Run ChatGLM-6B

Finetune your ChatGLM from scratch!

TOC

- GLM
- Fintune
  - Prerequisite: Mixed Precision, ZeRO
  - P-tuning
  - Full Parameter
  - LoRA
- Deploy with Gradio

GLM: Pretraining

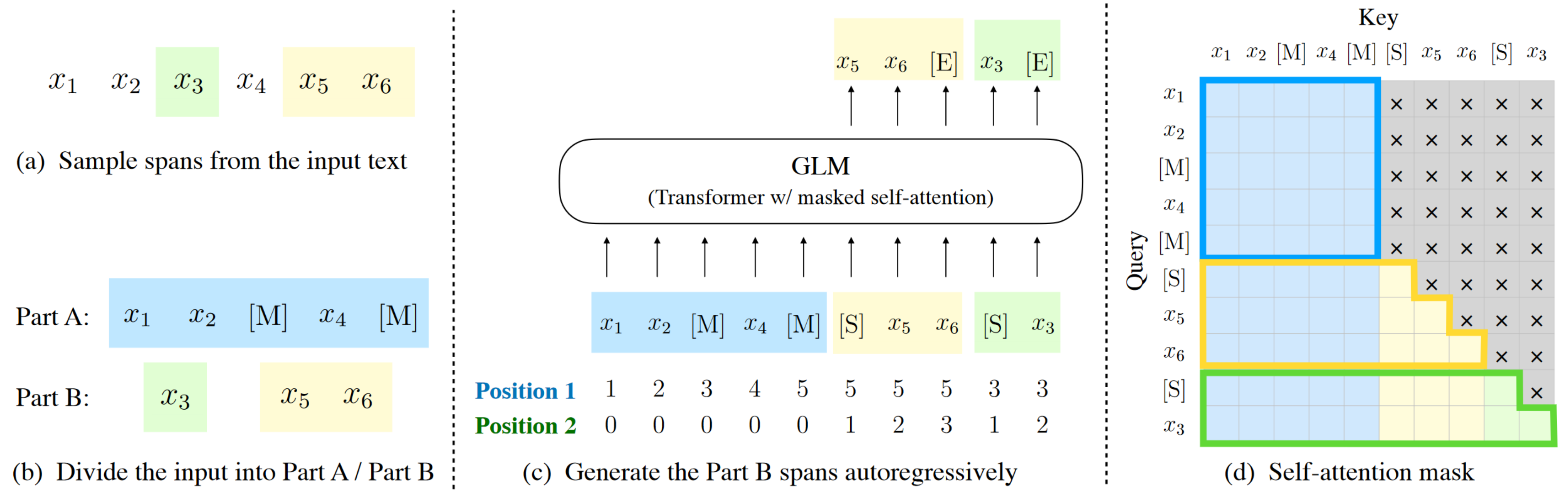


Figure 2: GLM pretraining. (a) The original text is $[x_1, x_2, x_3, x_4, x_5, x_6]$. Two spans $[x_3]$ and $[x_5, x_6]$ are sampled. (b) Replace the sampled spans with [M] in Part A, and shuffle the spans in Part B. (c) GLM autoregressively generates Part B. Each span is prepended with [S] as input and appended with [E] as output. 2D positional encoding represents inter- and intra-span positions. (d) Self-attention mask. Grey areas are masked out. Part A tokens can attend to themselves (blue frame) but not B. Part B tokens can attend to A and their antecedents in B (yellow and green frames correspond to the two spans). [M] := [MASK], [S] := [START], and [E] := [END].

OpenSource GLM Series

- GLM Github Paper
- GLM-130B Github Paper
- **ChatGLM-6B** Github Blog
  - can be finetuned on consumer-grade GPUs

Demo

- Download ChatGLM-6B checkpoints
- Inference with ChatGLM-6B
- Finetuning
  - P-Tuning (1 $\times$ RTX3090 !)
  - LoRA (1 $\times$ RTX3090 !)
  - Full Parameter

Demo environemnt

- GPU: NVIDIA GeForce RTX 3090
  - **This is not a must**. **7GB** is sufficient for P-tuning + 4-bit quantization
- Image: nvidia-pytorch:22.08-py3
- Change your pip source

```
pip config set global.extra-index-url https://pypi.tuna.tsinghua.edu.cn/simple
# Writing to /opt/conda/pip.conf
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
# Writing to /opt/conda/pip.conf
pip config set global.trusted-host https://pypi.tuna.tsinghua.edu.cn/simple
# Writing to /opt/conda/pip.conf
```

Download Checkpoint

Option1: From HuggingFace Repo

- Step 1: Install `git-lfs`, Get Started
  - Verify installation

```
git lfs install
# > Git LFS initialized.
```

- Step 2: Setup a ...

- Step 3: clone the repo

```
git clone https://huggingface.co/THUDM/chatglm-6b
# Cloning into 'chatglm-6b'...
# remote: Enumerating objects: 522, done.
# remote: Counting objects: 100% (522/522), done.
# remote: Compressing objects: 100% (495/495), done.
# remote: Total 522 (delta 321), reused 54 (delta 27), pack-reused 0
# Receiving objects: 100% (522/522), 158.52 KiB | 823.00 KiB/s, done.
# Resolving deltas: 100% (321/321), done.
```

- Seems to stuck here is expected behaviour
  - It's downloading the checkpoint ...
  - Use `bwm-ng` to monitor network traffic

Option 2: Downloading Manually

Useful when downloading from huggingface repo is slow

- Step 1: clone the repo, skip large files

```
GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/THUDM/chatglm-6b
# Cloning into 'chatglm-6b'...
# remote: Enumerating objects: 522, done.
# remote: Counting objects: 100% (522/522), done.
# remote: Compressing objects: 100% (495/495), done.
# remote: Total 522 (delta 321), reused 54 (delta 27), pack-reused 0
# Receiving objects: 100% (522/522), 159.22 KiB | 1.37 MiB/s, done.
# Resolving deltas: 100% (321/321), done.
```

- Step 2: Download large files from Tsinghua Cloud
    - download one by one is painful ...

```
git clone git@github.com:chenyifanthu/THU-Cloud-Downloader.git
cd THU-Cloud-Downloader
pip install argparse requests tqdm
python main.py \
    --link https://cloud.tsinghua.edu.cn/d/fb9f16d6dc8f482596c2/ \
    --save ../chatglm-6b/
# Start downloading? [y/n] y
# [1/11] Downloading File: ../chatglm-6b/LICENSE
# 100%|██████████| 11.1k/11.1k [00:00<00:00, 316kiB/s]
```

Clone Source Code

```
git clone git@github.com:THUDM/ChatGLM-6B.git
```

- Install dependencies
  1. Install `torch>=1.10` manually according to your CUDA Version
     - See Previous Versions
  2. Run

     ```
     pip install -r requirements.txt
     ```

Play with ChatGLM-6B in CLI

- Specify model path

```
# cli_demo.py
tokenizer = AutoTokenizer\
    .from_pretrained("THUDM/chatglm-6b", trust_remote_code=True)
model = AutoModel\
    .from_pretrained("THUDM/chatglm-6b", trust_remote_code=True)\
    .half().cuda()
```

- Run

```
python cli_demo.py
```

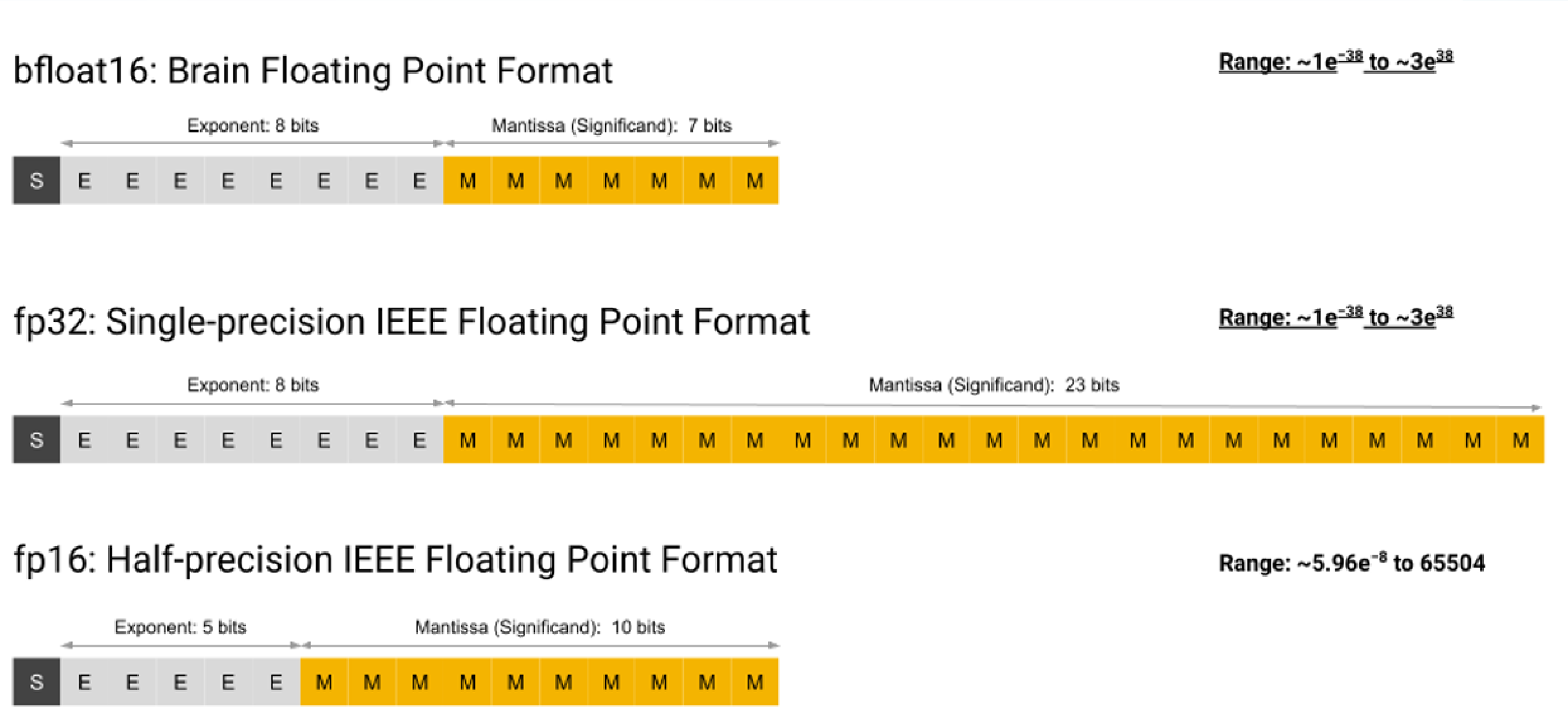
Play with ChatGLM-6B in Gradio

- Specify model path
- Run

```
python web_demo.py
```

- Interact with ChatGLM-6B in a browser 🐱
- VSCode port forwarding can be useful

Fine-tuning: Mixed Precision

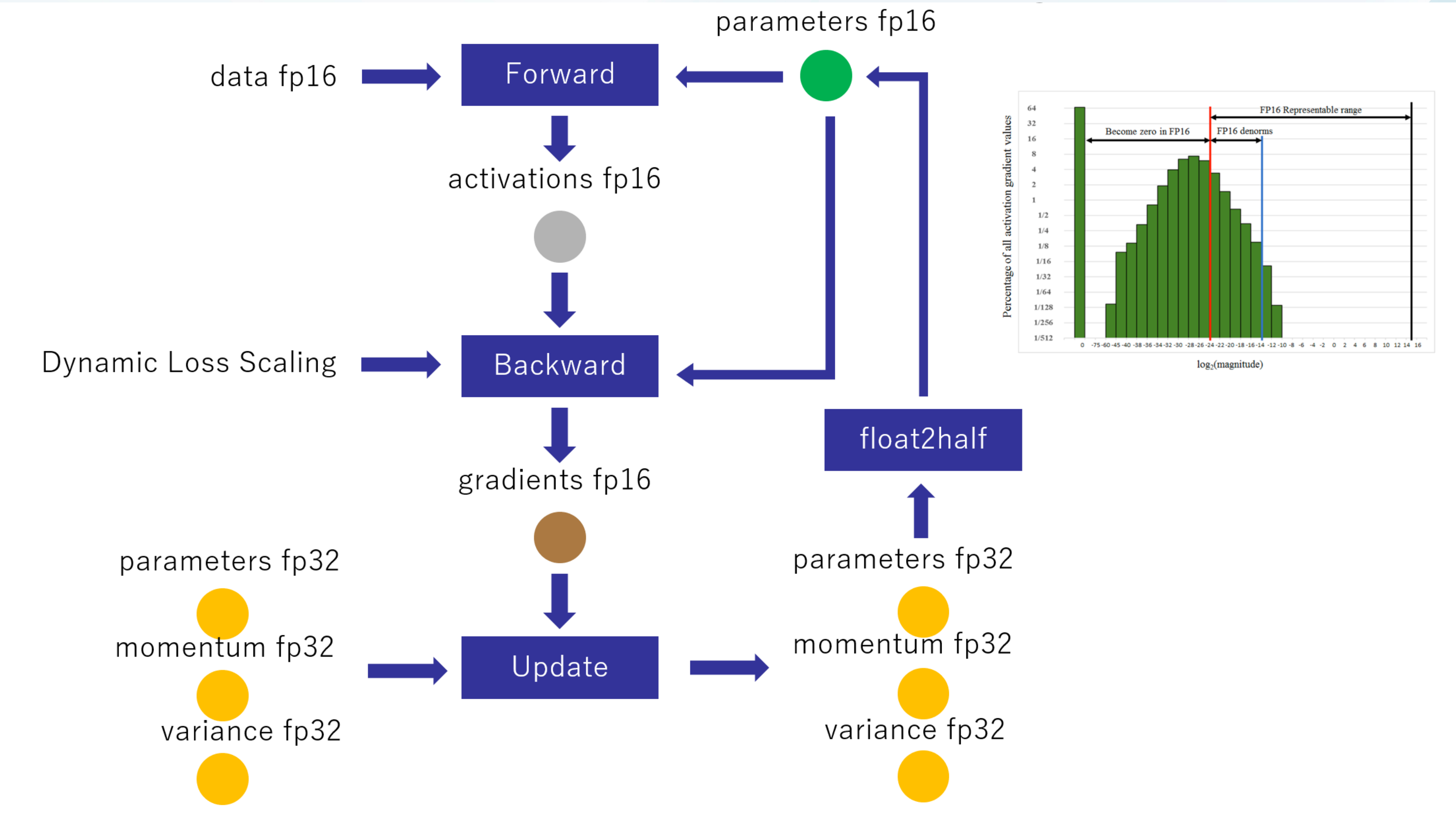


Fine-tuning: Mixed Precision

NVIDIA A100 TENSOR CORE GPU SPECIFICATIONS (SXM4 AND PCIE FORM FACTORS)

	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			
GPU Memory	40GB HBM2	80GB HBM2e	40GB HBM2	80GB HBM2e
GPU Memory Bandwidth	1,555GB/s	1,935GB/s	1,555GB/s	2,039GB/s
Max Thermal Design Power (TDP)	250W	300W	400W	400W
Multi-Instance	Up to 7	Up to 7	Up to 7	Up to 7

Fine-tuning: Mixed Precision

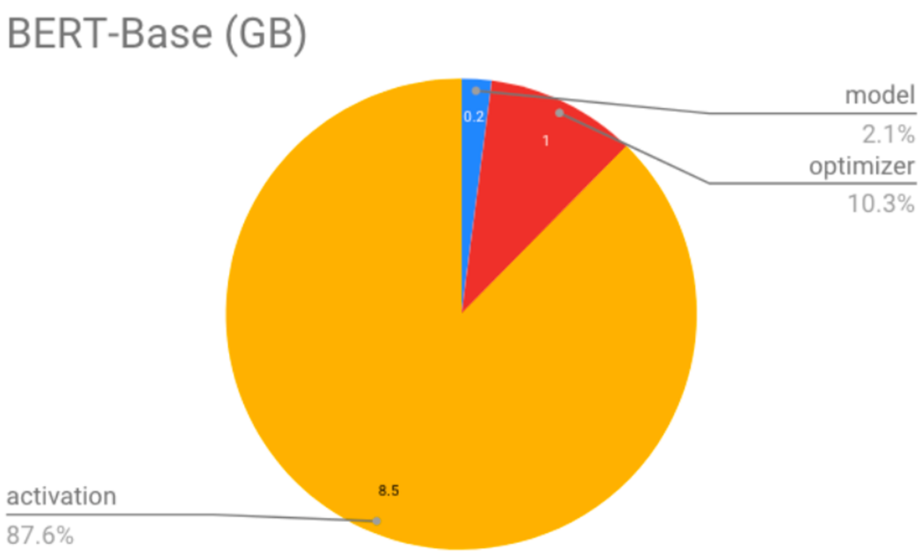


ZeRO: why not DP or MP?

> “ **Model states** often consume the largest amount of memory during training. DP has good compute/communication efficiency but poor memory efficiency while MP can have poor compute/communication efficiency.
>
> DP replicates the entire model states across all data parallel process resulting in redundant memory consumption; while MP partition these states to obtain high memory efficiency, but often result in too finegrained computation and expensive communication that is less scaling efficient. ”

ZeRO:Where the memory goes?

- **Model**
  - **Parameters（half）2 bytes**
  - **Gradients （half）2 bytes**
- **Optimizer states**
  - **Master Weight（fp32） 4 bytes**
  - **Adam m（fp32）4 bytes**
  - **Adam v（fp32） 4 bytes**
- **Activations：saved in forward function for backward**
- **Other：intermediate results in operators**



ZeRO Stages

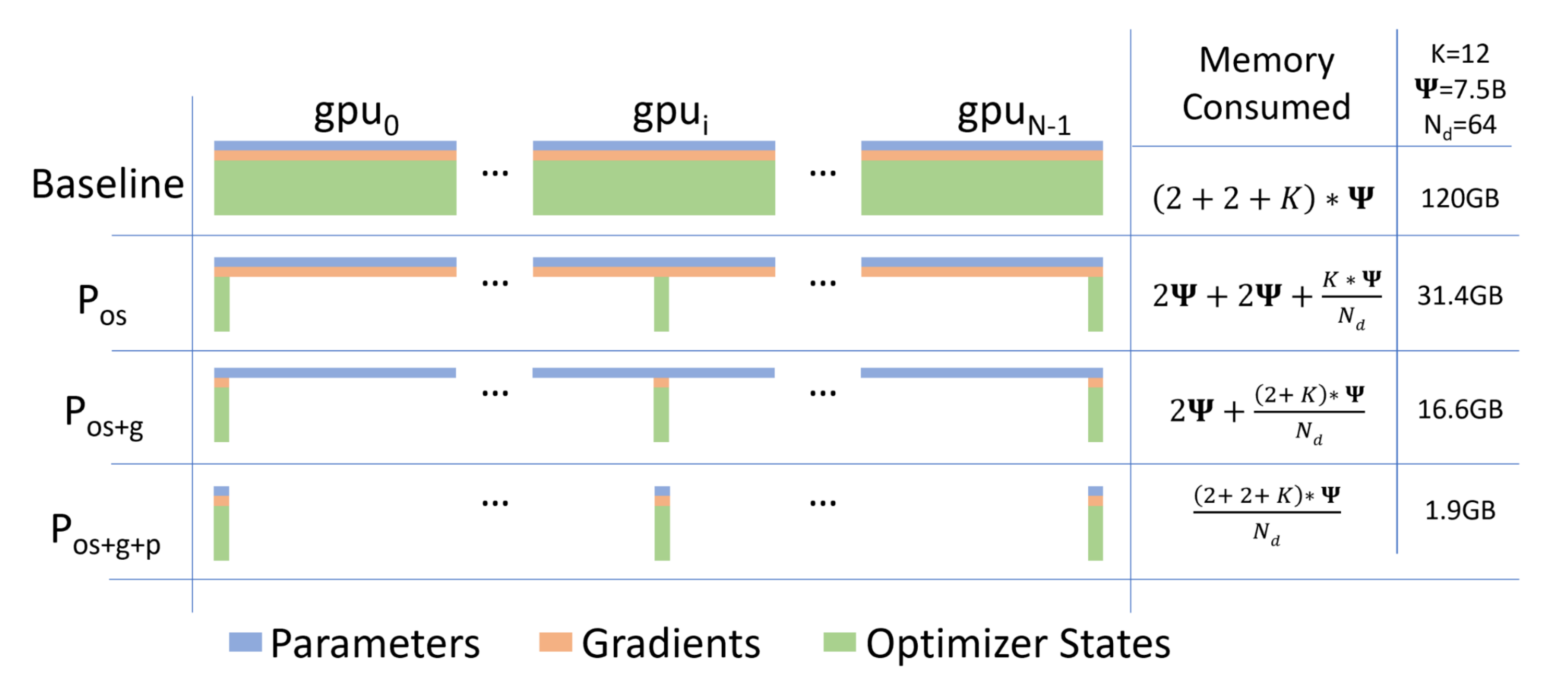
ZeRO-DP has three main optimization stages (as depicted in Figure 1), which correspond to the partitioning of optimizer states, gradients, and parameters. When enabled cumulatively:

1) Optimizer State Partitioning (P_{os}): 4x memory reduction, same communication volume as DP;

2) Add Gradient Partitioning (P_{os+g}): 8x memory reduction, same communication volume as DP;

3) Add Parameter Partitioning (P_{os+g+p}): Memory reduction is linear with DP degree N_d. For example, splitting across 64 GPUs ($N_d = 64$) will yield a 64x memory reduction. There is a modest 50% increase in communication volume.

ZeRO Stages


gpu_0
gpu_i
gpu_{N-1}
Memory Consumed
K=12
Ψ=7.5B
N_d=64
Baseline
$(2+2+K)*\Psi$
120GB
P_{os}
$2\Psi + 2\Psi + \frac{K*\Psi}{N_d}$
31.4GB
P_{os+g}
$2\Psi + \frac{(2+K)*\Psi}{N_d}$
16.6GB
P_{os+g+p}
$\frac{(2+2+K)*\Psi}{N_d}$
1.9GB
Parameters
Gradients
Optimizer States

P-tuning v2

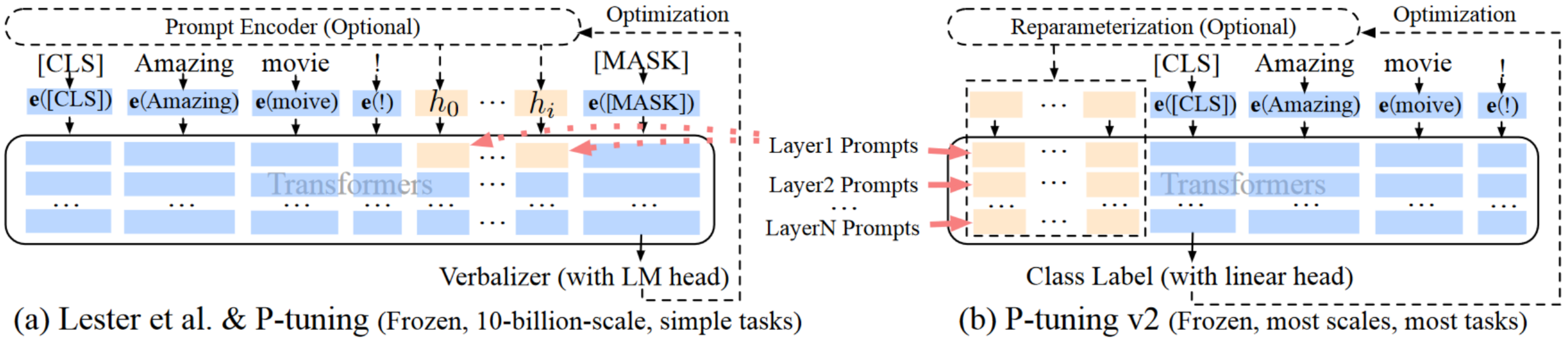


Figure 2: From Lester et al. (2021) & P-tuning to P-tuning v2. Orange blocks (i.e., $h_0, ..., h_i$) refer to trainable prompt embeddings; blue blocks are embeddings stored or computed by frozen pre-trained language models.

- Saves GPU memory & training time
- Similar performace

P-tuning v2: Results

	#Size	BoolQ			CB			COPA			MultiRC (F1a)		
		FT	PT	PT-2	FT	PT	PT-2	FT	PT	PT-2	FT	PT	PT-2
BERT$_{large}$	335M	**77.7**	67.2	75.8	**94.6**	80.4	**94.6**	69.0	55.0	**73.0**	70.5	59.6	**70.6**
RoBERTa$_{large}$	355M	**86.9**	62.3	84.8	98.2	71.4	**100**	**94.0**	63.0	93.0	**85.7**	59.9	82.5
GLM$_{xlarge}$	2B	**88.3**	79.7	87.0	**96.4**	76.4	**96.4**	**93.0**	92.0	91.0	84.1	77.5	**84.4**
GLM$_{xxlarge}$	10B	88.7	**88.8**	**88.8**	**98.7**	98.2	96.4	**98.0**	**98.0**	**98.0**	**88.1**	86.1	**88.1**

	#Size	ReCoRD (F1)			RTE			WiC			WSC		
		FT	PT	PT-2	FT	PT	PT-2	FT	PT	PT-2	FT	PT	PT-2
BERT$_{large}$	335M	70.6	44.2	**72.8**	70.4	53.5	**78.3**	74.9	63.0	**75.1**	**68.3**	64.4	**68.3**
RoBERTa$_{large}$	355M	89.0	46.3	**89.3**	86.6	58.8	**89.5**	**75.6**	56.9	73.4	63.5	**64.4**	63.5
GLM$_{xlarge}$	2B	91.8	82.7	**91.9**	**90.3**	85.6	**90.3**	**74.1**	71.0	72.0	**95.2**	87.5	92.3
GLM$_{xxlarge}$	10B	**94.4**	87.8	92.5	**93.1**	89.9	**93.1**	**75.7**	71.8	74.0	**95.2**	94.2	93.3

Table 2: Results on SuperGLUE development set. P-tuning v2 surpasses P-tuning & Lester et al. (2021) on models smaller than 10B, matching the performance of fine-tuning across different model scales. (FT: fine-tuning; PT: Lester et al. (2021) & P-tuning; PT-2: P-tuning v2; **bold**: the best; underline: the second best).

P-tuning @ ChatGLM-6B

Example: AdGen

- Dependencies

```
pip install rouge_chinese nltk jieba datasets
```

- Dataset
  https://cloud.tsinghua.edu.cn/f/b3f119a008264b1cabd1/?dl=1

```
{
    "content": "类型#上衣*版型#宽松*版型#显瘦*图案#线条*衣样式#衬衫*衣袖型#泡泡袖*衣款式#抽绳",
    "summary": "这件衬衫的款式非常的宽松，利落的线条可以很好的隐藏身材上的小缺点，穿在身上有着很好的显瘦效果。
        领口装饰了一个可爱的抽绳，漂亮的绳结展现出了十足的个性，配合时尚的泡泡袖型，尽显女性甜美可爱的气息。"
}
```

- Specify model path, dataset path & device ordinal in `train.sh` & `evaluate.sh`
- Run

```
bash train.sh
```

  - Default we use 4-bit quantization, this may take a while ...
  - remove `--quantization_bit 4` to use fp16

quantizaion	GPU memory	Training Time @ 3k steps
/	13GB	~2hrs
4bit	7GB	~3hrs

- See results

```
bash evaluate.sh
```

- This will make generation on the test set

Full parameter finetuning

- Install `deepspeed`

```
pip install deepspeed
```

- Specify model and dataset in `ds_train_finetune.sh` and `evaluate_finetune.sh`
- 3090 is in sufficient for this task ...
- Run

```
bash ds_train_finetune.sh
```

FAQs: Try just rerun

```
Traceback (most recent call last):
  File "main.py", line 435, in <module>
    main()
  File "main.py", line 374, in main
    train_result = trainer.train(resume_from_checkpoint=checkpoint)
  File "/root/ChatGLM-6B/ptuning/trainer.py", line 1635, in train
    return inner_training_loop(
  File "/root/ChatGLM-6B/ptuning/trainer.py", line 1704, in _inner_training_loop
    deepspeed_engine, optimizer, lr_scheduler = deepspeed_init(
  File "/opt/conda/lib/python3.8/site-packages/transformers/deepspeed.py", line 378, in deepspeed_init
    deepspeed_engine, optimizer, _, lr_scheduler = deepspeed.initialize(**kwargs)
  File "/opt/conda/lib/python3.8/site-packages/deepspeed/__init__.py", line 165, in initialize
    engine = DeepSpeedEngine(args=args,
  File "/opt/conda/lib/python3.8/site-packages/deepspeed/runtime/engine.py", line 266, in __init__
    self._configure_distributed_model(model)
  File "/opt/conda/lib/python3.8/site-packages/deepspeed/runtime/engine.py", line 1066, in _configure_distributed_model
    self.data_parallel_group = groups._get_data_parallel_group()
  File "/opt/conda/lib/python3.8/site-packages/deepspeed/utils/groups.py", line 327, in _get_data_parallel_group
    return _clone_world_group()
  File "/opt/conda/lib/python3.8/site-packages/deepspeed/utils/groups.py", line 315, in _clone_world_group
    _WORLD_GROUP = dist.new_group(ranks=range(dist.get_world_size()))
  File "/opt/conda/lib/python3.8/site-packages/deepspeed/comm/comm.py", line 179, in new_group
    return cdb.new_group(ranks)
  File "/opt/conda/lib/python3.8/site-packages/deepspeed/comm/torch.py", line 234, in new_group
    return torch.distributed.new_group(ranks)
  File "/opt/conda/lib/python3.8/site-packages/torch/distributed/distributed_c10d.py", line 3006, in new_group
    _store_based_barrier(global_rank, default_store, timeout)
  File "/opt/conda/lib/python3.8/site-packages/torch/distributed/distributed_c10d.py", line 239, in _store_based_barrier
    store.add(store_key, 1)
RuntimeError: Broken pipe
```

Contention? add a lock

```
# Load pretrained model and tokenizer
with FileLock("model.lock"):
    config = AutoConfig.from_pretrained(model_args.model_name_or_path, trust_remote_code=True)
...
with FileLock("model.lock"):
    tokenizer = AutoTokenizer.from_pretrained(model_args.model_name_or_path, trust_remote_code=True)

if model_args.ptuning_checkpoint is not None:
    ...
else:
    with FileLock("model.lock"):
        model = AutoModel.from_pretrained(model_args.model_name_or_path, config=config, trust_remote_code=True)
```

- Just don't start simultaneously

LoRA

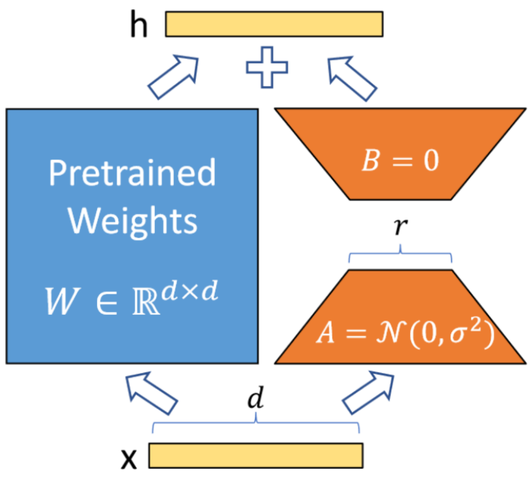


Figure 1: Our reparametrization. We only train A and B.

Hu, Edward J., et al. "LoRA: Low-Rank Adaptation of Large Language Models." International Conference on Learning Representations.

LoRA

- Suppose pre-trained weight $W_0 \in \mathbb{R}^{d \times k}$, input $x \in \mathbb{R}^k$
- Fine-tuning: $W = W_0 + \Delta W$
- ΔW is not necessarily full-rank!
- LoRA:
  - suppose $\Delta W = AB$ has rank r, where $A \in \mathbb{R}^{d \times r},\ B \in \mathbb{R}^{r \times k}$

$$W = W_0 + AB$$

  - $r \ll \min(d, k)$
  - trainable parameters are significantly reduced

LoRA

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

LoRA @ ChatGLM-6B

- We procede the demo with a community implementation
  - https://github.com/yuanzhoulvpi2017/zero_nlp
- Ref:
- It's implemented on a previous version of ChatGLM-6B
  - Download checkpoint a previous archive from HuggingFace

```
git clone https://huggingface.co/yuanzhoulvpi/chatglm6b-dddd
```

  - Note that `git-lfs` is required

LoRA @ ChatGLM-6B

- This Notebook demonstrates how to finetune ChatGLM-6B with LoRA on `alpaca_chinese` dataset
- Now we show steps to reuse the code and finetune on AdGen dataset
  - Understand code behaviour and your requirements
  - Make modifications accordingly
  - Sanity check, debug, run
  - Evaluate
- Takes ~15GB GPU memory

Thanks

Questions?