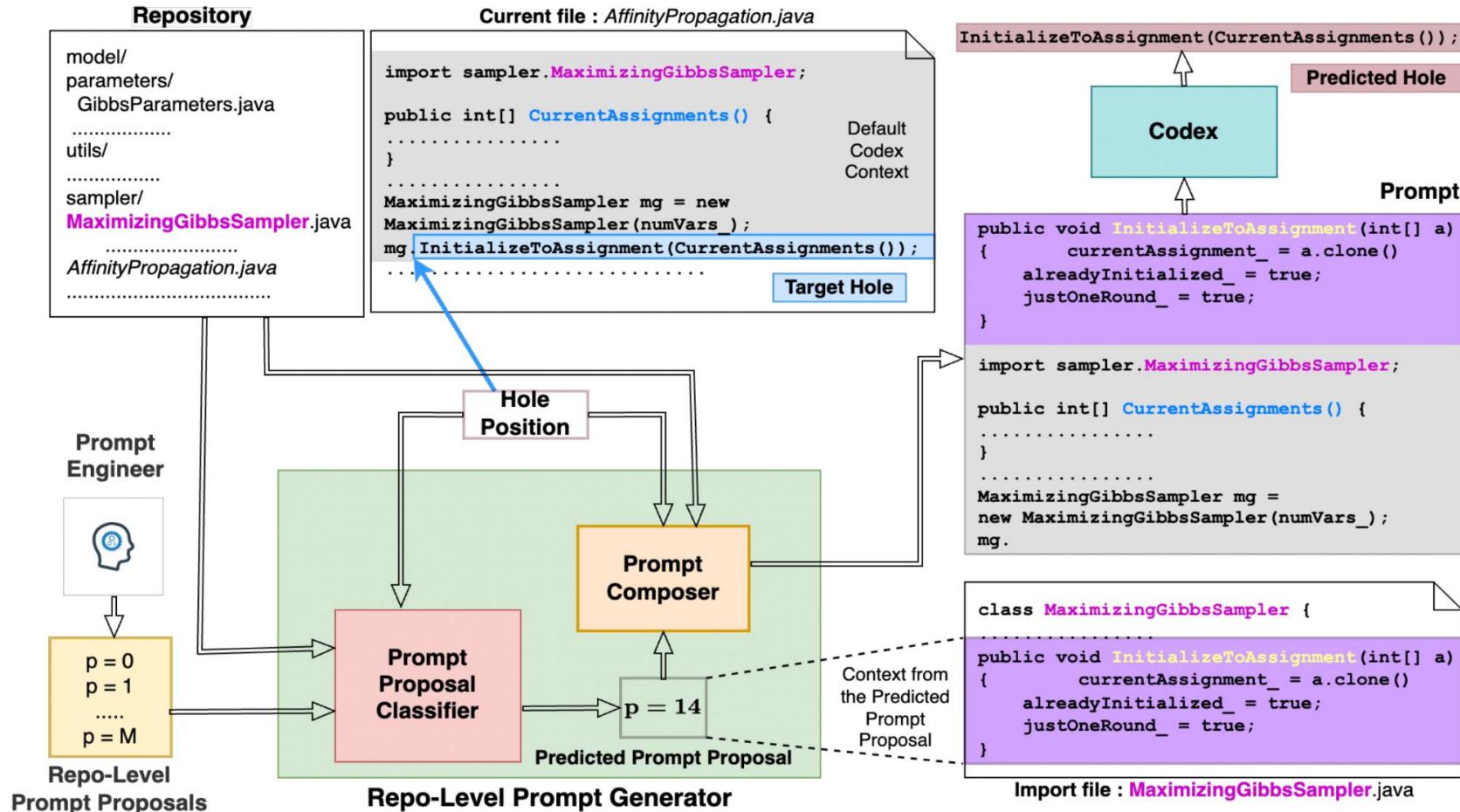
7 Prompt Context Types*

- Lines after the cursor
- Identifiers
- Field declarations
- Type identifiers
- String literals
- Method names
- Method names and bodies

Repo-Level Prompt Generator (RLPG)



Repo-Level Prompt Generator (RLPG)



Prompt Proposal Classifier

- Multi-label binary classifier that *learns* to select a prompt proposal that is likely to lead to a successful prediction for the target hole.
- *Success* = When inclusion of the context from the prompt proposal in the prompt leads to an accurate prediction of the hole.
- *Example-Specific*: different prediction conditioned on the hole.

Results

Table 2. Performance of the oracle relative to Codex.

Data Split	Success Rate Codex(%)	Success Rate Oracle(%)	Rel. ↑ over Codex(%)
Train	59.78	80.29	34.31
Val	62.10	79.05	27.28
Test	58.73	79.63	35.58

Including contexts from our prompt proposals during inference is quite useful even though Codex has not seen them during training.

Results

Table 2. Performance of the oracle relative to Codex.

Data Split	Success Rate Codex(%)	Success Rate Oracle(%)	Rel. ↑ over Codex(%)
Train	59.78	80.29	34.31
Val	62.10	79.05	27.28
Test	58.73	79.63	35.58

Using RLPG with prompt proposal classifier shows significant improvements.

Retrieval Baselines {

Non-learned RLPG {

Learned RLPG {

Table 3. Success Rate (SR) of different methods on the test data when averaged across all holes.

Method	Success Rate(%)	Rel. ↑(%)
Codex (Chen et al., 2021)	58.73	-
Oracle	79.63	35.58
Random	58.13	-1.02
Random NN	58.98	0.43
File-level BM25	63.14	7.51
Identifier Usage (Random)	64.93	10.55
Identifier Usage (NN)	64.91	10.52
Fixed Prompt Proposal	65.78	12.00
RLPG-BM25	66.41	13.07
RLPG-H	68.51	16.65
RLPG-R	67.80	15.44

Takeaways

Connection to our Framework

- **Input X** = all tokens prior to the cursor in the current file, **Target Y** = tokens after the cursor till end of line.
- **Context Meta-Info W** = position of the cursor + current file's repository
- **Support Context Z** = context from a single prompt proposal predicted by RLPG
- **Enhance** = Prompt Proposals + RLPG
- **Predict** = LLM of Code

Future Work

- Automatically combine contexts from multiple prompt proposals.
- Scale the evaluation to larger data and include comparisons with more code LLMs.

Outline

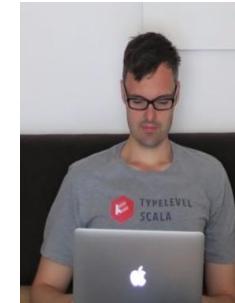
On-the-Fly Adaptation of Source Code Models (*CAP Workshop, NeurIPS 2020*)



Learning to Combine Per-Example Solutions for Neural Program Synthesis (NeurIPS 2021)

RepoFusion: Training Code Models to Understand Your Repository

arXiv 2023 (under review)



Code, data and trained checkpoints: <https://huggingface.co/RepoFusion>

Task: Single-line Code Completion in an IDE

Our setting simulates editing a file in an IDE

- Objective: Complete the line following the cursor (*target hole*)
- There can be code after the cursor line.

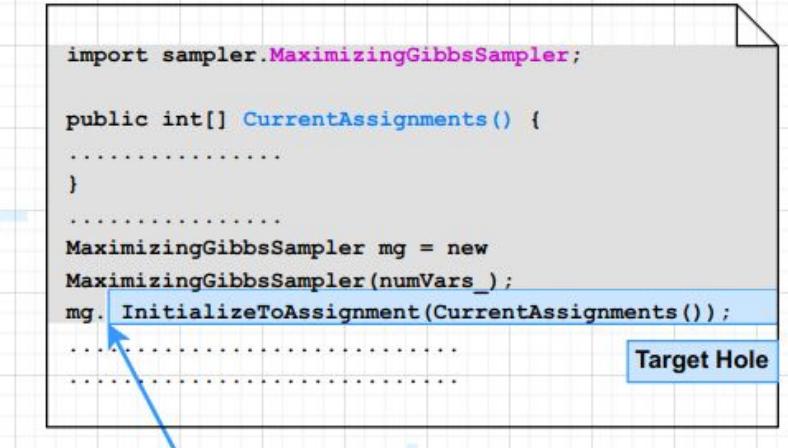
Current file : *AffinityPropagation.java*

```
import sampler.MaximizingGibbsSampler;

public int[] CurrentAssignments() {
    .....
}

MaximizingGibbsSampler mg = new
MaximizingGibbsSampler(numVars_);
mg. InitializeToAssignment(CurrentAssignments());
.....
.....
```

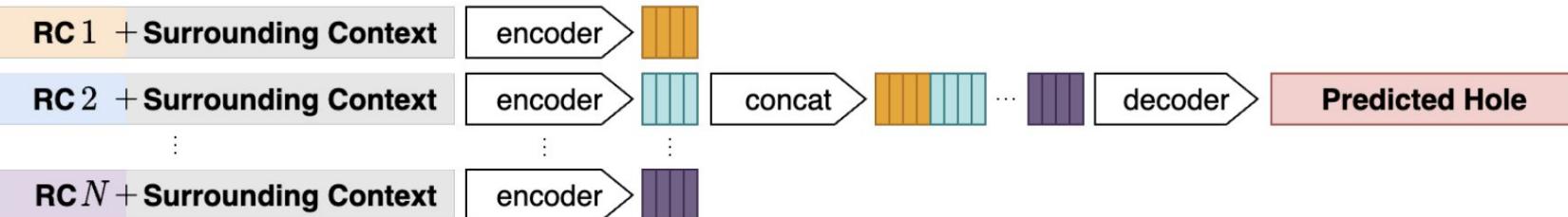
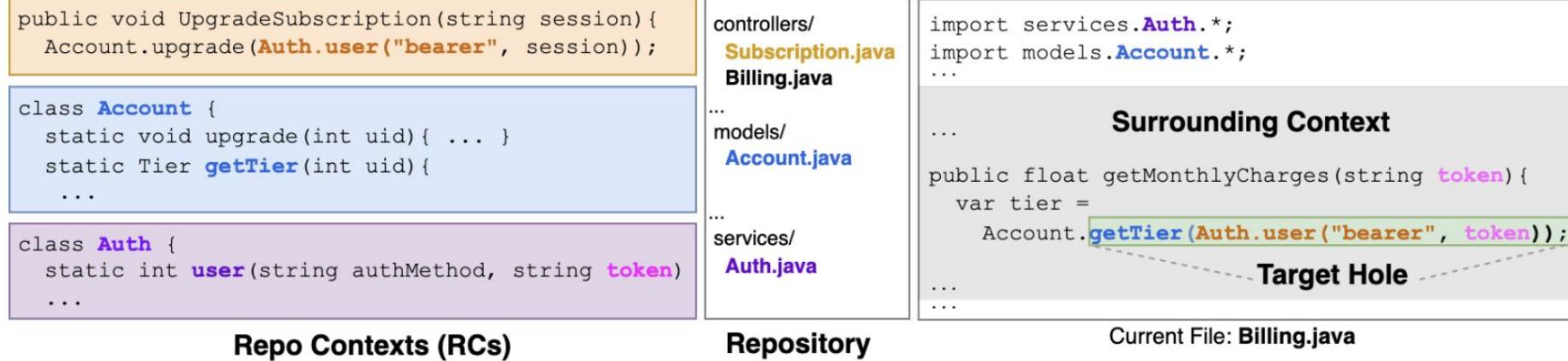
Target Hole



Cursor Position

RepoFusion

Train a model to **combine multiple relevant contexts** coming from the repository (repo contexts) such that it leads to an **accurate** prediction of the target hole.



Results

RepoFusion (220M) outperforms ~73X larger (CodeGen-16B) models trained with next-token prediction.

N = #RCs

l = size (# tokens) of each RC

RepoFusion (220M) is at par with ~70X larger StarCoder-15.5B model trained with Fill-in-the-Middle.

Model	Size (#params)	Effective context length	Context type	Success Rate (%)
CodeT5-base (FT)	0.22B	2048	prior	41.82 ± 0.12
CodeT5-base (FT)	0.22B	4096	prior	46.45 ± 0.12
CodeT5-large (FT)	0.77B	2048	prior	44.73 ± 0.12
CodeT5-large (FT)	0.77B	4096	prior	48.92 ± 0.12
SantaCoder	1.1B	2048	prior	39.51 ± 0.12
CodeGen	2B	2048	prior	49.45 ± 0.12
CodeGen	6B	2048	prior	49.19 ± 0.12
CodeGen	16B	2048	prior	50.20 ± 0.12
CodeT5-base (FT)	0.22B	2048	post+prior	48.89 ± 0.12
CodeT5-base (FT)	0.22B	4096	post+prior	49.97 ± 0.12
CodeT5-large (FT)	0.77B	2048	post+prior	51.72 ± 0.12
CodeT5-large (FT)	0.77B	4096	post+prior	52.43 ± 0.12
SantaCoder	1.1B	2048	post+prior	56.78 ± 0.12
CodeGen	2B	2048	post+prior	53.18 ± 0.12
CodeGen	6B	2048	post+prior	54.03 ± 0.12
CodeGen	16B	2048	post+prior	54.09 ± 0.12
RepoFusion ($N = 4, l = 512$)	0.22B	2048	NT-Prior-Last	65.96 ± 0.12
RepoFusion ($N = 8, l = 512$)	0.22B	4096	NT-Prior-Last	70.38 ± 0.11
RepoFusion ($N = 32, l = 768$)	0.22B	24576	NT-Prior-Last	77.32 ± 0.10

StarCoderBase	15.5B	8192	prior	52.97 ± 0.45
StarCoderBase	15.5B	8192	post+prior	79.79 ± 0.36
RepoFusion ($N = 16, l = 512$)	0.22B	8192	NT-Prior-Last	73.67 ± 0.43
RepoFusion ($N = 32, l = 2500$)	0.22B	80000	NT-Prior-Last	78.33 ± 0.37

Results

RepoFusion
outperforms
(CodeGen)
with next-best

Training smaller models with repository context
using RepoFusion is better or at par with training
significantly larger models without such context.

N = #RCs

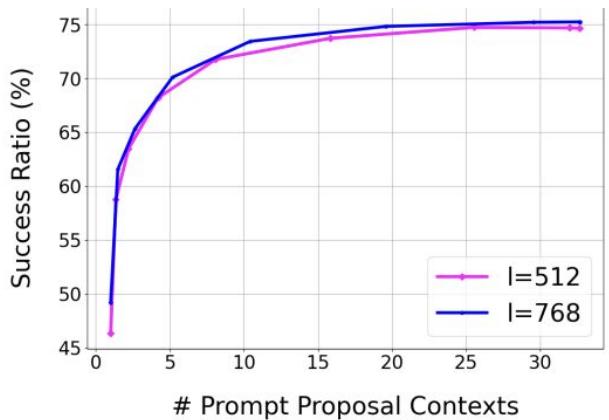
l = size (# tokens) of each RC

RepoFusion (220M) is at par
with ~70X larger
StarCoder-15.5B model trained
with Fill-in-the-Middle.

Model	Size (#params)	Effective context length	Context type	Success Rate (%)
CodeT5-base (FT)	0.22B	2048	prior	41.82 ± 0.12
CodeT5-base (FT)	0.22B	4096	prior	46.45 ± 0.12
CodeT5-large (FT)	0.77B	2048	prior	44.73 ± 0.12
CodeT5-large (FT)	0.77B	4096	prior	48.92 ± 0.12
SantaCoder	1.1B	2048	prior	39.51 ± 0.12
CodeGen	2B	2048	prior	49.45 ± 0.12
				49.19 ± 0.12
				50.20 ± 0.12
			prior	48.89 ± 0.12
			prior	49.97 ± 0.12
			prior	51.72 ± 0.12
			prior	52.43 ± 0.12
			prior	56.78 ± 0.12
			prior	53.18 ± 0.12
			prior	54.03 ± 0.12
			prior	54.09 ± 0.12

StarCoderBase	15.5B	8192	prior	52.97 ± 0.45
StarCoderBase	15.5B	8192	post+prior	79.79 ± 0.36
RepoFusion ($N = 16, l = 512$)	0.22B	8192	NT-Prior-Last	73.67 ± 0.43
RepoFusion ($N = 32, l = 2500$)	0.22B	80000	NT-Prior-Last	78.33 ± 0.37

Results



Performance scales with incorporation diverse repo contexts from multiple sources.

Model	Size (#params)	Effective context length	Context type	Success Rate (%)
CodeT5-base (FT)	0.22B	2048	prior	41.82 ± 0.12
CodeT5-base (FT)	0.22B	4096	prior	46.45 ± 0.12
CodeT5-large (FT)	0.77B	2048	prior	44.73 ± 0.12
CodeT5-large (FT)	0.77B	4096	prior	48.92 ± 0.12
SantaCoder	1.1B	2048	prior	39.51 ± 0.12
CodeGen	2B	2048	prior	49.45 ± 0.12
CodeGen	6B	2048	prior	49.19 ± 0.12
CodeGen	16B	2048	prior	50.20 ± 0.12
CodeT5-base (FT)	0.22B	2048	post+prior	48.89 ± 0.12
CodeT5-base (FT)	0.22B	4096	post+prior	49.97 ± 0.12
CodeT5-large (FT)	0.77B	2048	post+prior	51.72 ± 0.12
CodeT5-large (FT)	0.77B	4096	post+prior	52.43 ± 0.12
SantaCoder	1.1B	2048	post+prior	56.78 ± 0.12
CodeGen	2B	2048	post+prior	53.18 ± 0.12
CodeGen	6B	2048	post+prior	54.03 ± 0.12
CodeGen	16B	2048	post+prior	54.09 ± 0.12
RepoFusion ($N = 4, l = 512$)	0.22B	2048	NT-Prior-Last	65.96 ± 0.12
RepoFusion ($N = 8, l = 512$)	0.22B	4096	NT-Prior-Last	70.38 ± 0.11
RepoFusion ($N = 32, l = 768$)	0.22B	24576	NT-Prior-Last	77.32 ± 0.10
StarCoderBase	15.5B	8192	prior	52.97 ± 0.45
StarCoderBase	15.5B	8192	post+prior	79.79 ± 0.36
RepoFusion ($N = 16, l = 512$)	0.22B	8192	NT-Prior-Last	73.67 ± 0.43
RepoFusion ($N = 32, l = 2500$)	0.22B	80000	NT-Prior-Last	78.33 ± 0.37

Takeaways

Connection to our Framework

- **Input X** = all tokens prior to the cursor in the current file, **Target Y** = tokens after the cursor till the end of line.
- **Context Meta-Info W** = position of the cursor + current file's repository
- **Support Context Z** = multiple repo contexts
- **Enhance** = module for obtaining repo contexts
- **Predict** = RepoFusion

Future Work

Leverage contextual cues from other relevant sources such as API documentations, StackOverflow, bug reports, GitHub issues.

We create and release [Stack-Repo](#), a dataset of 200 Java repositories with permissive licenses and near-deduplicated files that are augmented with three types of repository contexts.

Outline

On-the-Fly Adaptation of Source Code Models (*CAP Workshop, NeurIPS 2020*)



Learning to Combine Per-Example Solutions for Neural Program Synthesis (NeurIPS 2021)

Broad Applicability of Our Framework

Size of the Support Context

- Limited context can be given as input to Predict
- Combining multiple relevant contexts such as in RepoFusion
 - Determining the optimal number and size of each relevant context
- LLM with large context window
- Retrieval-augmented models that work with external memory
- Comes with increased inference costs

Broad Applicability of Our Framework

Size of the Support Context

Capturing the Dependence between Enhance and Predict

- Predict should learn to effectively leverage Z provided by Enhance
- Enhance should use the feedback signal from Predict to guide the selection of Z
- Joint training of Enhance and Predict difficult in practise.
- Separate training offers more flexibility
 - Predict: Larger LLM, trained on large data
 - Enhance: Smaller model, task-specific training on curated data.

Broad Applicability of Our Framework

Size of the Support Context

Capturing the Dependence between Enhance and Predict

Generality of the Support Context

- Automatic selection of Z conditioned on the task
- Instruction-tuned LLM as both Enhance and Predict
 - Generate relevant contextual cues when prompted with instructions capturing the task (challenging to make this work across diverse tasks)
 - Use the generated contextual cues as input to generate predictions
 - Can do these iteratively to refine the predictions.

Broad Applicability of Our Framework

Size of the Support Context

Capturing the Dependence between Enhance and Predict

Generality of the Support Context

Human-in-the-loop

- Human-interpretable contextual cues from Enhance
 - More control over what goes in the Predict stage such as prompt proposals
- Utilize human feedback to come up with better metrics and refine predictions to better align with user's preferences.

Broad Applicability of Our Framework

Size of the Support Context

Capturing the Dependence between Enhance and Predict

Generality of the Support Context

Human-in-the-loop

Performance-Latency Tradeoff

Optimizing resource allocation between Enhance and Predict (especially during inference) to match specific time and computational requirements.

Going Forward

Modeling the Code Ecosystem

Derive contextual cues from the complex programming workflow

- Iterative and dynamic aspect
 - Different program stages: *writing -> testing-> committing -> maintaining*
 - Codebases keep evolving
- Interaction with tools
 - Compiler
 - Static Analyzer
 - GitHub
 - Web, e.g. StackOverflow
- Interaction with other developers
 - Code reviewers
 - Collaborators

Going Forward

Modeling the Code Ecosystem

Derive contextual cues from the complex programming workflow

- Iterative and dynamic aspect
 - Different program stages: *writing -> testing-> committing -> maintaining*
 - Codebases keep evolving
- Interaction with tools
 - Compiler
 - Static Analyzer
 - GitHub
 - Web, e.g. StackOverflow
- Interaction with other developers
 - Code reviewers
 - Collaborators

Modeling the User

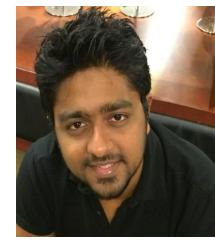
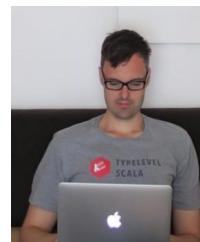
Inform the selection of contextual cues and predictions based on user interactions

- Metrics based on user preferences
 - Acceptance rate
 - User edits
- Mode of user interaction [1]
 - *Accelerated*: fixed contextual cues, single, short predictions
 - *Exploratory*: diverse contextual cues, several, long predictions
- Changing user beliefs [2]
 - Dynamically adapt the model
 - Align more with user values: *agency, creativity, trust, verifiability*

[1] “Grounded Copilot: How Programmers Interact with Code-Generating Models”. Barke et al. (2022)

[2] “Approach Intelligent Writing Assistants Usability with Seven Stages of Action”. Bhat et al. (2023)

Thank You



Questions/ Comments

