Final Project – Text Generation

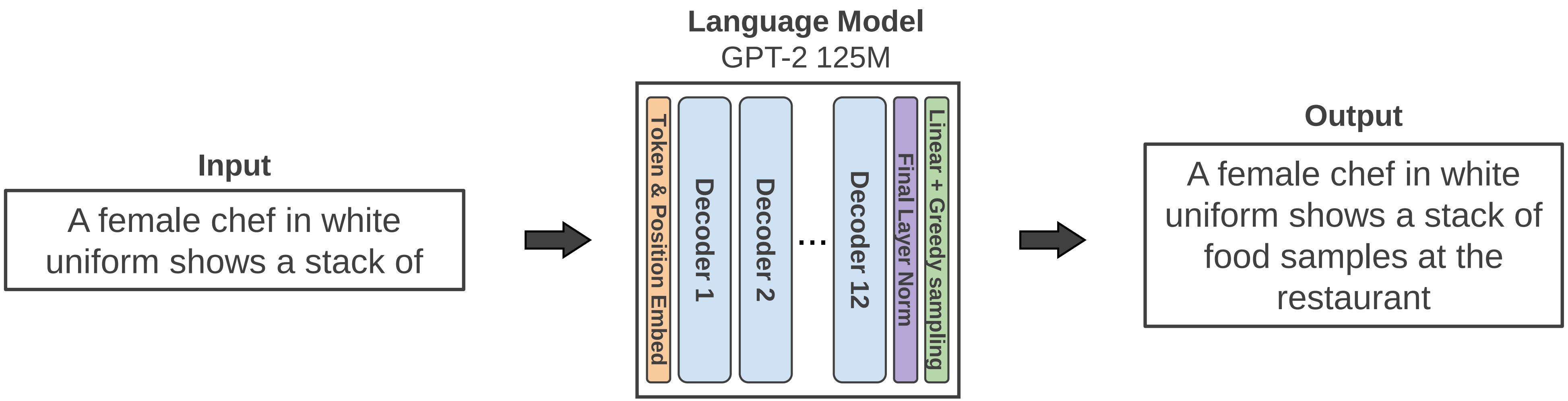
Project due: 2024. 06. 17. 11:59 PM

Last updated: 2024. 06. 10. 21:00 PM

Project Goal

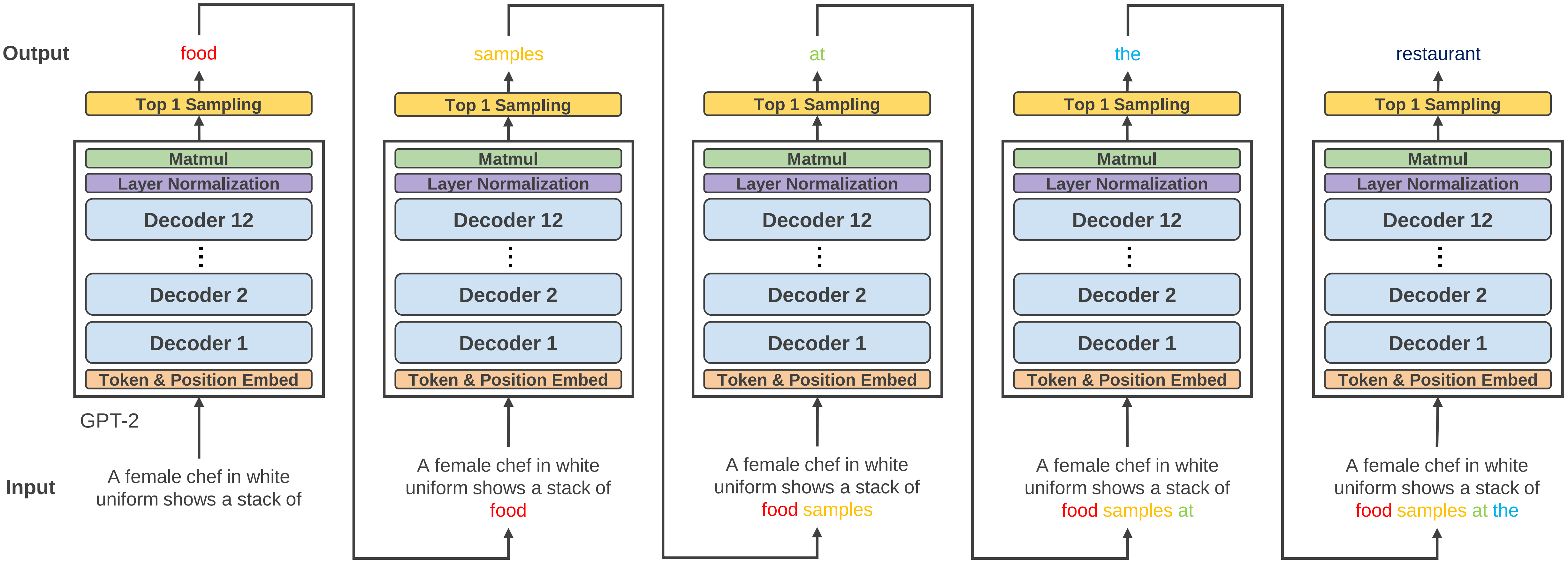
Optimize GPT-2 125M text generation

- You are given a sequential (CPU, single thread) code
- Parallelize and optimize the code across 4 nodes (16 GPUs in total)



Background and Skeleton Code

Text Generation using GPT-2



Skeleton Code

```
/shpc24/skeleton/final-project/
                    include/
                              tensor.h          # Tensor structure definition
                              layer.h           # Layer definitions
                              model.h           # GPT-2 model configuration and interfaces, do not modify
                    src/
                              tensor.cu         # Tensor structure implementation
                              layer.cu          # DNN layer implementation, where actual computations are done
                              model.cu          # GPT-2 model implementation using DNN layers
                              main.cpp          # Driver code, do not modify
                    data/                       # Directory to store input/output
                    Makefile                    # Makefile, do not modify
                    run.sh
```

How to Run (1)

Compile with `make` command

```
shpcta@login0:~/final-project$ make -j
mkdir -p obj
mpic++ -std=c++14 -O3 -Wall -march=native -mavx2 -fopenmp -I/usr/local/cuda/
/usr/local/cuda/bin/nvcc  -Xcompiler=-std=c++14  -Xcompiler=-O3  -Xcompiler=
iler=-Iinclude -c -o obj/model.o src/model.cu
/usr/local/cuda/bin/nvcc  -Xcompiler=-std=c++14  -Xcompiler=-O3  -Xcompiler=
iler=-Iinclude -c -o obj/tensor.o src/tensor.cu
/usr/local/cuda/bin/nvcc  -Xcompiler=-std=c++14  -Xcompiler=-O3  -Xcompiler=
iler=-Iinclude -c -o obj/layer.o src/layer.cu
cc -std=c++14 -O3 -Wall -march=native -mavx2 -fopenmp -I/usr/local/cuda/inc
art -lm -lmpi -lmpi_cxx
shpcta@login0:~/final-project$
```

How to Run (2)

Run with `run.sh` script

- Make sure to understand the script
- Check out the program options

```
shpcta@login0:~/chundoong-lab-ta/SHPC2024/final-project$ ./run.sh -h
salloc: Pending job allocation 527267
salloc: job 527267 queued and waiting for resources
salloc: job 527267 has been allocated resources
salloc: Granted job allocation 527267
 Usage: ./main [-i 'pth'] [-p 'pth'] [-o 'pth'] [-a 'pth'] [-t 'tokens'] [-n 'prompts'] [-v] [-h]
 Options:
  -i: Input binary path (default: data/input.bin)
  -p: Model parameter path (default: /shpc24/project_model_parameters.bin)
  -o: Output binary path (default: output.bin)
  -a: Answer binary path (default: data/answer.bin)
  -n: Number of input prompts (default: 1)
  -t: Number of tokens to generate (default: 8)
  -v: Enable validation (default: OFF)
  -h: Print manual and options (default: OFF)
salloc: Relinquishing job allocation 527267
shpcta@login0:~/chundoong-lab-ta/SHPC2024/final-project$
```

How to Run (3)

Run example

- 4 prompts
- Generate 8 tokens per prompt
- Validate output

```
shpcta@login0:~/final-project$ ./run.sh -v -n 4 -t 8
salloc: Pending job allocation 527342
salloc: job 527342 queued and waiting for resources
salloc: job 527342 has been allocated resources
salloc: Granted job allocation 527342

=============================================
 Model: GPT-2 125M
---------------------------------------------
 Validation: ON
 Number of Prompts: 4
 Number of Tokens to generate: 8
 Input binary path: ./data/input.bin
 Model parameter path: /shpc24/project_model_parameters.bin
 Answer binary path: ./data/answer.bin
 Output binary path: ./data/output.bin
=============================================

Initializing input and parameters...Done
Generating tokens...Done!
Elapsed time: 29.984716 (sec)
Throughput: 1.067210 (tokens/sec)
Finalizing...Done
Saving output to ./data/output.bin...Done
Validation...PASS
salloc: Relinquishing job allocation 527342
```

How to Run (4)

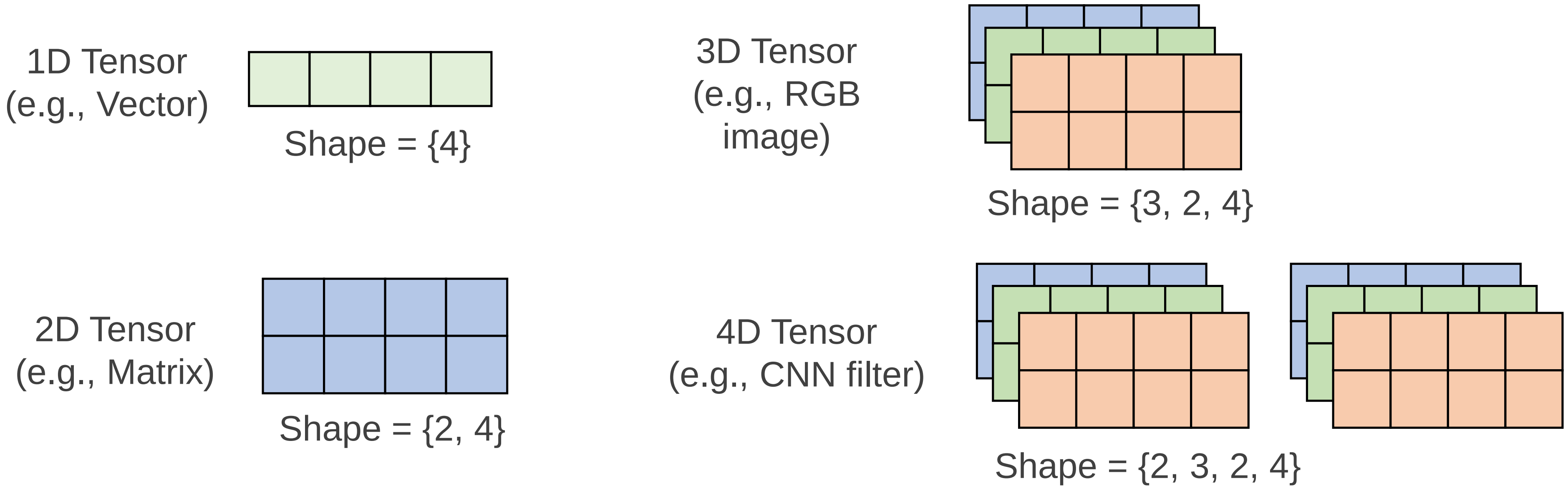
- Skeleton code uses tokenized texts (integers)
- Use `shpc24-project-bin2text` utility to see the actual text

```
shpcta@login0:~/final-project$ shpc24-bin2text
Usage: shpc24-bin2text <input_path> <output_path> <prompt_count (optional)>
 e.g., shpc24-bin2text ./data/input.bin ./data/output.bin 1
 <input_path>: Path to the input binary file
 <output_path>: Path to the output binary file
 <prompt_count>: Number of prompts to display (optional)
shpcta@login0:~/final-project$ shpc24-bin2text ./data/input.bin ./data/output.bin 4
Loading GPT-2 tokenizer...
Prompt # 1
 Input Prompt: Then, the man writes over the snow covering the window of a car, and
 Generated Output:  the woman, who is sitting on the

Prompt # 2
 Input Prompt: A female chef in white uniform shows a stack of baking pans in a large kitchen
 Generated Output: . She is wearing a white dress with

Prompt # 3
 Input Prompt: A female chef in white uniform shows a stack of baking pans in a large kitchen
 Generated Output: . She is wearing a white dress with

Prompt # 4
 Input Prompt: A tray of potatoes is loaded into the oven and removed. A large tray of
 Generated Output:  potatoes is placed on the stovetop and
```

Tensor

Data structure used in deep learning

- High-dimensional matrix
- Need to understand how the actual data is stored
- Defined in `include/tensor.h`, implemented in `src/tensor.cu`



Layers

Unit operations where the actual computations are done

- We use these layers to construct GPT-2 model
- Layers in GPT-2 are classified into four types
  1. Matrix multiplication
  2. Data movement
  3. Elementwise
  4. Other
- Defined in `include/layer.h`, implemented in `src/layer.cu`

Layers (cont'd)

Type 1: Matrix multiplication

1. Linear – `void linear(in, w, b, out)`
2. Matmul – `void matmul(in1, in2, out)`

Type 2: Data movement operations

1. Copy – `void copy(in, out)`
2. Transpose – `void transpose(in, out)`
3. Split QKV – `void split_qkv(in, out)`
4. Split Attention Heads – `void split_head(in, num_heads, out)`
5. Concat Attention Heads – `void concat_head(in, out)`
6. Extract QKV – `void extract_qkv(in, head_idx, num_heads, q, k, v)`
7. Merge Attention Heads – `void merge_head(in, head_idx, num_heads, out)`
8. Token Embeddig & Positional Embedding – `void token_pos_embedding(in, wte, wpe, out)`

Layers (cont'd)

Type 3: Elementwise Operations

1. GELU – `void gelu(inout)`
2. Add – `void add(inout, x)`
3. Scaling – `void scaling(inout, scale)`

Type 4: Other Operations

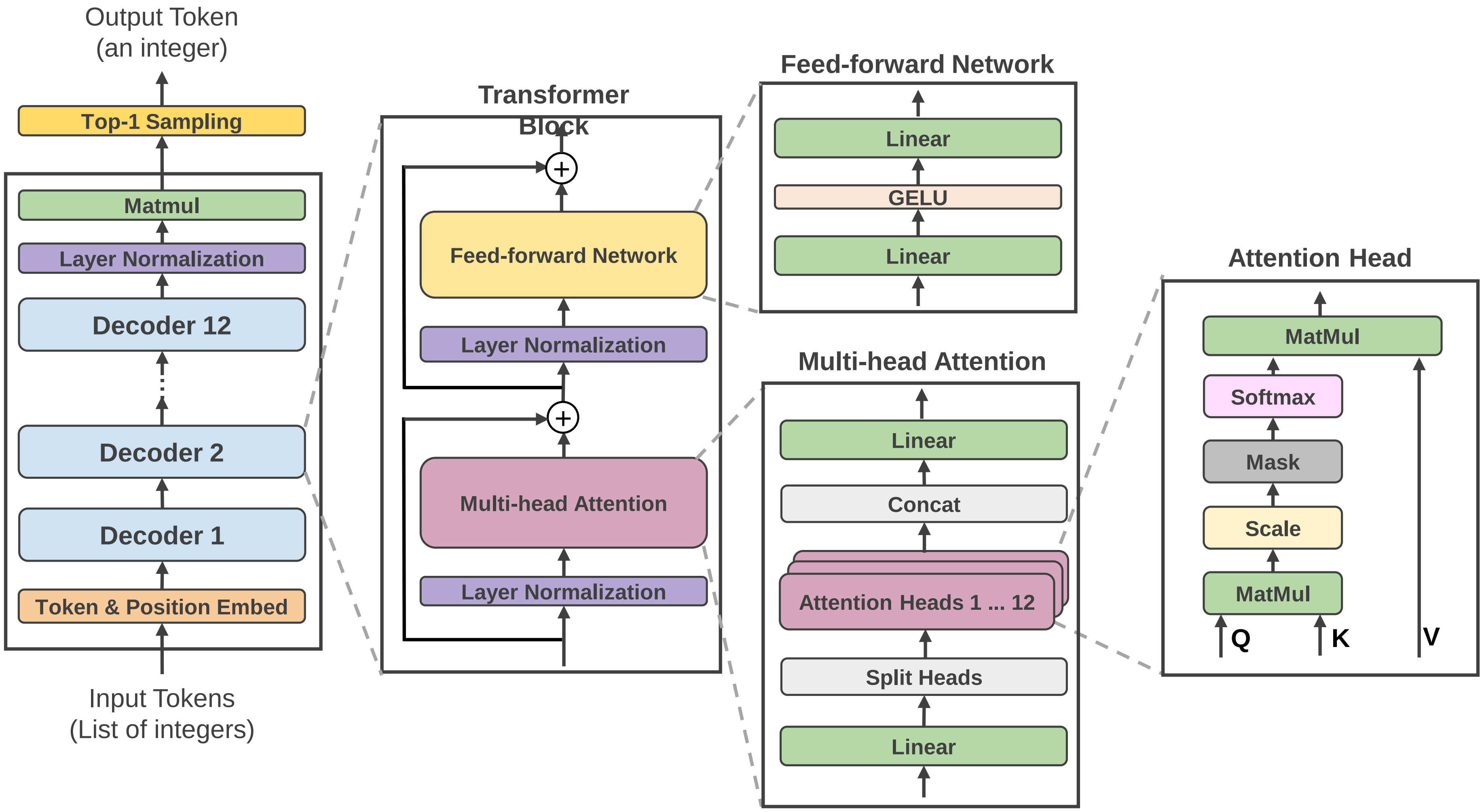
1. Softmax – `void softmax(inout)`
2. Layer Normalization – `void layer_norm(inout, gamma, beta)`
3. Mask Generation – `void generate_mask(inout)`
4. Top 1 Sampling – `int top1_sampling(in)`

Model

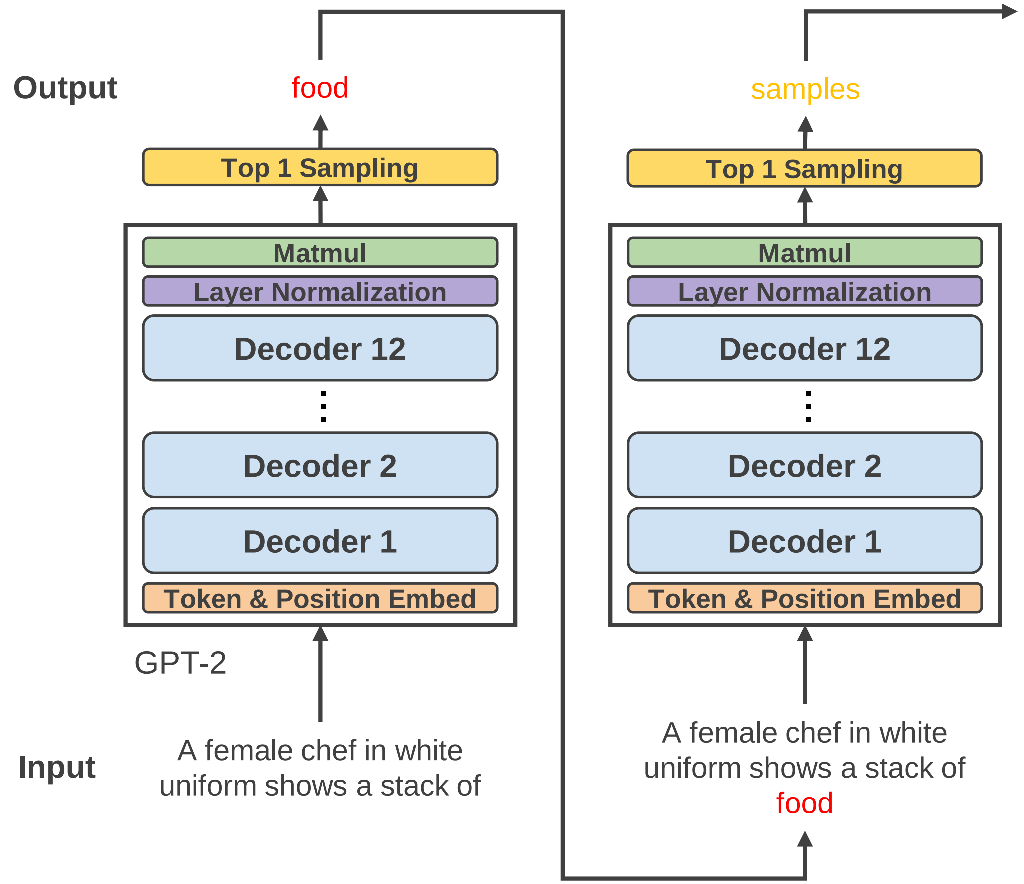
GPT-2 model architecture and interface are defined in `include/model.h`, implemented in `src/model.cu`

- `void alloc_and_set_parameters(float *param)`
  - Initialization of the model
  - Allocate memory for the model parameters, read the parameter file, and set
- `void generate_tokens(int *input, int *output, size_t n_prompt, size_t n_token)`
  - Main body of text generation using GPT-2
  - Our optimization target
  - Takes `n_prompt` prompts of the length 16, generate 8 tokens for each prompt
- `void free_parameters()`
  - Finalize the model
  - Free up the allocated memory for the model parameters

GPT-2 Model Architecture Breakdown

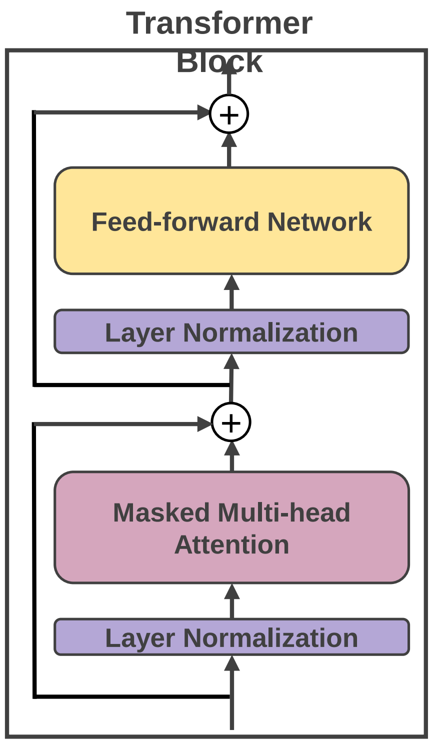


Model Architecture Breakdown – Text Generation Loop



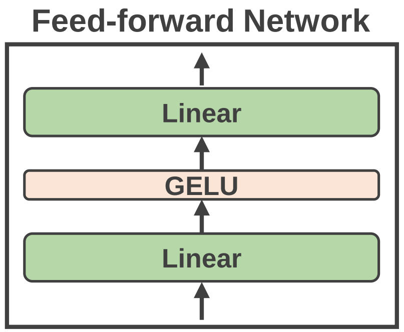

```cpp
void generate_tokens(int *input, int *output, size_t n_prompt, size_t n_token) {
  int mpi_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
  if (mpi_rank == 0) {
    int prompt_size = input_tokens;

    for (size_t p = 0; p < n_prompt; p++) {
      prompt_size = input_tokens;

      vector<int> input_prompt(prompt_size);
      memcpy(input_prompt.data(), input + p * prompt_size,
             prompt_size * sizeof(int));

      for (size_t t = 0; t < n_token; t++) {
        alloc_activations(prompt_size);

        token_pos_embedding(input_prompt, wte, wpe, embd_a);

        for (size_t l = 0; l < NUM_LAYER; l++) {
          transformer_block(embd_a, attn_b[l], attn_w[l], proj_b[l], proj_w[l],
                            ln_1_b[l], ln_1_g[l], ln_2_b[l], ln_2_g[l],
                            mlp1_b[l], mlp1_w[l], mlp2_b[l], mlp2_w[l],
                            transformer_block_a);

          copy(transformer_block_a, embd_a);
        }

        layer_norm(embd_a, ln_f_g, ln_f_b);

        transpose(wte, wte_transposed_a);
        matmul(embd_a, wte_transposed_a, logit_a);

        int next_token_id = top1_sampling(logit_a);

        input_prompt.push_back(next_token_id);
        prompt_size += 1;

        output[p * n_token + t] = next_token_id;

        free_activations();
      }
    }
  }
}
```

Model Architecture Breakdown – Transformer Block




```c
void transformer_block(Activation *in, Parameter *attn_b, Parameter *attn_w,
                       Parameter *proj_b, Parameter *proj_w, Parameter *ln_1_b,
                       Parameter *ln_1_g, Parameter *ln_2_b, Parameter *ln_2_g,
                       Parameter *mlp1_b, Parameter *mlp1_w, Parameter *mlp2_b,
                       Parameter *mlp2_w, Activation *out) {
  copy(in, residual_a);

  layer_norm(in, ln_1_g, ln_1_b);

  mha(in, attn_b, attn_w, proj_b, proj_w, mha_out_a);

  add(mha_out_a, residual_a);

  copy(mha_out_a, residual_a);

  layer_norm(mha_out_a, ln_2_g, ln_2_b);

  ffn(mha_out_a, mlp1_w, mlp1_b, mlp2_w, mlp2_b, out);

  add(out, residual_a);
}
```

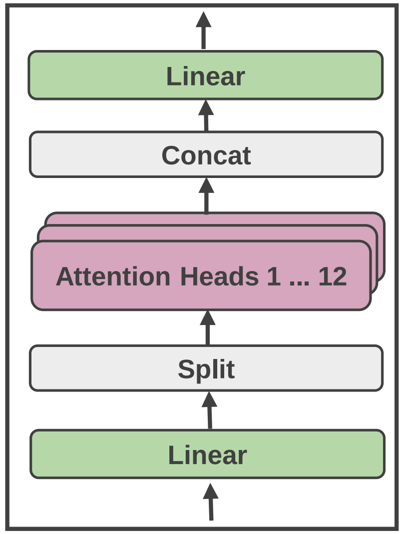
Model Architecture Breakdown – Feed-forward Network




```c
void ffn(Activation *in, Parameter *mlp1_w, Parameter *mlp1_b,
         Parameter *mlp2_w, Parameter *mlp2_b, Activation *out) {
  linear(in, mlp1_w, mlp1_b, ffn_proj_a);

  gelu(ffn_proj_a);

  linear(ffn_proj_a, mlp2_w, mlp2_b, out);
}
```

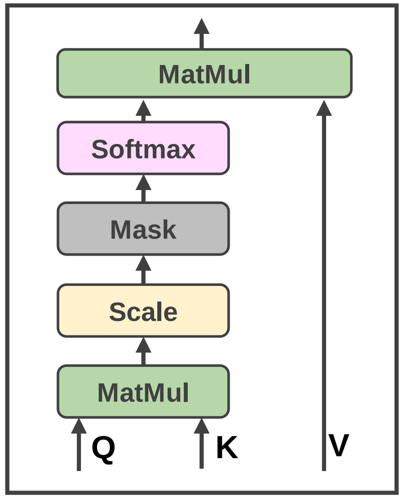
Model Architecture Breakdown – MHA and Attention Head

Multi-Head Attention




```
void mha(Activation *in, Parameter *attn_b, Parameter *attn_w,
         Parameter *proj_b, Parameter *proj_w, Activation *out) {
  linear(in, attn_w, attn_b, mha_qkv_proj_a);

  split_qkv(mha_qkv_proj_a, mha_split_qkv_a);

  split_head(mha_split_qkv_a, NUM_HEAD, mha_split_head_a);

  generate_mask(mha_mask_a);

  for (size_t idx = 0; idx < NUM_HEAD; idx++) {
    extract_qkv(mha_split_head_a, idx, NUM_HEAD, mha_q_a, mha_k_a, mha_v_a);
    attention(mha_q_a, mha_k_a, mha_v_a, mha_mask_a, mha_attn_out_a);
    merge_head(mha_attn_out_a, idx, NUM_HEAD, mha_merge_head_a);
  }

  concat_head(mha_merge_head_a, mha_concat_head_a);

  linear(mha_concat_head_a, proj_w, proj_b, out);
}
```

Attention Head




```
void attention(Activation *q, Activation *k, Activation *v, Activation *mask,
               Activation *out) {
  transpose(k, k_transposed_a);
  matmul(q, k_transposed_a, attn_score_a);

  scaling(attn_score_a, (1.0 / sqrt(k->shape[1])));

  add(attn_score_a, mask);

  softmax(attn_score_a);

  matmul(attn_score_a, v, out);
}
```

Project Grading

Grading

- **Performance (80%)** - Program throughput (generated tokens/sec)
  - Prompt (input sequence) length is fixed to 16 tokens. Generate 8 tokens for each prompt
  - You can choose the number of prompts. Maximum allowed is 8192
  - 1^{st} to 4^{th} place get full points, 10%p deduction for each time your code is 2x slower than 4^{th} place
  - Zero point if validation fails
- **Report (20%)**
  - Filename is `report.pdf`
  - Less than 5 pages, no restriction in format
  - Write concisely about your parallelization & optimization methods
  - **Make sure to include how to run your program and the expected performance result**

Restrictions

- Only the libraries we've covered in the class are allowed
  - x86 intrinsics, Pthread, OpenMP, OpenMPI, CUDA
- Any other external libraries are not allowed
  - cuBLAS, cuDNN, MAGMA, BLIS, PyTorch, Intel MKL, Tensorflow ...
- Any modification that changes the program logic is not allowed
  - You cannot
    - Do any computation in initialization phase
    - Use different model or text generation algorithm that makes the same output
    - ~~Skip some operations that does not affect the final output~~
  - You can
    - Change memory layout, change loop order, pad data, add some auxiliary operations, kernel fusion, ...
- Use eTL board if you have any questions

Submission

- Deadline: 2024. 06. 17. 11:59 PM
  - You cannot use grace days
  - This is the strict deadline, we can’t extend it anymore
- Submit the following files using `shpc-submit`
  - `tensor.h, tensor.cu, layer.h, layer.cu, model.cu, run.sh`
  - `report.pdf`
  - The other files will be overwritten by the skeleton code
- Start early!
  - The cluster server may be (will be) overloaded near the deadline
  - We can’t extend the deadline anymore

Comments

General Tips

- No need to understand the rationale behind the layers and model architecture. Just focus on the computation and data access patterns
- Whenever you modify something, make sure to check the correctness
  - It is very hard to debug when you make multiple parallelization/optimization at once
- Find out the problem **before** you do something
- Don’t go directly into famous LLM inference optimization techniques... do the easy things first
- The skeleton code does many things inefficiently. Find them out and get rid of it

Things to Try (When you don’t have any idea)

- Understand the skeleton code
- Parallelize the layers one-by-one, starting from the most time-consuming ones
  - There is an example CUDA kernel in the skeleton code. Check out `add_kernel()` and `add_cuda()` in `src/layer.cu`
- Minimize memory allocation and free
- Minimize data movement
- Batch the inputs

Things to Try (When you think there’s nothing to do)

- Profile your program
  - Try Nsight Systems profiler. We will cover this later in the class
  - Inspect the trace of your program. Find out any unexpected behavior
- Kernel optimization
  - How fast is your kernel compared to the peak FLOPS?
  - Try some kernel optimization techniques covered in the class
- Find out any redundant computations/data movement, and remove it

Updates - Code

[2024.05.09. 21:00] main.cpp 코드 수정 (eTL 공지)

[2024.05.12. 16:00] answer.bin 오류 수정, main.cpp 코드 수정 (fflush 추가)

[2024.06.10. 21:00] main.cpp 코드 수정 (validate 함수 수정 및 tolerance 값 상향 조정)

Updates - Restrictions

[2024.06.04. 12:00]

- ~~Init phase에서 새로운 메모리 할당(malloc)은 가능하지만 메모리 복사(memcpy)는 불가능~~
  - ~~E.g., input, output, activation 등을 미리 malloc 가능~~
  - ~~E.g., input의 값을 다른 MPI rank로 memcpy (broadcast, scatter 등)는 불가능~~
- Batch size는 본인이 원하는 값으로 설정 가능 (레포트에 본인이 설정한 batch size 명시)
- Input prompt 개수(-n)은 임의로 선택 가능 (레포트에 본인이 선택한 n의 값 명시)
- FP16 사용 불가능 (Tensor Core 쓰지 말 것)

Updates - Restrictions (cont'd)

[2024.06.05. 14:00]

- (p.22 restriction 수정사항) 최종 결과가 같다면 불필요한 연산은 생략 가능
  - E.g., LM head의 linear 연산 중 일부
- (p.29 restriction 수정사항) Init phase에서 알 수 있는 정보로 할 수 있는 작업은 모두 허용