Response to the reviewers

We thank the reviewers for their critical assessment of our work. In the following we address their concerns point by point.

Reviewer 1

Reviewer Point P 1.1 — The paper does not compare against recent fault tolerance systems such as Swift, Just-In-Time Checkpointing, or Parcae, which also explore minimizing checkpoints and improving recovery speed.

Reply:

Table 1: Comparison between different fault recovery methods.

Methods	Fault Detection	Task Restart	Periodic Checkpointing	Training Recovery	Redo Training	Application Scenario
Swift	Unoptimized	Unoptimized	Required by logging-based recovery (activations and gradients)	Replication-based: Device Communication Loggging-based: Load checkpoint and redo calculation	step / 2	Static Topology Fault Recovery
JIT	Unoptimized	Unoptimized	Intermediate Checkpoint (on disk) when a failure occurs	Load checkpoint	step / 2	Static Topology Fault Recovery
Parcae	Unoptimized	Unoptimized	Not Required	Device Communication	step / 2	Dynamic Topology Preemptible Instances on Clouds
FlashRecovery	Active Real-time Failure Detection	Scale-independent Task Restart	Not Required	Device Communication	step / 2	Static Topology Fault Recovery

FlashRecovery is a holistic fault-tolerance framework that optimizes the entire failure recovery pipeline, unlike prior works that focus on some isolated phases. We formulate the costs of every phases (Eq. (1) in our paper), and systematically minimize each component. Tab. 1 presents a systematic comparison of recent fault recovery approaches across Swift, JIT, Parcae, and our FlashRecovery system, evaluating key dimensions including fault detection, task restart, and training recovery mechanisms. As we can see from the table, recent methods ignore the cost resulting from failure detection and task restart. The following sections provide detailed analysis.

Swift introduces two distinct fault recovery approaches: (1) replication-based recovery and (2) logging-based recovery. The replication-based method demonstrates conceptual alignment with our checkpoint-free approach, as both leverage data parallelism (DP) redundancy for state restoration. However, the logging-based alternative, designed for standalone pipeline parallelism (PP) scenarios, proves incompatible with our framework for two fundamental reasons: First, in large language model (LLM) training configurations, DP represents the dominant paradigm while standalone PP deployments are virtually nonexistent, making replication-based solutions sufficient for most cases. Second, logging-based recovery imposes prohibitive storage requirements by persisting both forward pass activations and backward pass gradients. Empirical results from Swift demonstrate this overhead, reporting 11.51GB additional storage for a 1.64B-parameter ViT-128/32 model in an 8-node cluster, escalating to 24.66GB in 16-node configurations. When scaled to modern LLM training scenarios involving hundreds of billions of parameters across hundreds even thousands of AI accelerators, these storage demands and associated I/O overhead become computationally intractable. Notably, Swift's documentation omits discussion of fault detection and task restart mechanisms. Swift decides a failure via a communication error, which delays the moment to detect the failure and makes waste of devices' work.

JIT implements a complete fault tolerance solution comprising three core components: (1) fault detection,(2) fault recovery mechanisms, and (3) checkpoint restoration. For fault detection, JIT adopts conventional timeout monitoring using watchdog processes (serving as our detection baseline). In the

recovery phase, the dominant overhead in JIT stems from NCCL connection re-establishment, which we optimizes through multi-level optimizations and reduce the time cost in seconds (detailed in Section "Scale-Independent Task Restart"). During training recovery, JIT utilizes DP redundancy to bound computational waste to approximately half a training step (step/2). However, its checkpoint-based approach necessitates that operational nodes persist model states before new nodes can reload them—an operation exhibiting linear scaling with model size. In contrast, our system completely eliminates checkpoint I/O through direct state synchronization via communication between devices, achieving significantly faster recovery.

Parcae adopts a fundamentally different approach as a DNN training system optimized for preemptible cloud instances in which the parallelism topology maybe change[1]. By design, it intentionally excludes traditional fault detection and recovery components. The most pertinent comparison lies in parameter migration strategies during instance preemption, where Parcae implements three variants: (1) intra-stage migration, (2) inter-stage migration, and (3) pipeline migration. Notably, Parcae's inter-stage and pipeline migration strategies employ device communication for state transfer, demonstrating architectural parallels with our checkpoint-free design where recreated processes recover state directly from DP replicas.

Table 2: Experiments on JIT and FlashRecovery (measured in seconds).

Methods (Model-Params-Devices)	Parallelism	Failure Detection	Checkpoint Saving	Task Restart	State Recovery	Average Redo Training
JIT (GPT-8B-16)	2D-4P-2T	1800	18.8	15.5	28.6 (Checkpoint Loading)	step/2
FlashRecovery (GPT-8B-16)	2D-4P-2T	≈ 10	0 (Not Required)	5.2	≈ 10 (State Broadcast)	step/2

As previously discussed, the approach Parcae is designed for preemptible scenarios, which is not applicable to our LLM training framework. Swift's implementation requires approximately 2,600 lines of Python code modifications and 400 lines of CUDA/C++ changes based on PyTorch 1.9 and NCCL 2.7.6. Due to substantial architectural differences in modern PyTorch 2.x frameworks, these modifications cannot be directly ported to our experimental environment. This incompatibility currently precludes direct quantitative comparisons with Swift's approach under identical conditions. We reproduced the results from JIT and run our method under identical experimental conditions (Tab. 2). Quantitative results indicate our method outperforms JIT across all recovery phases.

Reviewer Point P 1.2 — Although large-scale experiments are reported, the system implementation lacks sufficient detail to assess reproducibility or deployment complexity.

Reply: While extending our fault recovery technique to NVIDIA/AMD GPU clusters remains an important research objective, current trade compliance restrictions prevent legal access to NVIDIA / AMD hardwares for validation in our academic research. Nevertheless, we argue that porting our system to mainstream NVIDIA platforms would require minimal engineering effort due to its modular design.

FlashRecovery is constructed upon native PyTorch[2] interfaces and Ascend-specific extensions, with most core components already open-sourced:

- Torch_NPU[3], PyTorch backend for NPU acceleration

[1]Dynamic topology means the number of devices and parallel settings of a training task change during training.
[2]`https://github.com/pytorch/pytorch`
[3]`https://gitee.com/ascend/pytorch`

- ClusterD[4], a cluster-range information collection component, which can be used to sense failures.
- Device_Plugin[5], a component to detect device-related failures.
- Volcano[6], a component to reschedule devices of a cluster.

Notably, ClusterD and Volcano are hardware-agnostic components, implementing standardized distributed training abstractions that obviate platform-specific modifications. Only the Device_Plugin component, responsible for low-level hardware fault detection and resource enumeration, maintains direct hardware dependencies. For NVIDIA / AMD systems, this component could be substituted with equivalent monitoring solutions. This architecture ensures that most of the system code remains unchanged during platform migration, with only the device abstraction layer requiring adaptation.

Furthermore, we emphasize that our key optimization contributions focus not on low-level code modifications, but rather on enhancing the overall recovery workflow. Through our proposed principles, including active failure detection, parallelized communication group establishment, and leveraging DP redundancy for rapid model state recovery, we demonstrate that comparable fault recovery efficiency can be achieved on both NPU and NVIDIA GPU clusters.

Reviewer Point P 1.3 — While recovery latency is evaluated, the paper lacks ablation experiments to isolate the benefits of each design component.

Reply:

Table 3: The time costs of every phase of baseline and our method (measured in seconds).

Methods (Model-Params-Devices)	Parallelism	Failure Detection	Checkpoint Saving	Task Restart			State Recovery	Redo Training
				Container Cleanup	Container Create	Communication Reestablishment		
Baseline (GPT-25B-64)	8D-4P-2T	1800	22.3	0.14	18	9.27	21 (Checkpoint Loading)	CheckpointPeriod/2
FlashRecovery (GPT-25B-64)	8D-4P-2T	≈ 10	0 (Not Required)	0 (Not Required)	18	7.52	9.7 (State Broadcast)	step/2

Table 4: The time costs of every phase of baseline and our method (measured in seconds).

Methods (Model-Params-Devices)	Parallelism	Failure Detection	Checkpoint Saving	Task Restart			State Recovery	Redo Training
				Container Cleanup	Container Create	Communication Reestablishment		
Baseline (GPT-175B-5472)	57D-12P-8T	1800	30	7.2	58	120	60	CheckpointPeriod/2
FlashRecovery (GPT-175B-5472)	57D-12P-8T	≈ 10	0 (Not Required)	0 (Not Required)	18	16.26	4.94	step/2

In response to the reviewers' constructive feedback, we performed two comprehensive ablation studies to quantitatively evaluate the contribution of each key component in our FlashRecovery system. The first set of experiments was conducted on a cluster of 64 devices equipped with Huawei Ascend NPUs, using a 25B LLM as the test benchmark. The second set of experiments was executed on an expanded cluster of 5, 472 devices (also with Huawei Ascend NPUs), testing a 175B LLM to assess scalability. For both configurations, we measured the baseline time consumption (unoptimized) and the optimized time consumption for each system module. The detailed results are presented in Tab. 3 (25B model)

[4] https://gitee.com/ascend/mind-cluster/tree/master/component/clusterd

[5] https://gitee.com/ascend/mind-cluster/tree/master/component/ascend-device-plugin

[6] https://gitee.com/ascend/mind-cluster/tree/master/component/ascend-for-volcano

and Tab. 4 (175B model), demonstrating the efficiency gains achieved by our optimizations. The fault recovery pipeline in our system comprises three critical phases: (1) Fault Detection, (2) Task Restart, and (3) Training Recovery.

1. Fault Detection. In conventional distributed training frameworks lacking dedicated failure detection modules, failures are typically identified passively through communication timeouts during collective operations. This approach suffers from significant latency, which can last up to 30 minutes in PyTorch. To overcome this limitation, we propose active and real-time failure detection. This module performs continuous training state monitoring, enabling immediate identification of hardware and software failures within seconds. As demonstrated in Tab. 3 and 4, our solution reduces failure detection time from approximately 30 minutes (baseline) to about 10 seconds regardless of cluster size.

2. Task Restart. Conventional task restart mechanisms employ indiscriminate destruction and recreation of all containers, followed by the initialization of a new global communication group. As shown in Tab. 3 and 4, this approach exhibits linear $\mathcal{O}(n)$ time complexity, where the baseline system incurs 0.14s, 18s, and 9.27s for container cleanup, container creation, and communication reestablishment respectively on a 64-device cluster. On a 5,472-device cluster, these metrics escalate to 7.2s, 58s, and 120s. In contrast, FlashRecovery eliminates the need for container cleanup entirely, resulting in zero overhead for this phase. Moreover, the time costs of container creation and communication reestablishment remain nearly constant across scaling scenarios.

3. Training recovery. Traditional periodic checkpointing recovery mechanisms require loading checkpoints from disk before resuming training and recomputing all training work since the last checkpoint. The checkpoint load time scales linearly with model size, increasing from 21s (25B model) to 60s (175B model) as shown in Tab. 3 and 4. And statistical analysis shows approximately half of work between checkpoints is redundantly recomputed after each failure. In contrast, FlashRecovery leverages device-to-device communication to transfer model state directly between healthy and replacement devices, bypassing disk I/O entirely. This approach achieves significantly lower recovery overhead, with checkpoint restoration times of 9.7s (25B model) and 4.94s (175B model) [7]. Furthermore, FlashRecovery limits redundant computation to half a training step, reducing wasted work by an order of magnitude compared to traditional methods.

To further validate our experimental results, we have took a video demonstration[8] documenting the complete workflow of experiments in Tab. 3. The video clearly shows the temporal distribution of each module's execution, with all time measurements being consistent with the values reported in Tab. 3.

Reviewer 2

Reviewer Point P 2.1 — The assumption that data parallel redundancy is always sufficient does not always hold. How does the system behave and recover if all replicas in a data parallel group fail simultaneously?

[7] In the scheme of recovering via device-to-device communication, the time of state recovery is related to the size of the model shards instead of the complete size of the model. In our experiments, the 175B model is split into more smaller shards than the 25B model and thus its time of state recovery is shorter.

[8] `https://github.com/cszhj1990/FlashRecovery`

Table 5: The DP configurations of some LLM models.

Models	Num. of Devices	DP Degree	Sources
Llama3	16, 000 H100	128	https://arxiv.org/pdf/2407.21783
DeepSeekV3	2, 048 H800	128	https://arxiv.org/pdf/2412.19437
Megatron-4	6, 144 H100	64	https://arxiv.org/pdf/2406.11704
Megatron-Turing NLG	4480-A100	16	https://arxiv.org/pdf/2201.11990
GLM-130	768-A100	24	https://arxiv.org/pdf/2210.02414
BLOOM	384-A100	8	https://arxiv.org/pdf/2211.05100

Reply:

We appreciate the reviewer's insightful comments regarding complete DP redundancy failure. Our recovery mechanism does necessitate multiple DP replicas, as it fundamentally relies on retrieving at least one complete state from any device within a DP group. Nevertheless, the high degree of DP inherently ensures an extremely low probability of simultaneous failures across all DP ranks ($< 10^{-12}$, a estimation of the probability can be referred to the responses to the 3^{rd} reviewer P 3.3.). This high degree of DP parallelism is consistent with standard practices in large-scale LLM training. Given the huge volume of data that must be processed, data parallelism is an essential component of large-scale LLM training tasks and is often used in conjunction with other parallelization techniques. Tab. 5 presents the DP configurations of several prestigious open-source LLM models.

Furthermore, to ensure robustness against extreme failure scenarios, we maintain a low-frequency periodic checkpointing as a fail-safe mechanism. Although our system can generally recover tasks without relying on checkpoints, this supplementary protective measure ensures the availability of at least one valid checkpoint when failures occur. In addition, checkpointing of our system is asynchronous and introduces negligible overhead under low-frequency operation.

Reviewer Point P 2.2 — The implementation on Huawei Ascend/Kunpeng hardware may limit applicability to mainstream environments. Can FlashRecovery operate effectively with popular hardware/software configurations (e.g., NVIDIA + PyTorch)? What are the main porting challenges?

Reply: Our institute is subject to trade compliance restrictions and thus is unable to legally use the devices from NVIDIA or AMD. In fact, we have already open-sourced most components of our system. Please refer to the response P 1.2 to Reviewer 1 for more details on open source components and the main porting challenges.

Reviewer 3

Reviewer Point P 3.1 — The scope of evaluation is a bit narrow, focused on Huawei's devices but not general devices. Re-run key experiments on mainstream GPUs and at least one public cloud. What will be the differences in experimental results when this method is applied to clusters equipped with NVIDIA or AMD GPUs?

Reply: Our institute is subject to trade compliance restrictions and thus is unable to legally use the devices from NVIDIA or AMD. In fact, we have already open-sourced most components of our system. Please refer to the response P 1.2 to Reviewer 1 for more details on open-source components. While current trade compliance restrictions prevent legal access to NVIDIA hardware for validation, we argue that experimental results on mainstream GPUs and public clouds are similar to those on Huawei Ascend / Kunpeng hardwares. From a series of experiments in our paper, it can be concluded that the performance gains of our system mainly come from the optimization for the process of recovery, and most of them are not device dependent.

Reviewer Point P 3.2 — The baselines are weak. No state-of-the-art research work has been compared quantitatively. Compare the work with the state-of-the-art work in experiments.

Reply: Tab. 1 presents a systematic comparison of fault recovery approaches across Swift, JIT, Parcae, and our FlashRecovery system, evaluating key dimensions including fault detection, task restart, and training recovery mechanisms. Detailed analysis are available in our response P 1.1 to Reviewer 1.

Reviewer Point P 3.3 — Over-claimed words at some places such as "Scale-independent" and "Checkpoint-free". For example, checkpoints are still needed in circumstances and I don't think it works for TP or EP. How would this method apply to TP and EP models?

Reply:

We sincerely appreciate the constructive feedback from the reviewers on our description. In response, we provide the following clarifications.

In our paper, we analyze the total overhead during task restart, which includes the overhead of (1) container management, (2) communication group establishment, and (3) I/O of initialization. Among these procedures, the time complexity of container management and I/O of initialization can be optimized to a constant (i.e., $\mathcal{O}(1)$) via a local container recreation. Although the time complexity of the establishment of communication groups can be optimized from $\mathcal{O}(n)$ to $\mathcal{O}(\frac{n}{p})$ in a parallel style, it remains a linear time complexity when $n > p$. However, the elapsed time consumed by this procedure is short ($<$ 1 second even when operating on 10, 000 devices, Tab. 1 in our paper), which is acceptable in most cases and can be ignored compared with other procedures. Therefore, our solution is "Scale-independent" during task restart for the vast majority of training scales.

The term "Checkpoint-free" denotes a recovery mechanism wherein the model state can be retrieved directly from other DP replicas through device-to-device communication, obviating the need for checkpoint restoration during failure recovery. Checkpoints are retained solely to address extreme edge cases where all devices within a DP group experience simultaneous failure. In large-scale LLM training scenarios, the degree of DP typically ranges from dozens to hundreds. This configuration results in an extremely low probability (less than one in dozens of trillion [9]) of simultaneous failure across all devices within a DP group, rendering such an event virtually impossible in practice.

Our checkpoint-free recovery mechanism is fundamentally based on DP redundancy and is therefore not applicable when using TP or EP exclusively. However, modern LLM training systems universally adopt hybrid parallelism strategies (e.g., configurations in Tab. 5). FlashRecovery fully supports these practical configurations, including DP+TP and DP+EP. As demonstrated in Fig. 3 in our paper, these hybrid approaches inherently maintain model state replicas across devices, satisfying the redundancy

[9]Suppose that the probability of a device failing is 0.1, then the probability of 12 devices of a DP group fail simultaneously is one in a trillion, i.e., 10^{-12}.

requirement for our recovery mechanism. In our ablation studies(P 1.3) and supplementary video materials[10], we validate this through training in the 25B and 175B model using combined DP + TP + PP parallelism. In reality, a training task without DP is unlikely to exist in practice because a training setting without DP would greatly degrade training efficiency.

We hope that the above responses can dispel the concerns of the reviewers.

[10] https://github.com/cszhj1990/FlashRecovery