# PFN Summer Internship 2022
# Theme Assignments

*JE01. Development of framework and library for deploying deep learning models in real world*

## About the assignment

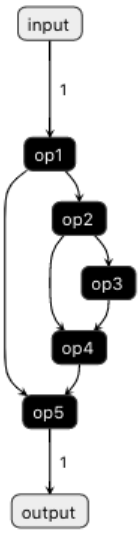
In principle, work on Assignment A for those who wish to participate in the PFVM development, or work on Assignment B for the CuPy development.

## Assignment A

### Task: Recomputation for Memory Reduction

Recomputation is known as a method to reduce the memory usage of neural networks. Literally, recomputation algorithms reduce memory by freeing variables that are not consumed soon after and re-compute them as needed in the future.
Generally, recomputation has a tradeoff between memory usage and computational time.

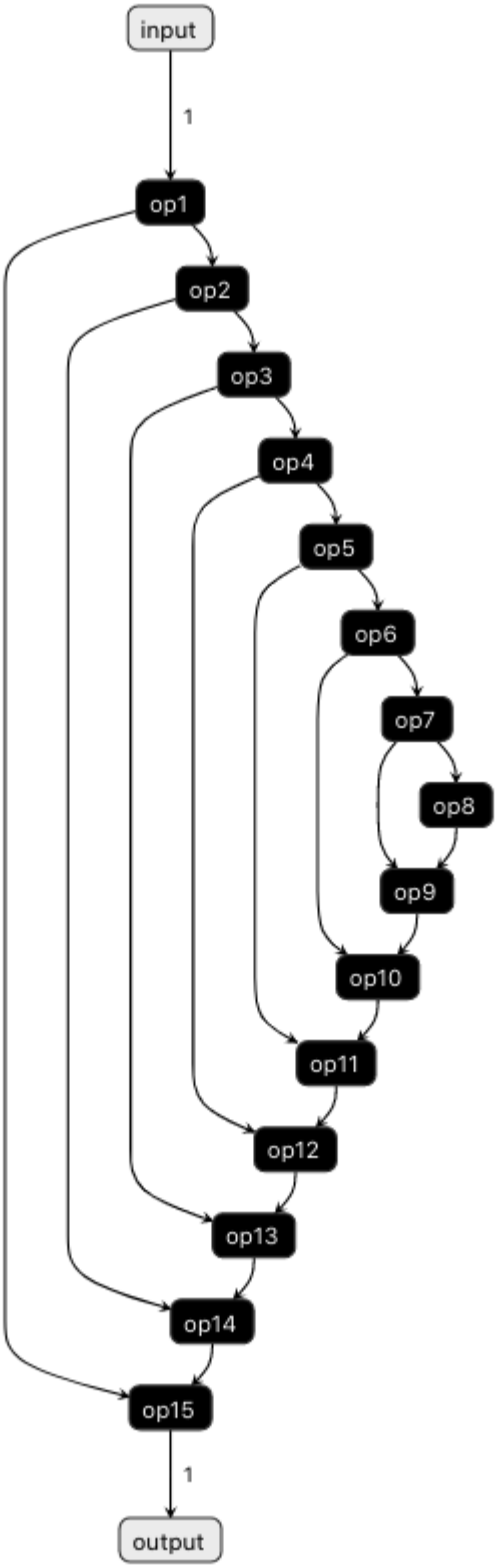
Consider the following example.  This computational graph has five operators: op1, …, op5. In this task, the number of outputs for each operator is always 1, and all the variables have a unit size.




When computing operators naively in the order [op1, op2, op3, op4, op5], provided that input is always on the memory, we need 5 units of memory when computing op4, because the outputs of op1 to op4 are all alive.
However, we can process this graph with at most 4 units of memory by computing in the order [op1, op2, op3, op4, op1, op5] (the output of op1 computed at first is freed after the computation of op2).

## Problem

For the below graph, consider a computational orders with recomputation. Also, report the tradeoffs compared to the naive execution.




You can refer to https://tech.preferred.jp/ja/blog/recomputation/ if necessary.

### Submission

Please include the following files in a single zip.

1. Programs used for experiments (if any)
2. A program that checks that your execution order is valid and outputs its memory usage.
3. A report of a few pages

The report should contain the following items:

- The graph execution order using recomputation, and its explanation.
- Tradeoffs compared to naive execution.
- Choose a paper on recomputation from the list below and provide a simple explanation of it.
    - Chen, Tianqi, et al. "Training deep nets with sublinear memory cost." arXiv preprint arXiv:1604.06174 (2016).
    - Kusumoto, Mitsuru, et al. "A graph theoretic framework of recomputation algorithms for memory-efficient backpropagation." Advances in Neural Information Processing Systems 32 (2019).
    - Kumar, Ravi, et al. "Efficient rematerialization for deep networks." Advances in Neural Information Processing Systems 32 (2019).

# Assignment B

Please select one of the following two tasks.

## Task 1: CUDA implementation of a modal value finding program

You are given a single unsigned array of 64-bit integers as input. Implement a CUDA program that outputs the minimum of the most frequently occurring values in the array. If needed, you can use any of the standard libraries in CUDA (including Thrust, CUB), but cannot use any other libraries.

### About Inputs

The size of the input array is 2^24. The input is generated by the following procedure:

1. Select 2^20 candidate values, may include duplicates, uniformly at random from the entire range of unsigned 64-bit integers.
2. Generate the array by selecting uniformly at random 2^24 times from the candidate values.

For example, a C++ implementation of the function to generate the input would look like this

```
void generate_testcase(uint64_t* a, int seed) {
  const int N = 1 << 24;
  const int M = 1 << 20;
  std::mt19937 mt(seed);
  std::uniform_int_distribution<uint64_t> dist1;
  std::vector<uint64_t> x(M);
  for (int i = 0; i < M; ++i) {
    x[i] = dist1(mt);
  }
  std::uniform_int_distribution<int> dist2(0, M - 1);
```

```
  for (size_t i = 0; i < N; ++i) {
    a[i] = x[dist2(mt)];
  }
}
```

### Submissions

Please include the following files in a single zip.

1. An implementation
    - A function that computes the modal value of the input array
    - A command line to compile
    - Tests to verify the validity of output values
    - Benchmarks to measure execution time per function call
    - Add the minimal required comment in the code for readability.
2. A report
    - The report should contain 2 pages of A4 size paper.
    - Please describe your idea and performance evaluation.

## Task 2: Function Transpiler from Python to C++

Implement a function in Python that parses a given Python function using the ast module and converts it to a C++ function. For example:

```
def func(a, n):
    i = 0
    out = 0.
    while i < n:
        out += a[i]
        i += 1
    return out

transpile(func, (list[float], int))
```

will be transpiled into the following C++ function:

```
double func(double *a, int n) {
  int i = 0;
  double out = 0;
  while (i < n) {
    out += a[i];
    i += 1;
  }
  return out;
}
```

The above output is just one example. The assignment doesn't require you to output the same exact code for the above input.

## Transpiler Specification

- Your transpiler should be able to parse Python functions including the following features.
    - Argument types are bool, int, float, list[int] or list[double].
    - Arithmetic operations (+, -, *, /, //, %), comparison operations (==, ! =, <, <=, <=, >, >=), boolean operations (not, and, or)
    - Assignments, "if" statements, "while" statements, and function calls (including recursive functions)
    - Minimal list operations: lookup and mutation
        - Don't need to consider list slicing or construction
- Do not use `` `auto` `` in the generated C++ functions.
- You can define other specifications not defined above.

## Submissions

Please include the following files in a single zip.

1. Transpiler implementation
    - Add the minimal required comment in the code for readability.
    - Include an example input that confirms the operation of the transpiler.
2. A report
    - The report should contain 2 pages of A4 size paper.
    - Please write about any additional specifications and your ideas.
    - If there are input examples that are difficult to be converted by the transpiler, or if you have ideas that could not be fully implemented, you can write about them.