# Distributed GPT2 Model Project

Abhijeet Sahdev, Andrew Mathews, Felix Guerrero

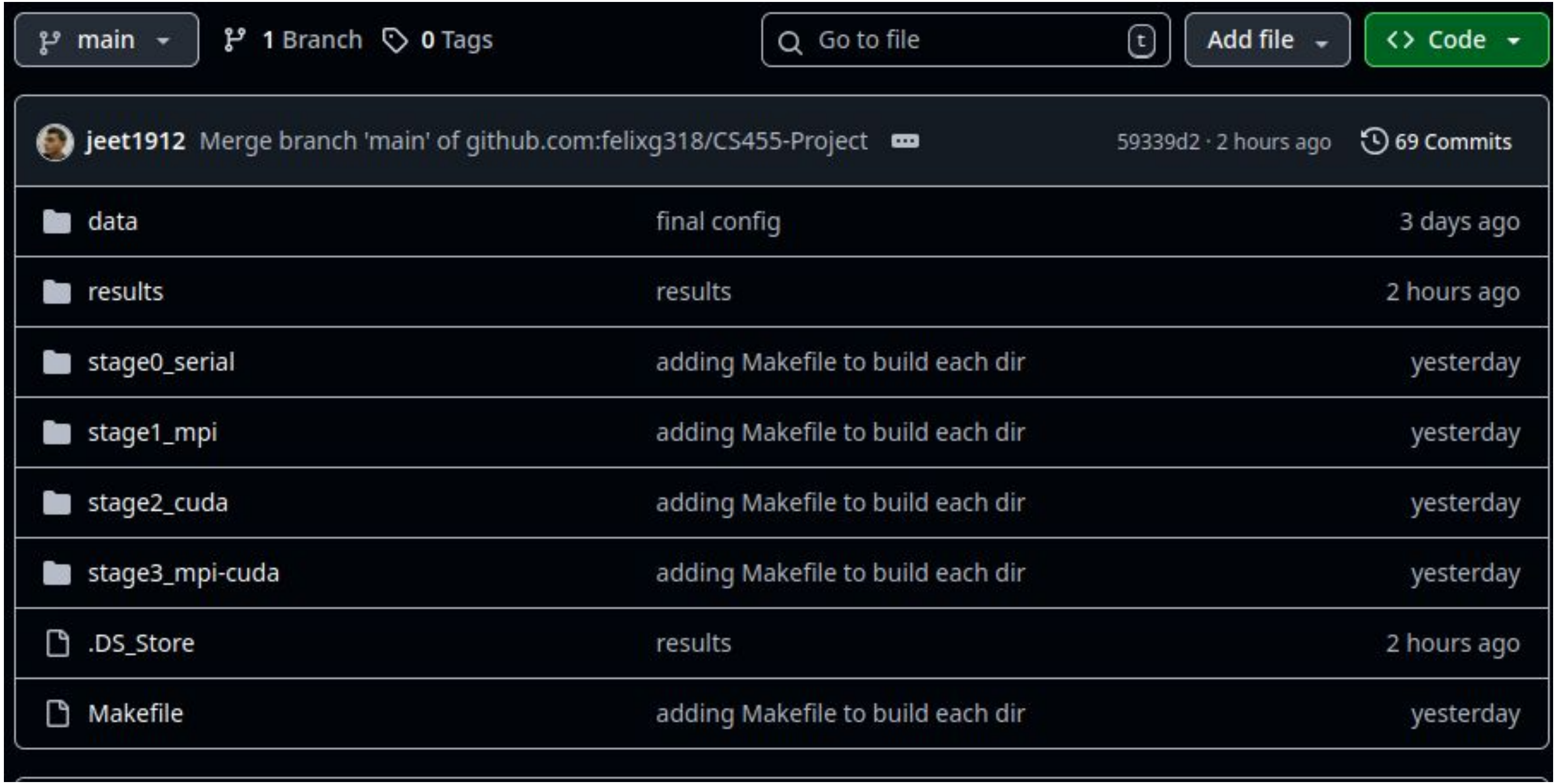
# Rationale

- AI is the biggest topic in computing over the last few years
- GPU parallelism allows thousands of operations to run at once
- Training AI can take a really long time on a single machine
- Using MPI and/or CUDA can dramatically reduce the time required for training an AI model
- This project would help us understand how to train an AI model

# Methodology

- We briefly went over:
  - Pytorch whitepaper and documentation
    - https://arxiv.org/pdf/1912.01703)
    - Simple to use deep learning framework
  - Build a Large Language Model (From Scratch) by Sebastian Raschka
  - Let's build GPT: from scratch, in code, spelled out. A youtube video by Andrej Karpathy
- Our C implementation is heavily inspired by Pytorch object-oriented approach

# Codebase


main
1 Branch
0 Tags
Go to file
t
Add file
Code
jeet1912 Merge branch 'main' of github.com:felixg318/CS455-Project
59339d2 · 2 hours ago
69 Commits
data
final config
3 days ago
results
results
2 hours ago
stage0_serial
adding Makefile to build each dir
yesterday
stage1_mpi
adding Makefile to build each dir
yesterday
stage2_cuda
adding Makefile to build each dir
yesterday
stage3_mpi-cuda
adding Makefile to build each dir
yesterday
.DS_Store
results
2 hours ago
Makefile
adding Makefile to build each dir
yesterday

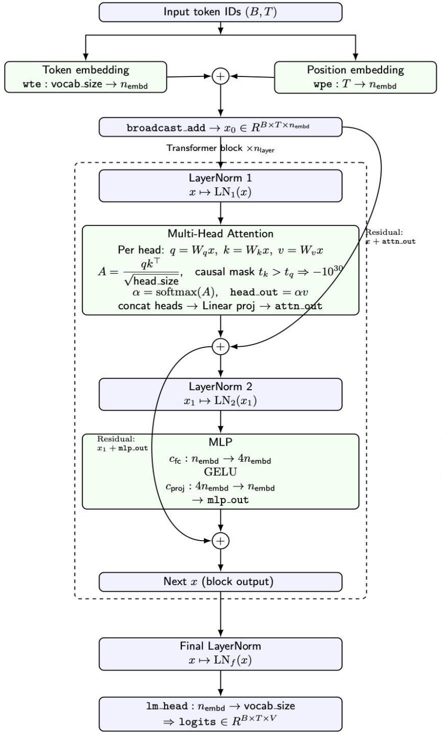
- Every stage has the following files:

Makefile
README.md
adam.h
add.h
autograd.h
block.h
broadcast.h
checkpoint.h
cross_entropy.h
cuda_kernels.cu
cuda_utils.h
dataloader.h
embedding.h
gelu.h
gpt.h
head.h
inference.cpp
layernorm.h
linear.h
matmul.h
mlp.h
multihead_attention.h
run_inference.sh
softmax.h
tensor.h
tokenizer.h
train_gpt2.cpp
trained_weights.bin
transpose.h

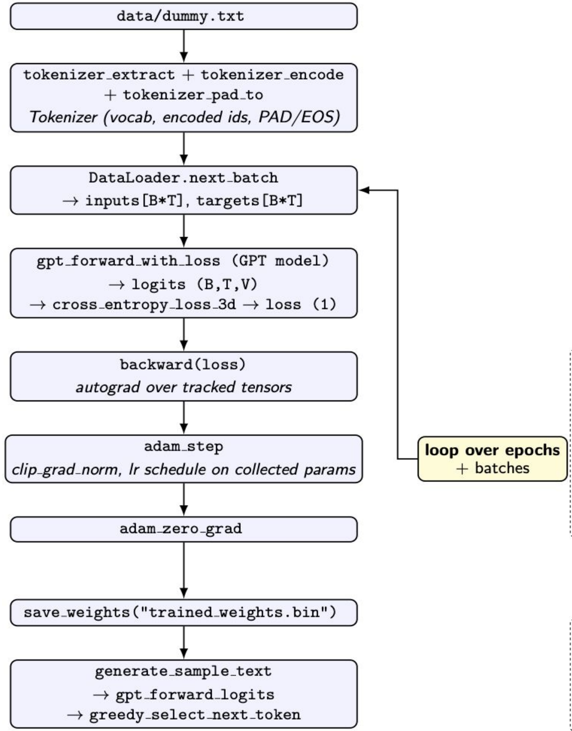
# Differences Between Versions

- **Serial**: Embeddings, transformer blocks, output head
- **MPI**: Batch sharing, parameter sync metadata, distributed generation, gpt_set_distributed
- **CUDA**: Core operations offloaded to kernels, no tokenizer/dataloader broadcast, same model graph
- **MPI + CUDA**: Rank 0 builds tokenizer and broadcasts parameters, coordinated distributed settings across processes and cores, end-to-end flow similar to CUDA version, but with mandatory data-parallel synchronization

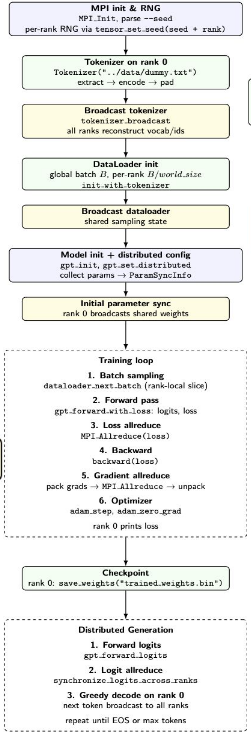
# Figures



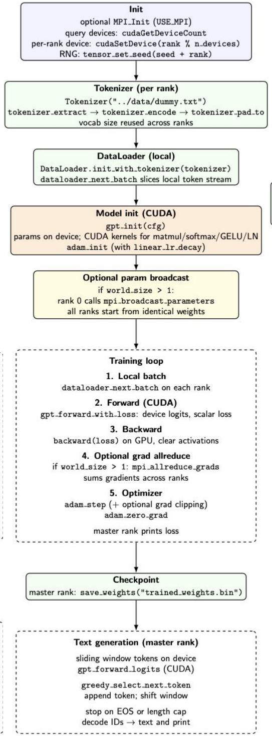

Model Architecture



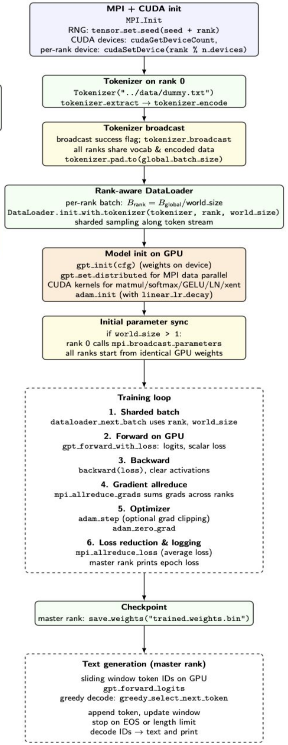

Stage 0




Stage 1




Stage 2




Stage 3

# Hyperparameters

**GPT model hyperparameters:**

- Block size: 256
- Layers: 6
- Heads: 6
- Embedding size: 384
- Dropout: *N/A*
- Learning rate: 3e-4f

**Data-loading stats:**

- Batch size: 16
- Sequence length: 256
- Epochs: 50

**Toy dataset stats:**

- 290,599 bytes
- Extracted 52,600 tokens
- Vocab size: 172

*Note: the dataset is small and highly repetitive

mpi@dhcp-10-196-106-63:~/Public/stage1_mpi$ cd ..
mpi@dhcp-10-196-106-63:~/Public$ mpirun -np 8 -f hostfile ./train_gpt2
ssh_askpass: exec(/usr/bin/ksshaskpass): No such file or directory
Host key verification failed.
[mpiexec@dhcp-10-196-106-63] ui_cmd_cb (mpiexec/pmiserv_pmci.c:51): Launch proxy failed.
[mpiexec@dhcp-10-196-106-63] HYDT_dmxu_poll_wait_for_event (lib/tools/demux/demux_poll.c:76): callback returned error status
[mpiexec@dhcp-10-196-106-63] HYD_pmci_wait_for_completion (mpiexec/pmiserv_pmci.c:173): error waiting for event
[mpiexec@dhcp-10-196-106-63] main (mpiexec/mpiexec.c:260): process manager error waiting for completion
mpi@dhcp-10-196-106-63:~/Public$ mpirun -np 8 -f hostfile ./train_gpt2
Training tokens (MPI): per epoch=16384, total=819200 (world_size=8)
Epoch 0, Loss: 9.569409
Epoch 1, Loss: 9.338302
top - 16:38:57 up 1 day, 44 min, 3 users, load average: 4.41, 3.93, 2.48
Tasks: 342 total, 5 running, 337 sleeping, 0 stopped, 0 zombie
MiB Mem : 31621.9 total, 21244.2 free, 8487.1 used, 2980.2 buff/cache
MiB Swap: 8192.0 total, 8192.0 free, 0.0 used. 23134.8 avail Mem
DELL
DELL

# Training Results

- We ran host MPI with 4 processes
    - Why? The Intel(R) Core(TM) i7-7820HQ CPU only has 4 cores (2 threads per core)
- All versions used the same hyperparameters

| Versions | Training Time |
| --- | ---: |
| Serial | 15264s (254.4 minutes) |
| MPI (Two nodes) | 1287s (21.5 minutes) |
| CUDA | 429s (7.2 minutes) |
| MPI+CUDA (Two nodes) | 226s (3.8 minutes) |

# Inference Results

| Versions | Inference Time |
| --- | --- |
| Serial | 113s (1.9 minutes) |
| MPI (Two nodes) | 112s (1.9 minutes) |
| CUDA | 8.0s (0.13 minutes) |
| MPI+CUDA (Two nodes) | 7.8s (0.13 minutes) |

# Conclusions

- We found significant speedup,
  - The MPI implementation yielded an immense speedup when utilizing both the CPU and CUDA (GPU) resources.
- You should,
  - Keep CUDA kernels simple
  - Minimize unnecessary data transfers between the host and the device
- Avoid,
  - Blocking calls for synchronization - can cause deadlocks or unnecessary waiting
  - Sending many small chunks of data rather than large chunks for communication calls (MPI)

# Future Direction

- Add NCCL (NVIDIA Collective Communications Library)
  - The standard for distributed training and inference across multiple GPUs.
  - Combines MPI and CUDA routines into a single framework.
  - Used to train actual AI models.
- Can also increase:
  - Hyperparameters
  - # of processes and/or machines
  - Data size