Programming Assignment 2

Accelerating LLM via Multi-Threading

COMP3230 Principle of Operating System, Fall 2023, HKU

Agenda

1. Introduction
2. Task
3. Takeaway

Introduction

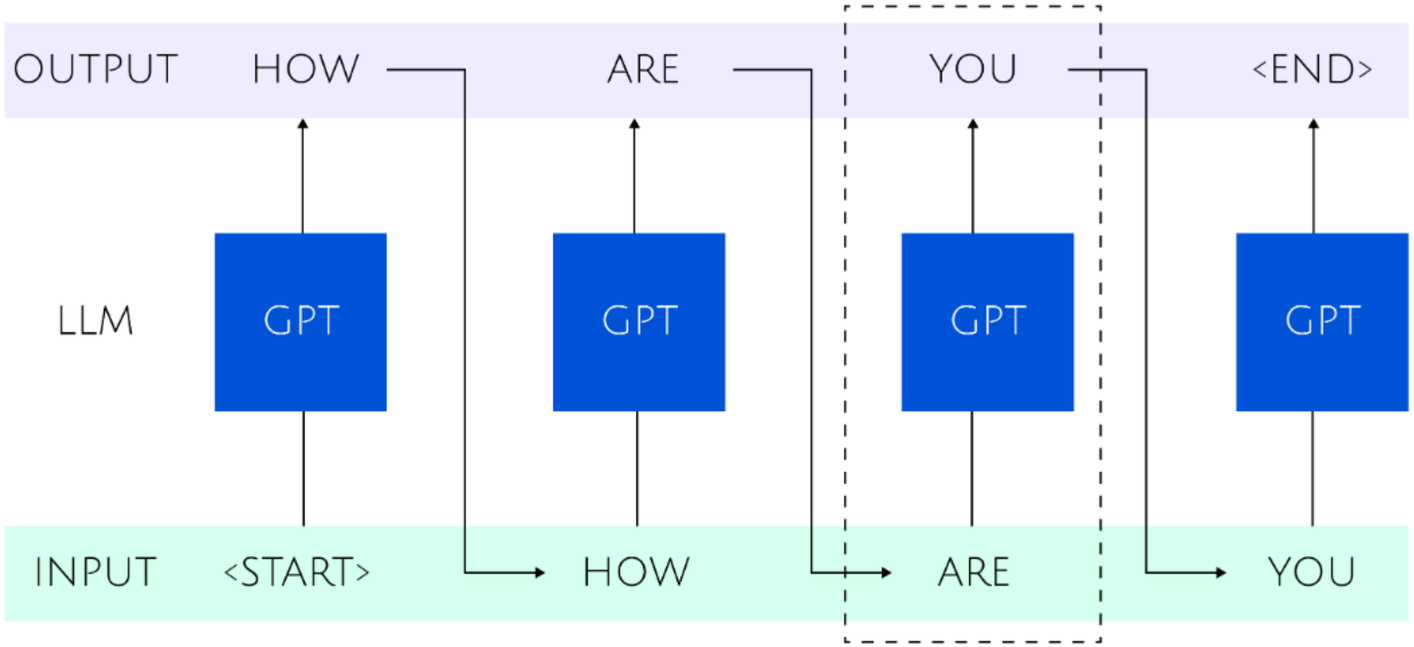
TL:DR; LLM use >50% time on **mat_vec_mul**

- High-Level
- Word to Vector
- Transformer
  - Attention - many **mat_vec_mul**
  - Feed-Forward Network - many **mat_vec_mul**
  - Positional Embedding
- Our LLM

High-Level

We speak **word by word**, so as LLM(GPT).

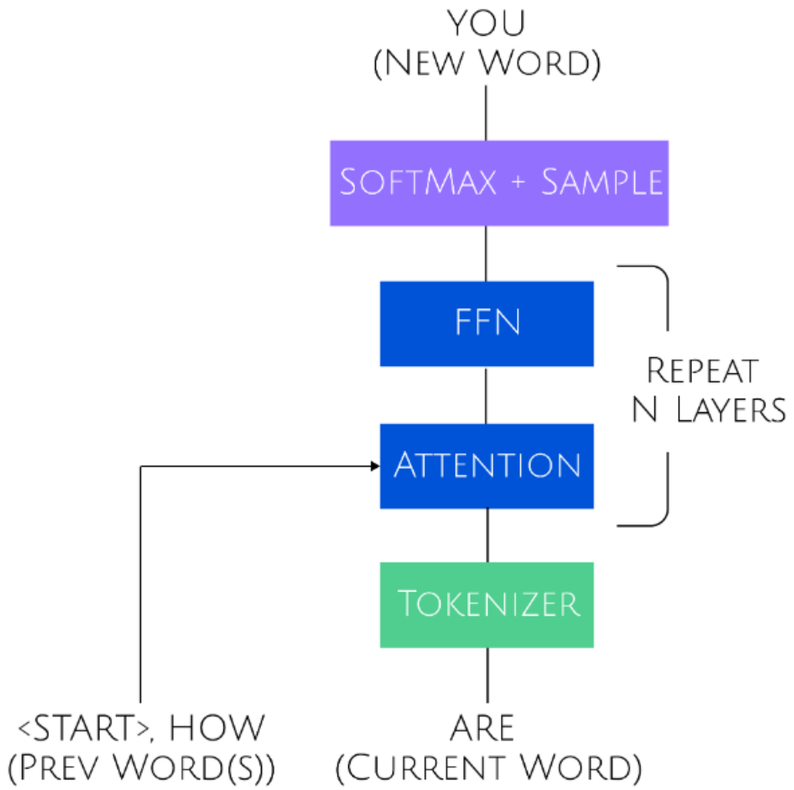
Large-Language Model is function to **generate a token**(word) **based on previous tokens** like <START>, and is **called repeatedly** until reaches <END> or max sequence length.



Instead of meaningless <START>, you can provide meaningful words, known as **prompts**.

GPT Insight

Natural Language consists of Strings but **Machine Learning runs on Numbers…**



Tokenizer

Give each word of dictionary a unique number!
For example, 1st word "a" is 0, 2nd word is 1…
tokenizer = {"a" : 1, "b" : 2} # read prompt
inv_tokenizer = {1: "a", 2: "b"} # print output
Fun fact: No.token is how GPT API charges you.

Then turn tokens into vectors, called embedding, to enable deep learning, presents like a table:
em_table = [vec1, …, vec_n] # tok→embedding
Many ways to train embedding, not our focus.

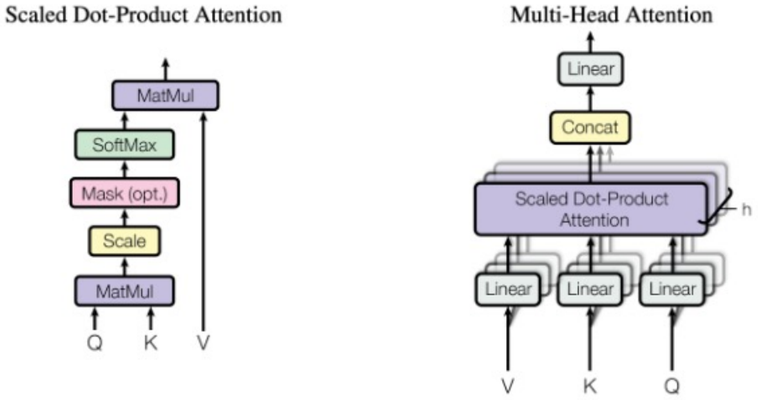
Don’t need to understand this page

Transformer - Computationally Intensive

Transformer transform the input token (and past tokens) to final prediction by:

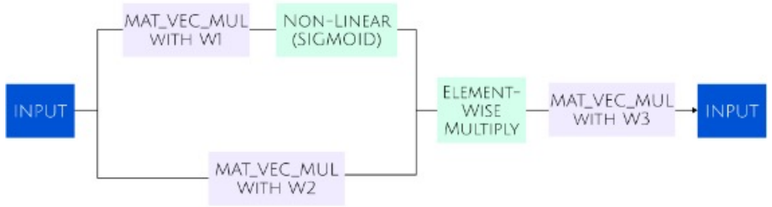
Attention

Simulate the human attention



Feed-Forward Network

Composition of mat_vec_mul and non-linear layers



Take Away - LLM has lots of Matrix-Vector-Multiplication

P.S. More details can be found in COMP3314 Machine Learning / COMP3361 Natural Language Processing

LLM in this assignment

Due to limited resources, we use a small but complete LLM - LLama2.c by Andrej Karpathy.

```
$ ./seq 42
 One day, a little girl named Lucy was in the park with her clumsy daughter. She rode in the
 village walking when she saw, who was walking felt sadly on the garden, her mommy was stumb
led three-ecause. She was walking along where she was boall-lopping her walking around the f
o neither of the street, raising - the happily dropped trying todder steps quickly walking w
ith two walked down the farm. Her name. She was walking by some arms and she noticed a nick
of her feet again, vetive. She saw. She was alone. Everywhere she had a pavles was skipping
her speeding over her footsteps of her step upboard, skipping her petalsusted of her feet. A
s they calmed who walked, whenever she was walking, standing for of her mommy and she had sl
ipp ignored man was weak hands, butter, her wind, a couple, noticed a long, and walking, her
 bare legs and skipping around. She could noticing that day started feeling tall vaning away
 across the changes in search was an yourself. All over the shaped of a younger, walking aro
und her footsteps carrying a brightly stepped for one dog ladybuging once she had

length: 256, time: 5.586000 s, achieved tok/s: 45.828858
main thread - user: 5.4787 s, system: 0.1601 s
```

In inference, it can generate some tiny (bed time) stories, but amazingly with (nearly) correct English grammar.
Our model will generate around 256 words in around 5 seconds on workbench2.

We have modified his code to help you concentrate on the task, mat_vec_mul.

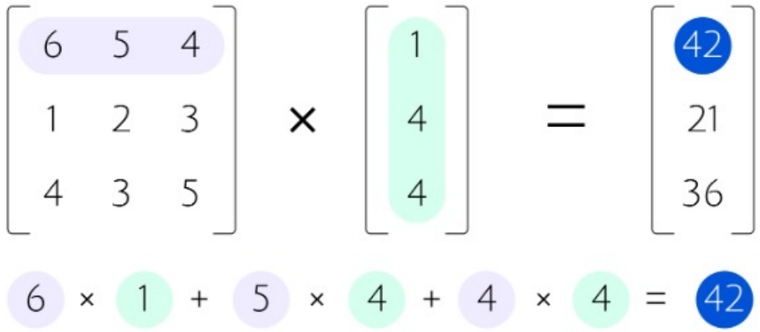
Task

Matrix-Vector-Multiplication

- Sequential Implementation
- Parallelism
- Reuse Thread

Matrix-Vector-Multiplication

Most fundamental algorithm in Linear Algebra, also called Linear/Fully-Connected Layer in ML:



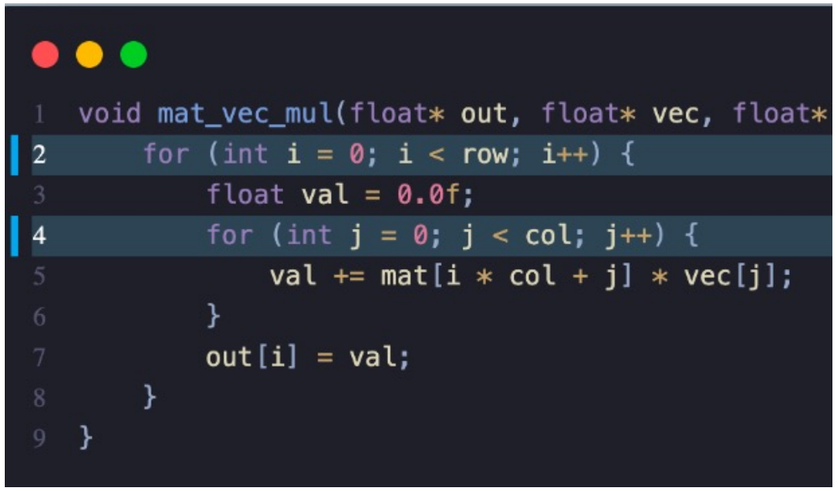

```c
void mat_vec_mul(float* out, float* vec, float* mat, int col, int row) {
    for (int i = 0; i < row; i++) {
        float val = 0.0f;
        for (int j = 0; j < col; j++) {
            val += mat[i * col + j] * vec[j];
        }
        out[i] = val;
    }
}
```

O(n²): for each row, sum the product of each column with corresponding vector elements.

In this assignment, you can assume No.column = Vector.Length → No checking need nor required.

Parallelism

Your task in this assignment is to accelerate the inference by multi-threading. As discussed in Tutorials, it means distributing iteration to different threads. But here we have two for-loop:



```
1  void mat_vec_mul(float* out, float* vec, float*
2      for (int i = 0; i < row; i++) {
3          float val = 0.0f;
4          for (int j = 0; j < col; j++) {
5              val += mat[i * col + j] * vec[j];
6          }
7          out[i] = val;
8      }
9  }
```

Question: Which Iteration to distribute and why?

1. Outer Loop in Line 2?
2. Inner Loop in Line 4?

Answer: Outer Loop! Because we can implement `val` as stack variables and make the whole computation lock-free.

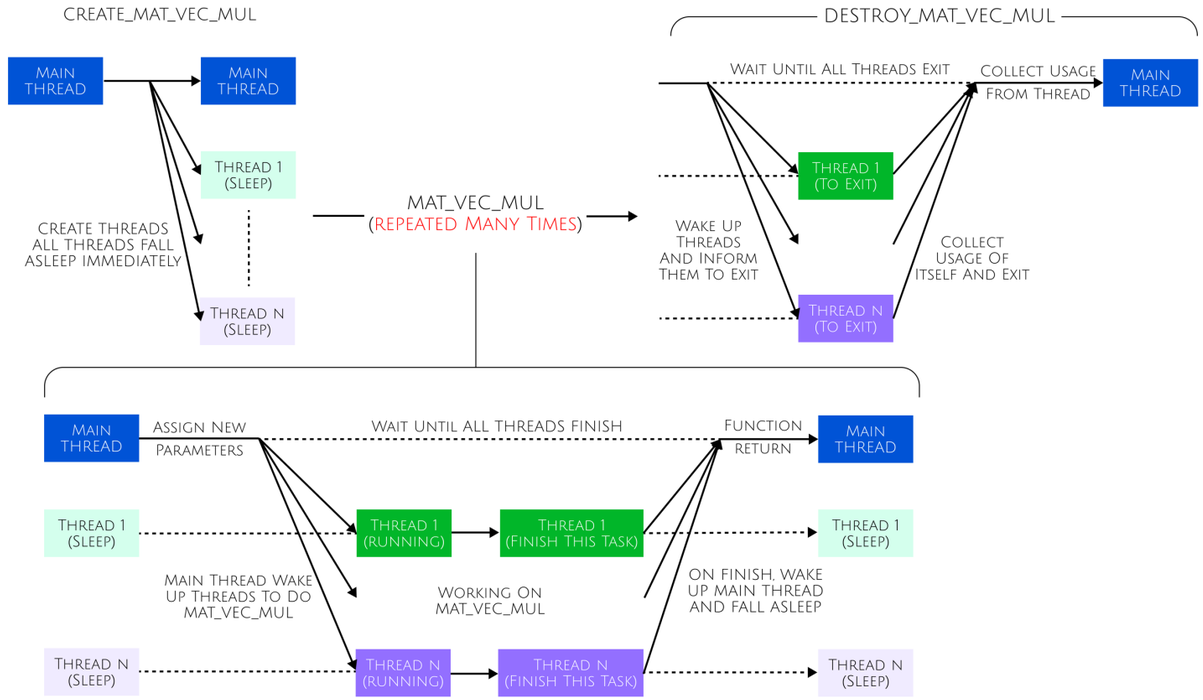
Reuse Threads

In **mat_vec_mul** we could:

1. Create *n* threads by **pthread_create**
2. Distribute workload
3. Wait for complete by **pthread_join**.

Question: **mat_vec_mul** will be called thousands times, how many threads in total?
Answer: O(n) and n is No. function calls to **mat_vec_mul**

Better way, and also a requirement for you is to reuse threads by synchronization like →



Thread Limits

👉 Workbench2 is the only server to use → No academy11 / academy21!!!

Workbench2 server has a limit for maximum **threads/processes** on user can create. Be careful with that number. Check that limit with `ulimit -u` (This year is 512)

To get the process created by you,
use `ps -fu [CSID]`

```
h3563791@workbench2:~$ ps -fu h3563791
UID          PID    PPID  C STIME TTY          TIME CMD
h3563791  226308       1  0 01:11 ?        00:00:00 /home/d001/h356
h3563791  226457       1  0 01:11 ?        00:00:00 /home/d001/h356
h3563791  229062       1  0 01:14 ?        00:00:01 /home/d001/h356
h3563791  233047 3191503  0 01:18 pts/164  00:00:00 /bin/bash
h3563791  233238 3191503  0 01:18 pts/165  00:00:00 /bin/bash
h3563791  233338 3191503  0 01:18 pts/162  00:00:00 /bin/bash
h3563791  233757 3191503  0 01:19 pts/170  00:00:00 /bin/bash
h3563791  252200  233047  0 01:37 pts/164  00:00:00 ps -fu h3563791
```

`ps -fu [CSID] | wc -l` to count number

To get the threads under specific process,
use `ps -T -p [PID]`

```
h3563791@workbench2:~$ ps -T -p 226308
    PID    SPID TTY          TIME CMD
 226308  226308 ?        00:00:00 cpptools-srv
 226308  226309 ?        00:00:00 cpptools-srv
 226308  226310 ?        00:00:00 cpptools-srv
 226308  226311 ?        00:00:00 cpptools-srv
 226308  226312 ?        00:00:00 cpptools-srv
 226308  226313 ?        00:00:00 cpptools-srv
```

`ps -T -p [PID] | wc -l` to count number

Please contact CS technical support if you lose control of your program!!!

Take Away

1. How Operating System supports LLM
    a. Parallelism utilising multi-threading to accelerate independent tasks
    b. Synchronization help reduce OS overhead
2. Fundamentals of Large Language Model
    a. Insights: Attention, Feedforward Network, Tokenizer, …
    b. Learned Parameters applied by Matrix-Vector-Multiplication