



# AI ASSISTED CODING


IDM SPRING 2023 SYMPOSIUM

CHRISTOPHER LORTON

# BREAKOUT SESSION RESOURCES

Source Code Repository with instructions to get started with hands-on exercise:

<https://tinyurl.com/idmai23>



MAGICAL BUT BY  
NO MEANS  
INFALLIBLE.

BRING YOUR CODING SKILLS!  
WHAT IS “BETTER”?

# CHATGPT V GITHUB COPILOT

## ChatGPT

- Built on GPT-3.5 and GPT-4
- Fine-tuned for *conversational* applications
- GPT-3.5 4k tokens (8k) / GPT-4 8k (32k)
- GPT-4  $10^{12}$  parameters(?)

## GitHub Copilot

- Built on OpenAI Codex (GPT-3 + 159GB Python code from 54M GitHub repositories)
- “Featuring a general purpose programming model, OpenAI Codex can be applied to essentially any programming task. OpenAI has used it for transpilation as well as for explaining and refactoring code.”

# SCENARIOS

- Explain Existing Code
  - Better than humans? TBD
- Optimize Existing Code
  - Effective with knowledgeable guidance
- Translate Code from One Coding Language to Another
- Write Code from Scratch
  - Incremental
  - With Tests
- Basic SIR (R) and SEIR (Python) models from scratch
  - With caveats



# PERSONAL EXAMPLES

```
void write_array_to_file(const char* filename, PyArrayObject* arr, int n) {
    PyArray_Descr* dtype = PyArray_DescrFromType(arr->descr->typecode);
    FILE* f = fopen(filename, "wb");
    fwrite(arr->data, dtype->elsize, n, f);
    fclose(f);
}

void calculate_block_size(int n_rows, int n_cols, int n_blocks) {
    // Square root of the number of rows in the CPU cache.
    int n_cache_rows = (int) sqrt(n_rows); // 22 (22*22=484)
    int n_cache_cols = (int) sqrt(n_cols); // 22 (22*22=484)
    int n_cache_size = n_cache_rows * n_cache_cols; // 484
    PRINTF("Cache size: %d\n", n_cache_size);
    // Block size is the number of rows in the cache.
    int n_block_rows = n_cache_rows;
    PRINTF("Block size: %d\n", n_block_rows);
    // Number of blocks is the number of rows in the array divided by the number of rows in the block.
    int n_blocks = (n_rows + n_block_rows - 1) / n_block_rows;
    PRINTF("Number of blocks: %d\n", n_blocks);
    return;
}
```

- Writing a script in the **Julia** programming language.
- Extracting data from a JSON formatted data file.
- Writing docstrings for **Python** functions.
- Translating Intel AVX instructions to ARM NEON (**C++**).
- Translating **MATLAB** code to Python.
- Writing plotting code in Python (matplotlib) and **R**.

# JUST FOR FUN: FIZZBUZZ IN LOLCODE

User: Can you re-write the classic FizzBuzz problem in LOLCODE?

ChatGPT: Sure! Here's the classic FizzBuzz problem implemented in LOLCODE:

```
HAI 1.2
CAN HAS STDIO?

I HAS A NUM
I HAS A LIMIT
GIMMEH "Enter the limit: " ANS
VISIBLE "FizzBuzz counting up to " ANS "!"
```

```
I HAS A COUNT ITZ 1
IM IN YR LOOP UPPIN YR COUNT TIL BOTH SAEM COUNT AN ANS
I HAS A MOD3 ITZ MOD OF COUNT AN 3
I HAS A MOD5 ITZ MOD OF COUNT AN 5

BOTH SAEM MOD3 AN 0
O RLY?
YA RLY
  BOTH SAEM MOD5 AN 0
  O RLY?
  YA RLY
    VISIBLE "FizzBuzz"
  NO WAI
  VISIBLE "Fizz"
  OIC
NO WAI
  BOTH SAEM MOD5 AN 0
  O RLY?
  YA RLY
    VISIBLE "Buzz"
  OIC
OIC

IZ COUNT BIGGER THAN ANS
O RLY?
YA RLY
  GTFO
  OIC
IM OUTTA YR LOOP

KTHXBYE
```



# EXPLAIN EXISTING CODE

```
load ../data/X_ref.mat;
X_ref = X;
load ../data/X_nref.mat;
X_nref = X;
clear X;
for patIndex = 1 : length(X_ref(1,:))
    msg = sprintf( "Processing sample %d", pathIndex )
    disp( msg)
    % these are SNP positions, P. falc has 5300 genes, s
    for geneIndex = 1 : length(X_nref(:,1))
        % cleaning and normalizing read count data
        totRead = X_ref(geneIndex,patIndex) + X_nref(geneIndex,patIndex);
        if totRead < 5
            X(geneIndex,patIndex) = 4;
            Xa(geneIndex,patIndex) = nan;
        elseif (X_nref(geneIndex,patIndex)<=1) && (X_ref(geneIndex,patIndex)>=1)
            X(geneIndex,patIndex) = 0;
            Xa(geneIndex,patIndex) = 0;
        elseif (X_ref(geneIndex,patIndex)<=1) && (X_nref(geneIndex,patIndex)>=1)
            X(geneIndex,patIndex) = 1;
            Xa(geneIndex,patIndex) = 1;
        else
            X(geneIndex,patIndex) = 2;
            Xa(geneIndex,patIndex) = X_nref(geneIndex,patIndex);
        end
    end
end
```

a. If `totRead` is less than 5, it means the read count is too low. In this case, the code assigns a value of 4 to `X(geneIndex,patIndex)`, and assigns NaN (Not-a-Number) to `Xa(geneIndex,patIndex)`.

d. If none of the above conditions are met, the code assigns a value of 2 to `X(geneIndex,patIndex)`, and calculates the ratio of `X_nref(geneIndex,patIndex)` to `totRead` and assigns it to `Xa(geneIndex,patIndex)`.

In summary, this code loads read count data from two files, creates copies of the data, and then performs data cleaning and normalization based on certain conditions. The cleaned and normalized data is stored in the variable `X`, and additional normalized data is stored in the variable `Xa`.

ed in two matrices, `X_ref` and `X_nref`. Here's a breakdown of the code:

mands load the contents of the files "X\_ref.mat" and "X\_nref.mat" respectively, which contain the `X_ref`.

ed data and store them in new variables `X_ref` and `X_nref`. This step is done to preserve the original

create the copies in the previous step. This is done to avoid confusion and potential errors later in

Index) and each gene (geneIndex) in the read count data.

es a string message indicating the current sample being processed.

window.

for each gene in the current sample.

; This line calculates the total read count for the current gene and sample by summing the read

o determine how to clean and normalize the data:

is case, the code assigns a value of 4 to `X(geneIndex,patIndex)`, and assigns NaN (Not-a-Number) to

or equal to 1 while `X_ref` is greater than or equal to 5, the code assigns a value of 0 to

equal to 1 while `X_nref` is greater than or equal to 5, the code assigns a value of 1 to

of 2 to `X(geneIndex,patIndex)`, and calculates the ratio of `X_nref(geneIndex,patIndex)` to `totRead`

sed.

opies of the data, and then performs data cleaning and normalization based on certain conditions.

ditional normalized data is stored in the variable `Xa`.



# EXTRACT CODE

Given the following MATLAB code, extract the two loops into a function taking two input matrices and returns two output matrices...

> Here's the extracted function from the given MATLAB code:

```
function [X, Xa] = cleanAndNormalize(X_ref, X_nref)
    X = zeros(size(X_ref));
    Xa = NaN(size(X_ref));
    for patIndex = 1 : length(X_ref(1,:))
        msg = sprintf( "Processing sample %d ", patIndex );
        disp(msg);
        for geneIndex = 1 : length(X_nref(:,1))
            totRead = X_ref(geneIndex,patIndex) + X_nref(geneIndex,patIndex);
            if totRead < 5
                X(geneIndex,patIndex) = 4;
                Xa(geneIndex,patIndex) = NaN;
            elseif (X_nref(geneIndex,patIndex) <= 1) && (X_ref(geneIndex,patIndex) >= 5)
                X(geneIndex,patIndex) = 0;
                Xa(geneIndex,patIndex) = 0;
            elseif (X_ref(geneIndex,patIndex) <= 1) && (X_nref(geneIndex,patIndex) >= 5)
                X(geneIndex,patIndex) = 1;
                Xa(geneIndex,patIndex) = 1;
            else
                X(geneIndex,patIndex) = 2;
                Xa(geneIndex,patIndex) = X_nref(geneIndex,patIndex) / totRead;
            end
        end
    end
end
end
```

# OPTIMIZE CODE

Please optimize the for loop in the previous code..

```
function [X, Xa] = cleanAndNormalize(X_ref, X_nref)
    % Calculate total read counts for each gene and sample
    totRead = X_ref + X_nref;

    % Initialize X and Xa matrices with default values
    X = repmat(4, size(X_ref));
    Xa = nan(size(X_ref));

    % Set conditions for data cleaning and normalization
    condition1 = totRead >= 5 & X_nref <= 1 & X_ref >= 5;
    condition2 = totRead >= 5 & X_ref <= 1 & X_nref >= 5;
    condition3 = ~condition1 & ~condition2;

    % Apply conditions to update X and Xa matrices
    X(condition1) = 0;
    Xa(condition1) = 0;

    X(condition2) = 1;
    Xa(condition2) = 1;

    X(condition3) = 2;
    Xa(condition3) = X_nref(condition3) ./ totRead(condition3);
end
```

> By utilizing element-wise operations and logical indexing, the optimized code avoids the need for explicit nested loops. Instead, it performs calculations on entire matrices at once, resulting in improved efficiency and faster execution.

# HMMM...

```
function [X, Xa] = cleanAndNormalize(X_ref, X_nref)
    X = zeros(size(X_ref));
    Xa = NaN(size(X_ref));
    for patIndex = 1 : length(X_ref(1,:))
        msg = sprintf( "Processing sample %d ", patIndex );
        disp(msg);
        for geneIndex = 1 : length(X_nref(:,1))
            totRead = X_ref(geneIndex,patIndex) +
                X_nref(geneIndex,patIndex);
            if totRead < 5
                X(geneIndex,patIndex) = 4;
                Xa(geneIndex,patIndex) = NaN;
            elseif (X_nref(geneIndex,patIndex) <= 1) &&
                (X_ref(geneIndex,patIndex) >= 5)
                X(geneIndex,patIndex) = 0;
                Xa(geneIndex,patIndex) = 0;
            elseif (X_ref(geneIndex,patIndex) <= 1) &&
                (X_nref(geneIndex,patIndex) >= 5)
                X(geneIndex,patIndex) = 1;
                Xa(geneIndex,patIndex) = 1;
            else
                X(geneIndex,patIndex) = 2;
                Xa(geneIndex,patIndex) = X_nref(geneIndex,patIndex)/totRead;
            end
        end
    end
end
```

```
function [X, Xa] = cleanAndNormalize(X_ref, X_nref)
    % Calculate total read counts for each gene and sample
    totRead = X_ref + X_nref;

    % Initialize X and Xa matrices with default values
    X = repmat(4, size(X_ref));
    Xa = nan(size(X_ref));

    % Set conditions for data cleaning and normalization
    condition1 = totRead >= 5 & X_nref <= 1 & X_ref >= 5;
    condition2 = totRead >= 5 & X_ref <= 1 & X_nref >= 5;
    condition3 = ~condition1 & ~condition2;

    % Apply conditions to update X and Xa matrices
    X(condition1) = 0;
    Xa(condition1) = 0;

    X(condition2) = 1;
    Xa(condition2) = 1;

    X(condition3) = 2;
    Xa(condition3) = X_nref(condition3) ./ totRead(condition3);
end
```

# TEST IT

Write a test function to load data/X\_ref.mat into a variable X\_ref and data/X\_nref.mat into a variable X\_nref and then compare the results of calling the function cleanAndNormalizeA() and the function cleanAndNormalizeB() with X\_ref and X\_nref.

```
function testCleanAndNormalize()
    % Load data/X_ref.mat and data/X_nref.mat
    load('data/X_ref.mat');
    X_ref = X;
    load('data/X_nref.mat');
    X_nref = X;

    % Call cleanAndNormalizeA and cleanAndNormalizeB
    [Xa_A, Xa_Aa] = cleanAndNormalizeA(X_ref, X_nref);
    [Xa_B, Xa_Ba] = cleanAndNormalizeB(X_ref, X_nref);

    % Compare the results
    areEqual_Xa = isequal(Xa_A, Xa_B);
    areEqual_Xa = isequal(Xa_Aa, Xa_Ba);

    if areEqual_Xa && areEqual_Xa
        disp('Both versions of cleanAndNormalize produce the same results.');
```

> In this test function, the X\_ref and X\_nref matrices are loaded from the respective files. Then, both versions of the cleanAndNormalize function are called with these matrices. The results of both versions are stored in separate variables. Finally, the function compares the results of Xa and Xa matrices from both versions to check if they are equal. If they are equal, it displays a message indicating that both versions produce the same results. Otherwise, it displays a message indicating that there is a difference between the results of the two versions.

# HELP ME FIX YOUR CODE...

**Me:** Write the MATLAB code to compare two matrices that might have NaN values in them.

**ChatGPT:** To compare two matrices that might have NaN values in them, you can use the `isequaln` function in MATLAB.

## MATLAB to Python

```
function [X, Xa] = cleanAndNormalize(X_ref, X_nref)
% Calculate total read counts for each gene and sample
totRead = X_ref + X_nref;

% Initialize X and Xa matrices with default values
X = repmat(4, size(X_ref));
Xa = nan(size(X_ref));

% Set conditions for data cleaning and normalization
condition1 = totRead >= 5 & X_nref <= 1 & X_ref >= 5;
condition2 = totRead >= 5 & X_ref <= 1 & X_nref >= 5;
condition3 = ~condition1 & ~condition2;

% Apply conditions to update X and Xa matrices
X(condition1) = 0;
Xa(condition1) = 0;

X(condition2) = 1;
Xa(condition2) = 1;

X(condition3) = 2;
Xa(condition3) = X_nref(condition3) ./ totRead(condition3);
end
```

```
import numpy as np
```

```
X_ref = np.load('../data/X_ref.npy')
X_nref = np.load('../data/X_nref.npy')
```

```
X_ref_copy = np.copy(X_ref)
X_nref_copy = np.copy(X_nref)
```

```
# Calculate total read counts for each gene and sample
totRead = X_ref_copy + X_nref_copy
```

```
# Initialize X and Xa arrays with default values
X = np.full_like(X_ref_copy, 4)
Xa = np.empty_like(X_ref_copy)
Xa[:] = np.nan
```

```
# Set conditions for data cleaning and normalization
condition1 = np.logical_and.reduce((totRead >= 5, X_nref_copy <= 1, X_ref_copy >= 5))
condition2 = np.logical_and.reduce((totRead >= 5, X_ref_copy <= 1, X_nref_copy >= 5))
condition3 = np.logical_not(np.logical_or(condition1, condition2))
```

```
# Apply conditions to update X and Xa arrays
X[condition1] = 0
Xa[condition1] = 0
```

```
X[condition2] = 1
Xa[condition2] = 1
```

```
X[condition3] = 2
Xa[condition3] = X_nref_copy[condition3] / totRead[condition3]
```

# CHATGPT ASSISTANCE

User: Write the code to extract variable "X" from a MATLAB .mat file using Python and the scipy package.

ChatGPT: Certainly! Here's an example code snippet that demonstrates how to extract the variable "X" from a MATLAB .mat file using Python and the scipy package:

```
import scipy.io  
  
# Load the MATLAB .mat file  
mat_data = scipy.io.loadmat('your_file.mat')  
  
# Extract the variable 'X'  
X = mat_data['X']
```

# CREATE REFERENCE DATA

```
load('../data/X_ref.mat');
X_ref = X;
load('../data/X_nref.mat');
X_nref = X;
clear X;

disp("Loaded data. Cleaning data...")

[X, Xa] = cleanAndNormalizeOpt(X_ref, X_nref);

disp("Cleaned data. Saving data...")

% Courtesy of ChatGPT
data = struct('X', X(1:1024,:), 'Xa', Xa(1:1024,:)); %
Create a struct to hold the variables
save('reference.mat', '-struct', 'data', '-v6'); % Save the variables to
a .mat file
```



# TEST AGAINST REFERENCE DATA

```
# Load reference data from MATLAB/Octave
reference = scipy.io.loadmat(SCRIPT_DIR / 'reference.mat')
X_test = reference['X']
Xa_test = reference['Xa']

rows = X_test.shape[0]
assert np.array_equal(X[:rows,:], X_test), "X != X_test"
assert np.array_equal(Xa[:rows,:], Xa_test), "Xa != Xa_test"
```

# FIX CONDITIONAL BUG

```
# condition3 = np.logical_not(np.logical_or(condition1, condition2))  
condition3 = (totRead >= 5) &  
               np.logical_not(np.logical_or(condition1, condition2))
```

LIVE DEMOS  
COPILOT  
S[E]IR MODEL





# RESOURCES

- [GitHub Copilot](#)  
[GitHub Copilot X](#)
- [ChatGPT](#) (free w/limitations)  
[ChatGPT+](#) (\$ for guaranteed access and features)
- [New AI coding features are coming to Google Colab \(blog.google\)](#)
- [GitHub Codespaces](#)