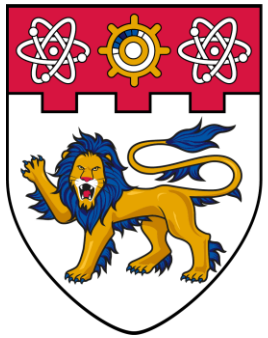


NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

AdaCheck: An Adaptive Checkpointing System for Efficient LLM Training with Redundancy Utilization

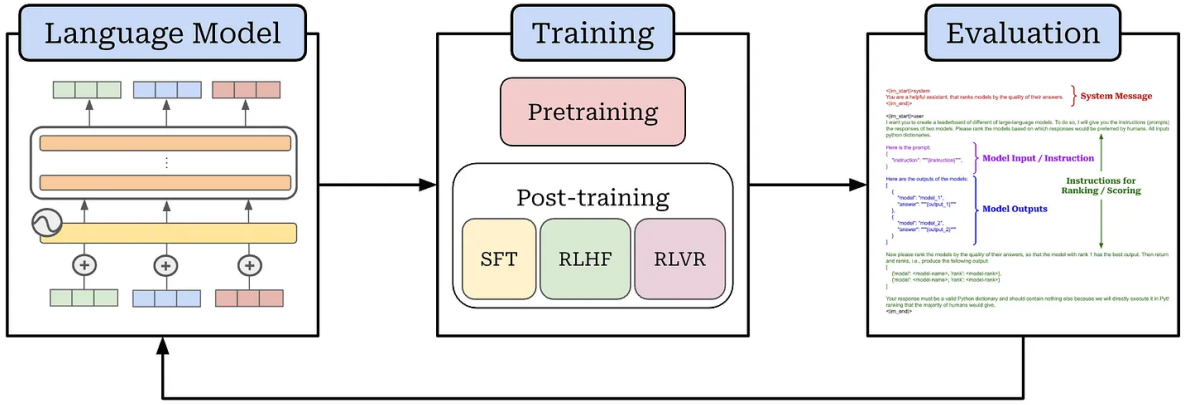
Weijie Liu*†, Shengwei Li*†, Zhiquan Lai†✉, Keshi Ge†, Qiaoling Chen‡, Peng Sun §, Dongsheng Li†✉, Kai Lu†

† National University of Defense Technology ‡ Nanyang Technological University § Shanghai AI Laboratory

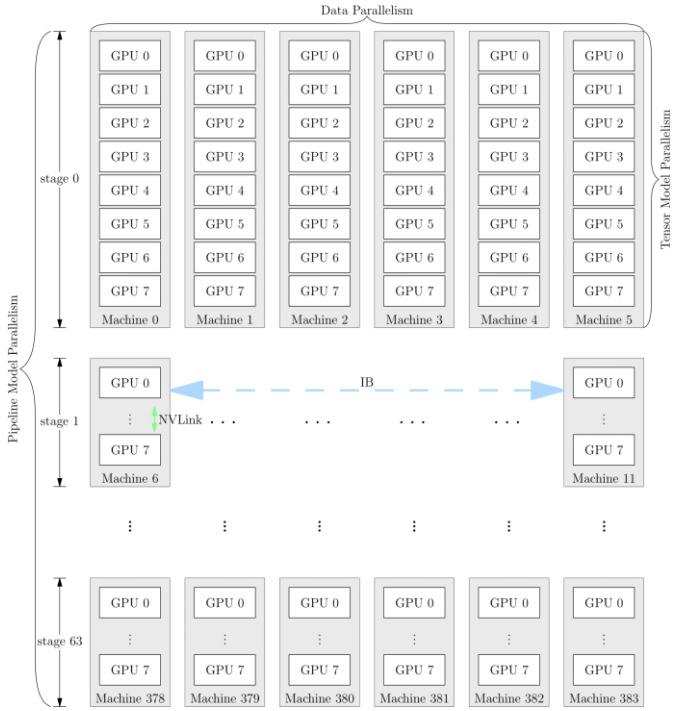
*: Co-first Authors

LLM Training

- Training recipe of LLMs: large-scale distributed training
  - Models are iteratively trained on hundreds or even thousands of workers.



Iterative training of models

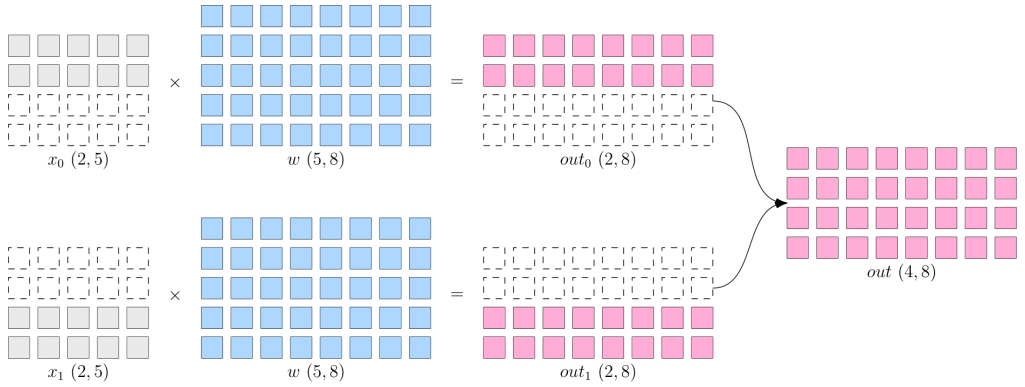


The training cluster and adopted parallelism

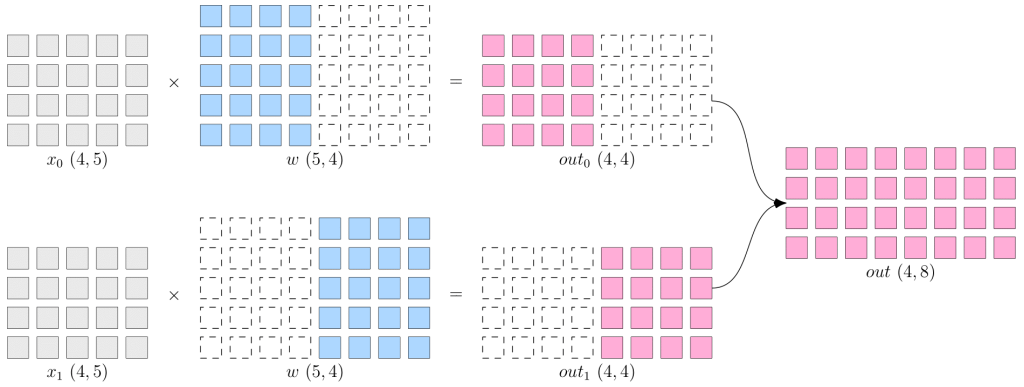
Image sources: https://cameronrwolfe.substack.com/p/llm-debugging; https://docs.oneflow.org/master/parallelism/01_introduction.html

LLM Training

- Training recipe of LLMs: large-scale distributed training
  - Models are iteratively trained on hundreds or even thousands of workers.
  - Model states are distributed to workers according to the adopted parallelism.



Data parallelism



Model parallelism

Image source: https://docs.oneflow.org/master/parallelism/01_introduction.html

LLM Training

- Training recipe of LLMs: large-scale distributed training
  - Models are iteratively trained on hundreds or even thousands of workers.
  - Model states are distributed to workers according to the adopted parallelism.
  - State redundancy, defined as the duplication of model states resulting from parallelisms, model architectures, and training iterations, facilitates the training process.

Checkpointing Systems

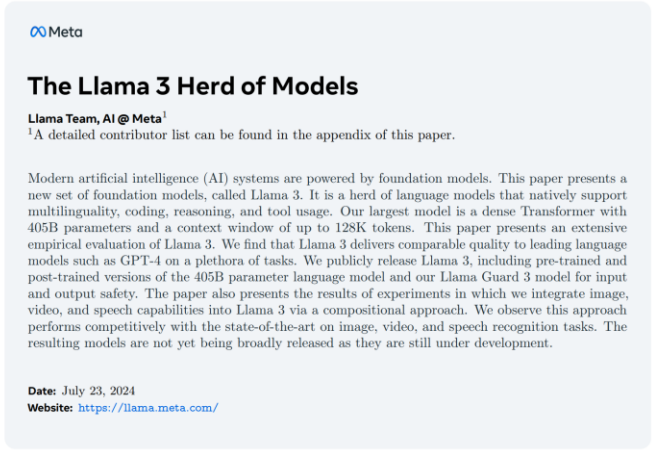
- Frequent failures in LLM training
  - LLaMA-3.1 encountered 419 failures over 54 days, failing every ~3 hours.
  - OPT model training reports a failure frequency of twice a day.

Checkpointing Systems

- Frequent failures in LLM training
  - LLaMA-3.1 encountered 419 failures over 54 days, failing every ~3 hours.
  - OPT model training reports a failure frequency of twice a day.
- Checkpointing: the core of failure recovery
  - Periodically save checkpoints during training.
  - Enable the resumption of training progress from the latest checkpoint.

Checkpointing Systems

- Notable waste of training resources in checkpointing
  - About 2M GPU hours are wasted in the training of LLaMA 3.1.
  - About 178,000 GPU hours are wasted in the training of OPT-175B.



Meta

The Llama 3 Herd of Models

Llama Team, AI @ Meta[1]
[1]A detailed contributor list can be found in the appendix of this paper.

Modern artificial intelligence (AI) systems are powered by foundation models. This paper presents a new set of foundation models, called Llama 3. It is a herd of language models that natively support multilinguality, coding, reasoning, and tool usage. Our largest model is a dense Transformer with 405B parameters and a context window of up to 128K tokens. This paper presents an extensive empirical evaluation of Llama 3. We find that Llama 3 delivers comparable quality to leading language models such as GPT-4 on a plethora of tasks. We publicly release Llama 3, including pre-trained and post-trained versions of the 405B parameter language model and our Llama Guard 3 model for input and output safety. The paper also presents the results of experiments in which we integrate image, video, and speech capabilities into Llama 3 via a compositional approach. We observe this approach performs competitively with the state-of-the-art on image, video, and speech recognition tasks. The resulting models are not yet being broadly released as they are still under development.

Date: July 23, 2024
Website: https://llama.meta.com/

LLaMA 3.1 technical report

OPT: Open Pre-trained Transformer Language Models

Susan Zhang[*], Stephen Roller[*], Naman Goyal[*], Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott[†], Sam Shleifer[†], Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, Luke Zettlemoyer

Meta AI

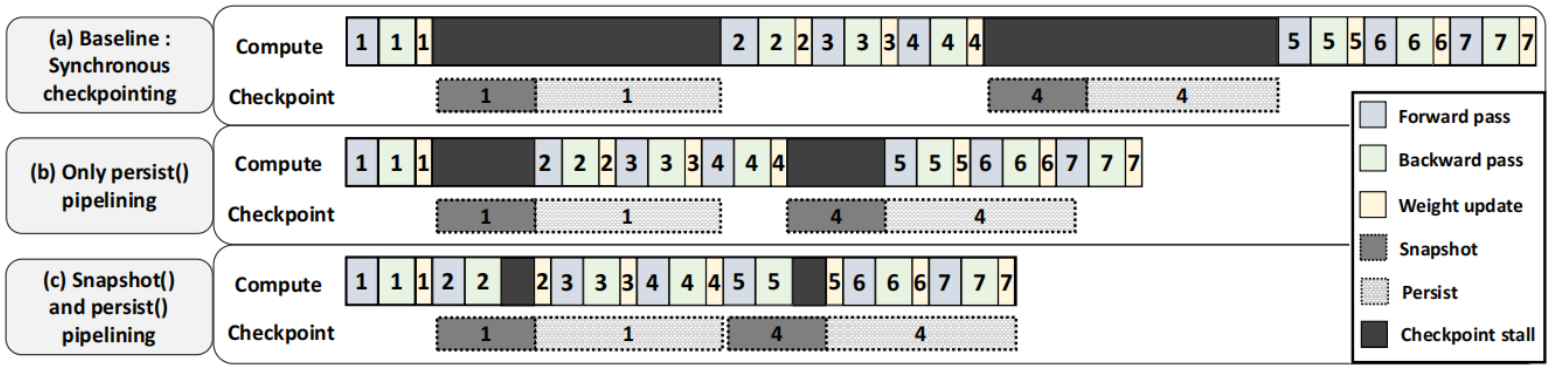
{susanz,roller,naman}@fb.com

OPT technical report

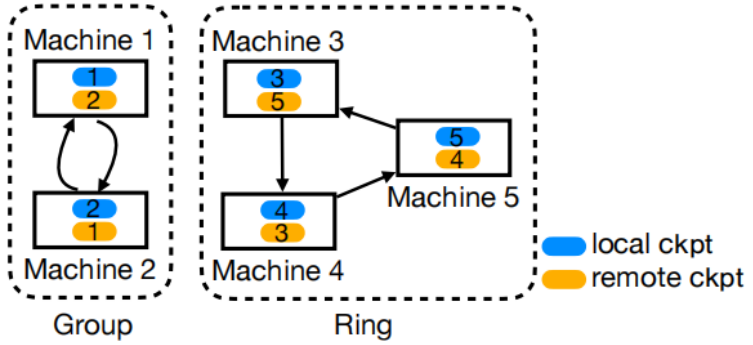
Image sources: https://arxiv.org/pdf/2407.21783; https://arxiv.org/pdf/2205.01068

Motivation and Opportunity

- Limitations of current checkpointing systems
  - Offline solutions tailored for specific parallelisms or model architectures.



CheckFreq: limit only rank 0 worker saves in data parallelism

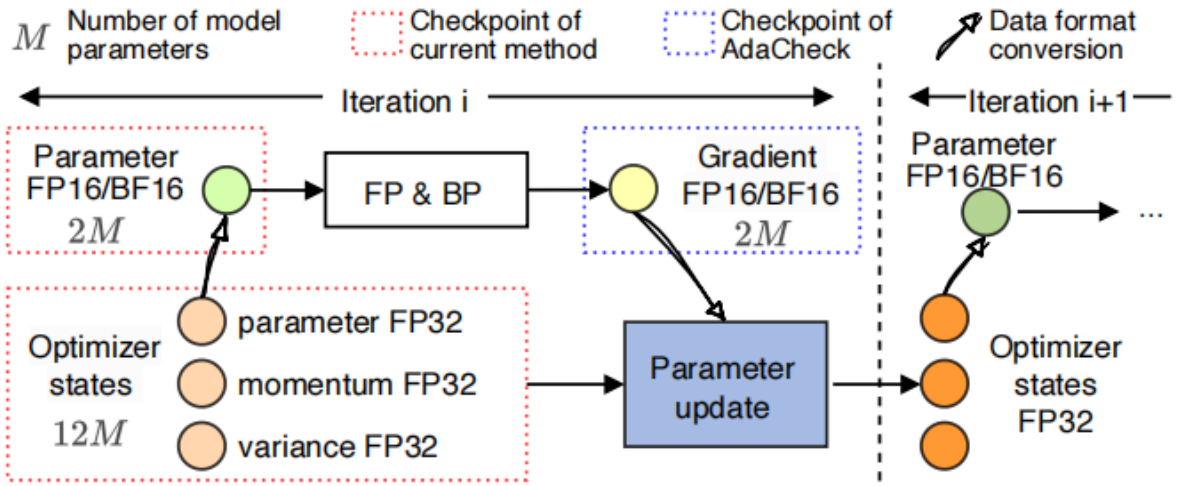


GEMINI: special design for ZeRO-3

Image sources: https://www.usenix.org/system/files/fast21-mohan.pdf; https://dl.acm.org/doi/epdf/10.1145/3600006.3613145

Motivation and Opportunity

- Limitations of current checkpointing systems
  - Offline solutions tailored for specific parallelisms or model architectures.
  - Fail to leverage the state redundancy across training iterations.



Workflow of mixed-precision training in each training iteration

Motivation and Opportunity

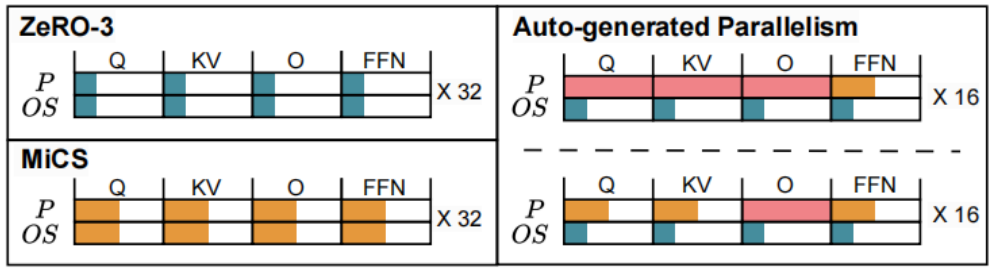
- Primary target of AdaCheck

  - Build an adaptive checkpointing system that achieves minimized checkpoint size by characterizing and exploiting state redundancy across various parallelisms, model architectures, and training iterations.

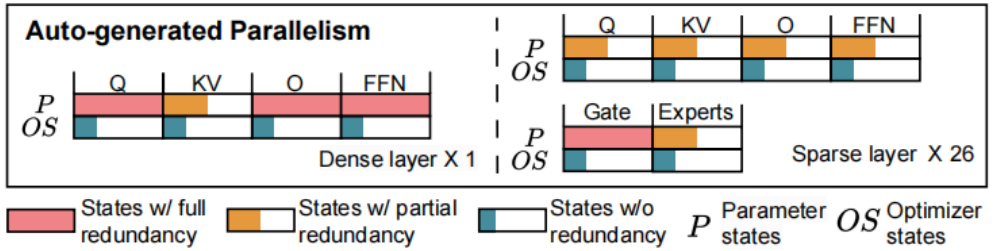
Motivation and Opportunity

Opportunities for AdaCheck

- Complexity of state redundancy.



(a) GPT-7B training in 128 3090 GPUs.

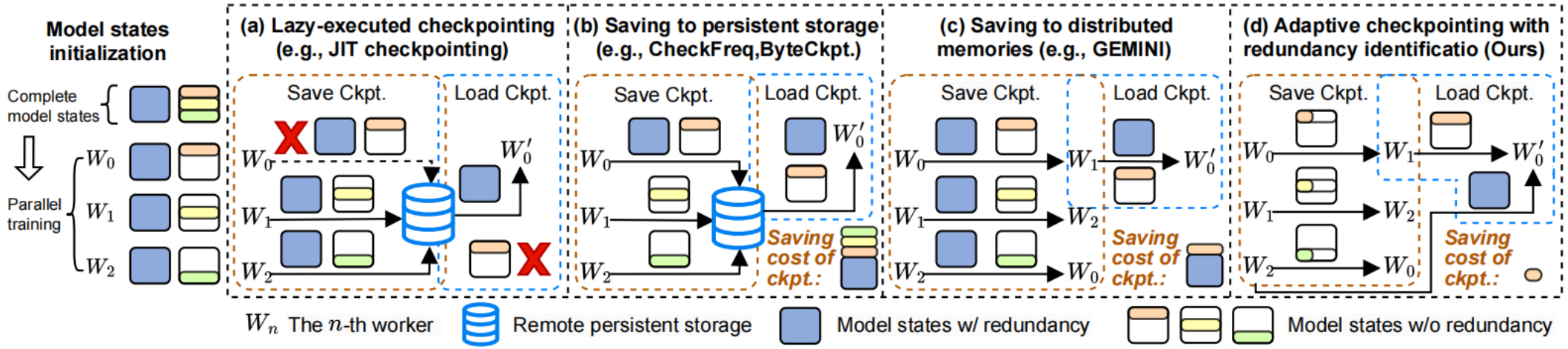


(b) DeepSeek-V2-Lite training in 32 A800 GPUs.

State redundancy across diverse parallel strategies and model architectures

Motivation and Opportunity

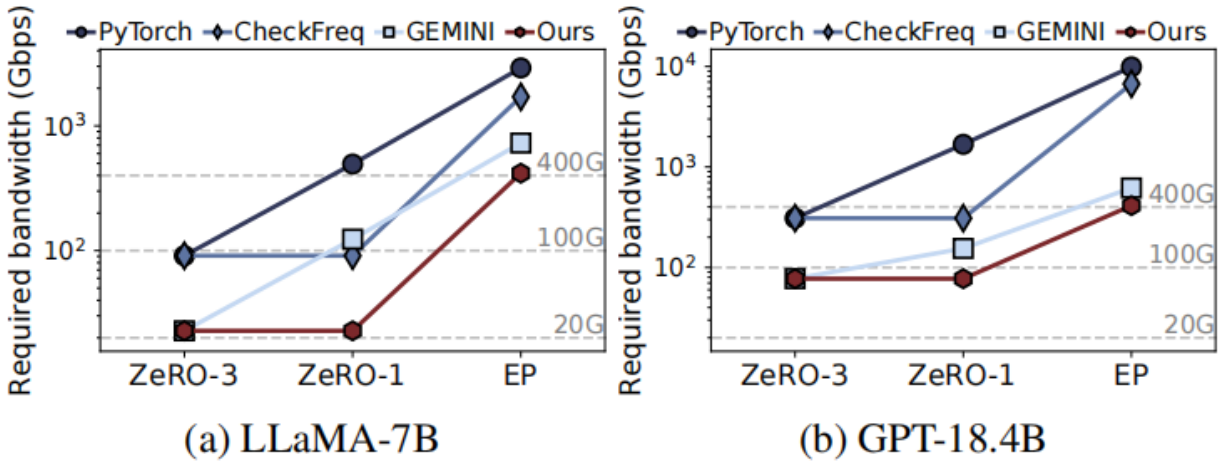
- Opportunities for AdaCheck
  - Complexity of state redundancy.
  - Difficulty of acquiring fine-grained state redundancy.



Workflow of different checkpointing systems

Motivation and Opportunity

- Opportunities for AdaCheck
  - Complexity of state redundancy.
  - Difficulty of acquiring fine-grained state redundancy.
  - Offline redundancy utilization is inadequate for 1S1C.

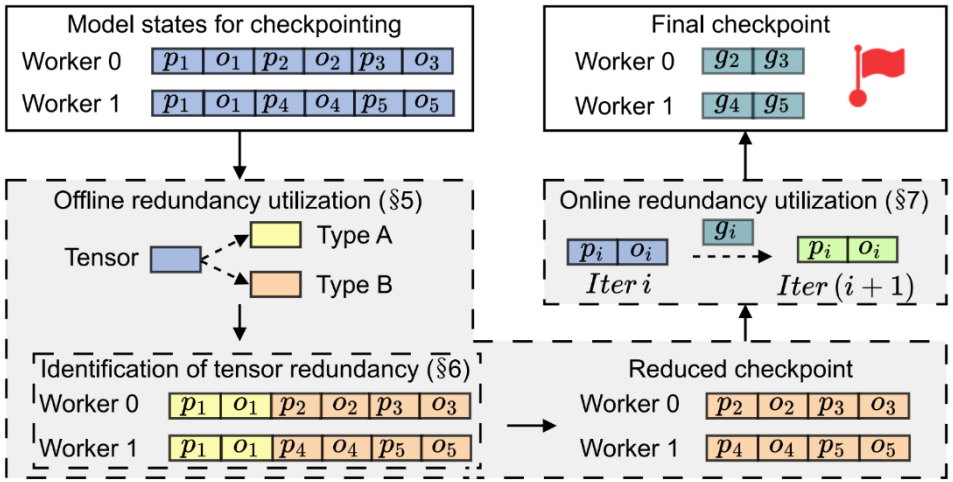


Required bandwidth for 1S1C on 32 A800 GPUs

AdaCheck Design

Overview

- Offline redundancy utilization.
  - Δ Model the state redundancy across parallelisms and model architectures.
  - Δ Select appropriate model states to generate a reduced checkpoint.
- Identification of tensor redundancy.
  - Δ Identify tensor redundancy efficiently.
- Online redundancy utilization.
  - Δ Minimize checkpoint size by using state redundancy across training iterations.



AdaCheck overview

AdaCheck Design

- Offline redundancy utilization
  - Proposal of tensor redundancy.
    - Δ Workers for LLM training: $W = \{w_1, w_2, \ldots, w_{|W|}\}$.
    - Δ Tensors possessed by w_i: $T_i = \{t_1, t_2, \ldots, t_{|T_i|}\}$.
    - Δ Location of tensor t_n on worker w_m: (m, n).
    - Δ Tensor redundancy for tensor t_k on worker w_i:

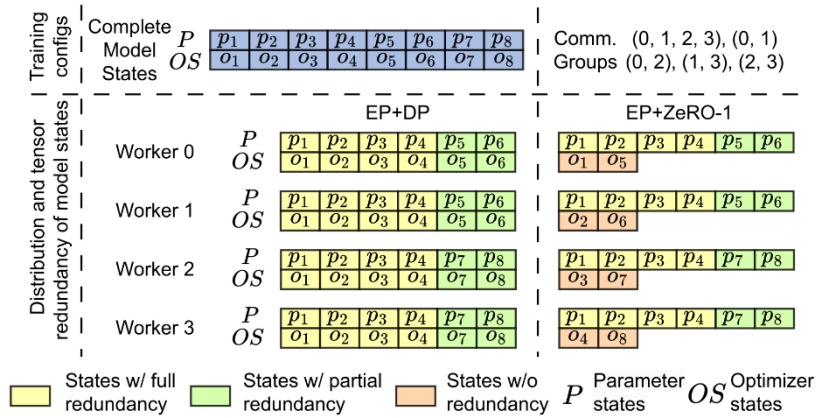
$$R_i^k = \{(m_1, n_1), (m_2, n_2), \ldots, (m_{|R_i^k|}, n_{|R_i^k|})\}.$$

AdaCheck Design

- Offline redundancy utilization
  - Proposal of tensor redundancy.
  - State redundancy across parallelisms and model architectures.
    - State redundancy for a state in worker w_i: $\{R_i^k, \ldots, R_i^j\}$.
    - Three types of state redundancy:
      - States with full redundancy. A state exhibits full redundancy if any tensor t_k within it satisfies $|R_i^k| = |W|$and $|W| > 1$.
      - States with partial redundancy. A state exhibits partial redundancy if any tensor $t_k$within it satisfies $1 < |R_i^k| < |W|$.
      - States with no redundancy. A state possesses no redundancy if any tensor $t_k$within it satisfies $|R_i^k| = 1$.

AdaCheck Design

- Offline redundancy utilization
  - ➢ Proposal of tensor redundancy.
  - ➢ State redundancy across parallelisms and model architectures.
  - ➢ Offline redundancy utilization method.
    - Δ Compute the intersection of state redundancy between parameters and their corresponding optimizer states.
    - Δ Utilize the intersection results to determine inclusion in the checkpoint.
    - Δ Store necessary parameters and their corresponding optimizer states.

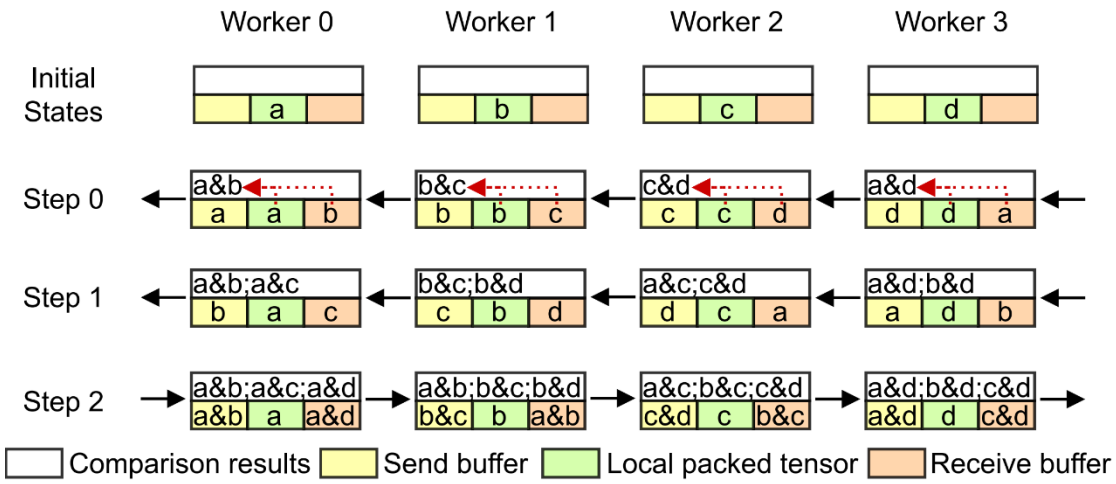


The distribution of state redundancy

AdaCheck Design

Identification of tensor redundancy

- Proposal of the redundancy detector.
  - Δ Hash-based consistency check.
  - Δ Reduce the comparison scale.
  - Δ Ring-based communication algorithm.



An example of redundancy detector

Algorithm 1 Redundancy detector of `AdaCheck`.

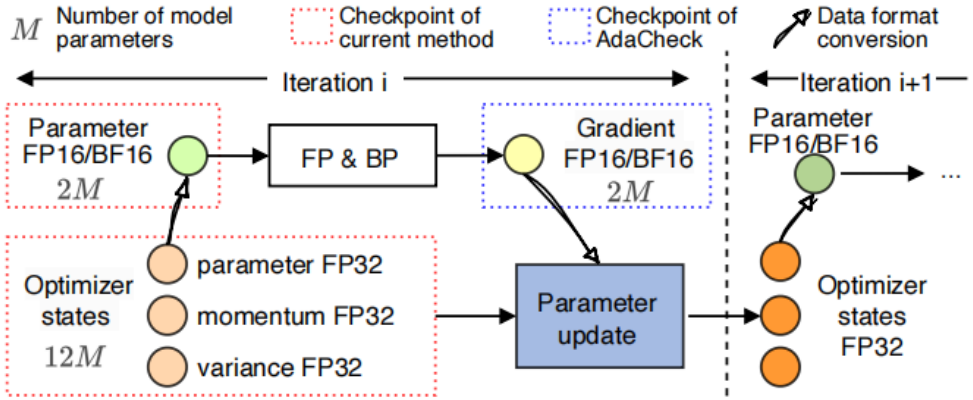
Input: The global rank r of current worker.
Output: Comparison results C_r.

1: Get the packed tensor of worker r as p.
2: Initialize comparison results C_r as a dictionary.
3: Get communication groups of worker r as C_g.
4: **for** communication group g **in** C_g **do**
5: Get the size of g as N.
6: trans_tensor = $N/2$;
7: trans_result = $N \bmod 2 == 0\ ?\ N/2-1 : N/2$;
8: Get index of r in g as r_i;
9: send_rank = $g[(r_i - 1 + N) \bmod N]$;
10: recv_rank = $g[(r_i + 1 + N) \bmod N]$;
11: send_tensor = p;
12: recv_tensor = $None$;
13: **for** i = 0 **to** trans_tensor **do**
14: $tensor\_rank = g[(r_i + 1 + i + N) \bmod N]$;
15: P2P.send(send_rank, send_tensor);
16: recv_tensor = P2P.recv(recv_rank);
17: **if** $tensor\_rank$ **not in** C_r **then**
18: $C_r[tensor\_rank]$ = **intersect**(p, recv_tensor);
19: send_tensor = recv_tensor;
20: **for** i = 0 **to** trans_result **do**
21: send_rank = $g[(r_i$+trans_result$-i+N) \bmod N]$;
22: recv_rank = $g[(r_i$−trans_result+$i+N) \bmod N]$;
23: send_tensor = C_r[send_rank];
24: P2P.send(send_rank, send_tensor);
25: recv_tensor = P2P.recv(recv_rank);
26: **if** recv_rank **not in** C_r **then**
27: C_r[recv_rank] = recv_tensor;

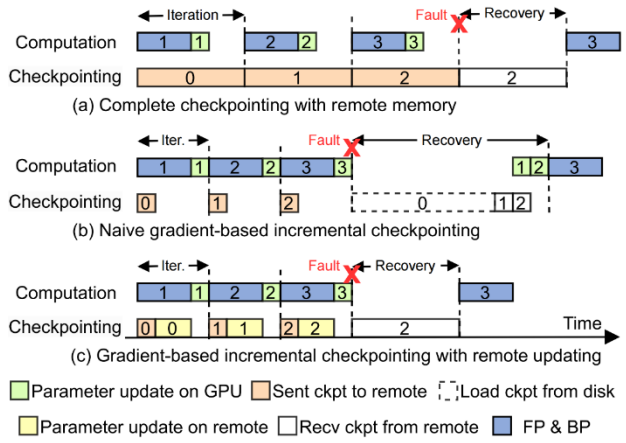
The algorithm of redundancy detector

AdaCheck Design

- Online redundancy utilization
  - ➢ Only store the smaller half-precision gradients to remote memory.
  - ➢ Gradient-based incremental checkpointing.
    - Δ Employ a modified CPU optimizer to update parameters in CPU memory.



Workflow of mixed-precision training in each training iteration



Gradient-based incremental checkpointing

Experimental Settings

- Platform
  - Datacenter (DCN) cluster.
  - Commodity (CMD) cluster.
- Models
  - Popular dense and sparse LLMs.
- Baselines
  - CheckFreq: asynchronous I/O checkpointing.
  - GEMINI: In-memory checkpointing.

Name	GPUs per node	# of GPUs	CPU mem.	Training bandwidth	Persistant storage bandwidth
DCN	8×A800 80G	32	2048GB	800Gbps	50Gbps
CMD	4×3090 24G	128	128GB	100Gbps	10Gbps

Hardware of clusters for evaluation

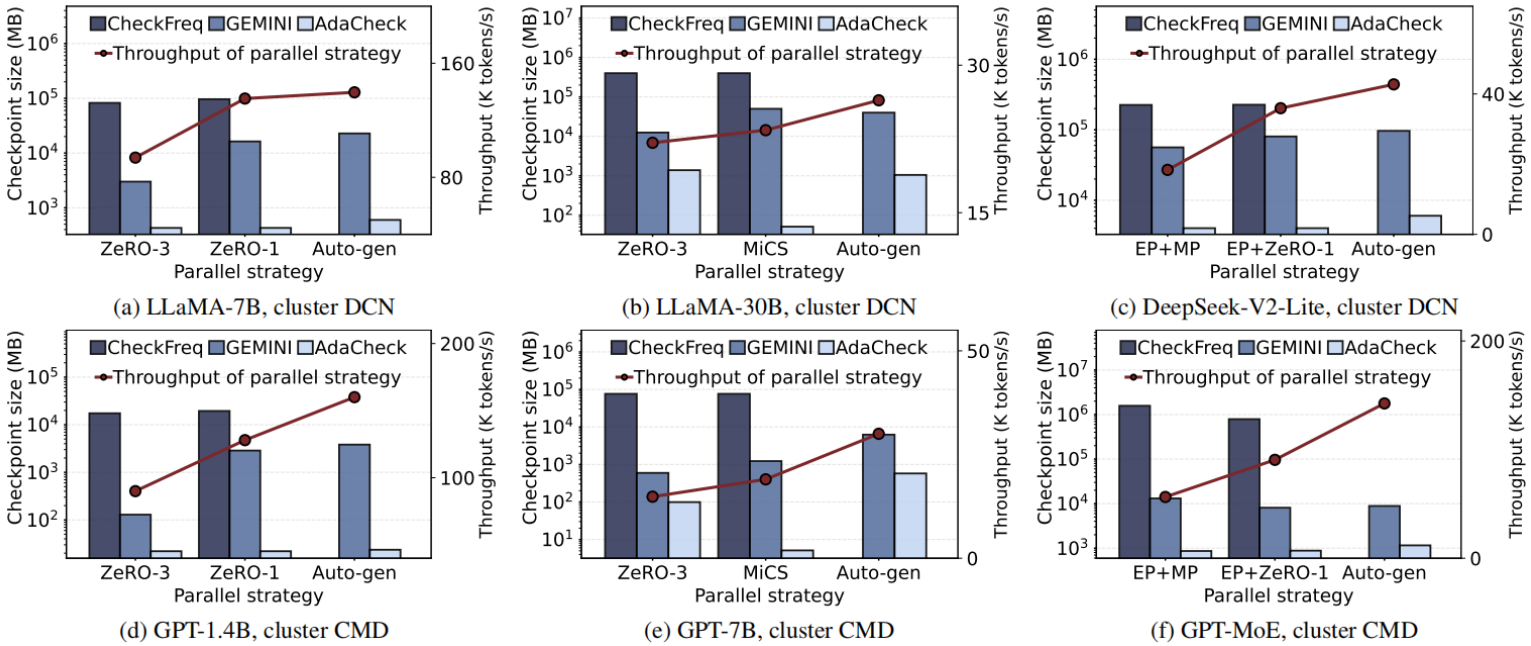
Cluster	Model	Arch.	Hidden	Intermediate	#Layers
DCN	LLaMA-7B	Dense	4096	11008	32
	LLaMA-30B	Dense	6144	16384	60
	DeepSeek-V2-Lite	Sparse	2048	10944	27
CMD	GPT-1.4B	Dense	1536	6144	48
	GPT-7B	Dense	4096	14336	32
	GPT-MoE	Sparse	1536	6144	48

Models used for evaluation

Evaluation

Checkpointing efficiency

- AdaCheck can significantly reduce the checkpoint size by 6.00–896x.

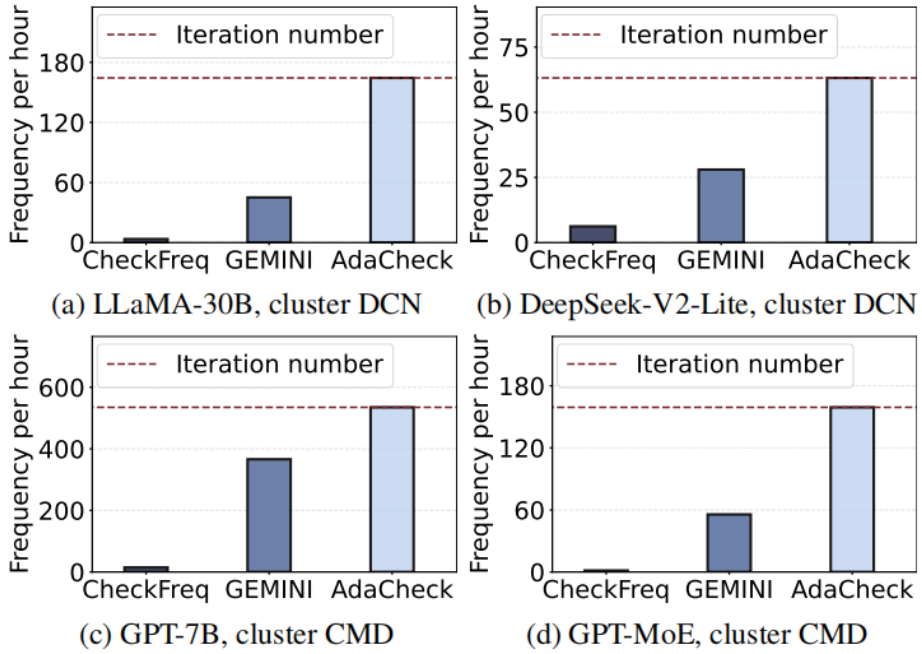


Checkpoint sizes of different checkpointing methods

Evaluation

Checkpointing efficiency

- AdaCheck enables checkpointing in each train iteration.



Checkpoint frequency of different checkpointing methods

More Details in Our Paper

- Design details
  - Time complexity analysis of redundancy detector (Appendix A).
  - State redundancy with auto-generated parallelism (Appendix B).
  - Proof of the recovery probability (Appendix C).
- Evaluations
  - Failure recovery efficiency (Section 9.3).
  - Parallel training efficiency (Section 9.4).
  - Checkpointing reliability (Section 9.5).
  - Ablation study (Section 9.6).

Summary and Contributions

Problem

- Current checkpointing systems lack the flexibility to accommodate various parallelisms, and fail to leverage the state redundancy across training iterations.

Key Idea

- Utilize state redundancy across parallelisms, model architectures, and training iterations to reduce checkpoint size and improve checkpointing frequency.

Techniques in AdaCheck

- Offline redundancy utilization.
- Redundancy detector.
- Online redundancy utilization.

Results

- AdaCheck can significantly reduce the checkpoint size by 6.00–896x.

Q & A

Thanks!