Characterization of Large Language Model Development in the Datacenter

Qinghao Hu* 1,2, Zhisheng Ye* 1,3, Zerui Wang* 1,4, Guoteng Wang1, Meng Zhang1,2, Qiaoling Chen1,2, Peng Sun1,5, Dahua Lin1, Xiaolin Wang3, Yingwei luo3, Yonggang Wen2, Tianwei Zhang2
Background
In the Era of Large Language Models

**InternLM**

**Qwen**

**OpenAI**

**ChatGPT-4 Turbo**

**Gemini**

**Brainstorming**

Planning

Coding

Writing

Write an email to apply for a PhD with a professor.

Subject: Request for PhD Position under Your Supervision

Dear Professor [Last Name],

I hope this message finds you well. My name is [Your Full Name], and I am writing to express my keen interest in pursuing a PhD under your esteemed supervision at [University Name]. I have recently completed my Master of Science degree in [Field of Study] from [Name of University], where my research focus was on [Briefly mention thesis topic]. Throughout my academic journey, I have nurtured a deep passion for [related field/discipline] which has led me to explore various aspects of it through research projects, internships, and publications. During this time, I came across several of your notable works, including "[Title of Publication]" and "[Title of Research Project]," both of which left a profound impact on my intellectual curiosity.

The reason behind my eagerness to pursue doctoral studies under your guidance stems not only from your groundbreaking contributions to the field but also because our mutual interests seem to align significantly. The ongoing work at your lab, particularly on [specific project/area of interest], resonates strongly with my long-term career goals and aspirations.

Enclosed herewith, please find:
In the Era of Large Language Models

Developing LLMs demands extensive computational resources and can span several months.

**LLaMA2 Model Family**
(7B, 13B, 34B, 70B)

- **Meta**
  - 16,000 NVIDIA A100 GPUs
  - Several months development

- **Google AI**
  - 6,144 TPU-v4
  - 2 months pretraining

**PaLM 540B**

2 months pretraining
Motivation

• What are the characteristics of workloads during the development of LLMs?

• What are the new datacenter requirements for running LLMs compared to prior DL workloads?

• How to tailor system software for LLMs?
LLM Development Pipeline: An Overview

1. **Data Preparation**
   - Process Pipeline:
     - Collection
     - Curation
     - Tokenization
   - Data Stores:
     - Pretraining Datasets
     - Alignment Datasets

2. **Pretraining**
   - Training Recipe:
     - Model Selection
     - Data Sampling
     - Hyperparameters
   - Placement Strategy:
     - Data, Pipeline, Tensor Parallelisms

3. **Alignment**
   - Placement Strategy
     - Fine-tuning
     - RLHF
     - Prompt Engineering

4. **Evaluation**
   - Multi-task
   - Multi-metric
   - Distributed Eval.

5. **Deployment**
   - Deployment is not discussed

[Diagram showing the flow from Data Preparation to Deployment with various sub-processes and strategies.]
Acme: GPU Datacenter of Shanghai AI Laboratory

- Two GPU Clusters Dedicated to LLM: Seren & Kalos

<table>
<thead>
<tr>
<th>Cluster</th>
<th>#CPUs/node</th>
<th>#GPUs/node</th>
<th>Mem(GB)</th>
<th>Network</th>
<th>#Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seren</td>
<td>128</td>
<td>8 × A100-80GB</td>
<td>1,024</td>
<td>1×200Gb/s</td>
<td>286</td>
</tr>
<tr>
<td>Kalos</td>
<td>2,048</td>
<td>5 × 200Gb/s</td>
<td>2,048</td>
<td>5×200Gb/s</td>
<td>302</td>
</tr>
</tbody>
</table>

Totally **4704 A100 GPUs** interconnected by NVLink and InfiniBand

- Model Scale: InternLM (LLaMA like architecture)from 7B~123B

More models see: [https://huggingface.co/internlm](https://huggingface.co/internlm)
Trace Overview

- **Collection Period**: traces are collected from March to August 2023
- **Number of Jobs**: a total of **684K GPU jobs** and **410K CPU jobs** in Seren and Kalos
- **Trace Sources**: (1) Job Metadata, (2) Hardware Monitor Data, (2)Runtime Log, (4) Profiling Data

<table>
<thead>
<tr>
<th>Datacenter</th>
<th>Acme Shanghai AI Lab</th>
<th>Helios SenseTime</th>
<th>PAI Alibaba</th>
<th>Philly Microsoft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>2023</td>
<td>2020</td>
<td>2020</td>
<td>2017</td>
</tr>
<tr>
<td>Duration</td>
<td>6 months</td>
<td>6 months</td>
<td>2 months</td>
<td>3 months</td>
</tr>
<tr>
<td>#Jobs</td>
<td>1.09M</td>
<td>3.36 M</td>
<td>1.25 M</td>
<td>113K</td>
</tr>
<tr>
<td>Workload</td>
<td>LLM Development</td>
<td>General DL workloads</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU Model</td>
<td>A100</td>
<td>1080Ti/V100</td>
<td>T4/P100/V100</td>
<td>12GB/24GB</td>
</tr>
<tr>
<td>Total #GPUs</td>
<td>4,704</td>
<td>6,416</td>
<td>6,742</td>
<td>2,490</td>
</tr>
</tbody>
</table>

Trace available at [https://github.com/InternLM/AcmeTrace](https://github.com/InternLM/AcmeTrace)
Datacenter Characterization
LLMs versus Prior DL Workloads

- Shorter GPU Job Duration

**Explanation**

- More advanced hardware
- Abundant resources of each job (Avg. 5~20 GPU)
- Extensive small-scale jobs
- Many failed jobs (~40%)

**Key Insight:** Need for a fault-tolerant system
LLMs versus Prior DL Workloads

- **Polarized GPU Utilization**
  - LLM workloads are either *almost entirely idle* or *fully active*

**Explanation**
- The computationally intensive nature of LLMs
- Many jobs fail at initialization without using any GPUs

**Key Insight:** GPU-sharing techniques may not be optimal for LLM development.
High-skewed Workload Distribution of LLMs

- **Kalos (b, orange)**: Top 5% of jobs, using >256 GPUs, account for ~96% of GPU time.
- **Seren (b, blue)**: Top 2% of jobs, using >64 GPUs, account for ~75% of GPU time.

**Key Insight**: Design scheduler optimizations for LLM clusters considering the skew in GPU usage.
Infrastructure Utilization Patterns

Higher GPU utilization in LLM development

(a) Median **SM activity ~40%** in both clusters. (20% in PAI)
(b) The majority of GPUs consume **>75% GPU memory** in Kalos (<25% in PAI)
Workload Profiling

- Pretraining
- Evaluation
Profiling Pretraining Workloads: Methodology

**Workloads**
- 123B InternLM using 2,048 A100 GPUs

**Frameworks and Strategies**
- **InternEvo-v1**: 3D Parallelism (Megatron-LM)
  - with pipeline parallelism=4, tensor parallelism=8
- **InternEvo-v2**: Hierarchical ZeRO [1]
  - Parameter sharding limited in subgroups of 64 GPUs
  - Recomputing

Profiling Pretraining Workloads: GPU SM Utilization

Key Improvements of *InternEvo V2*
- Reduced GPU idle/lower period
- Higher peak SM utilization

Optimizations Under the Hood
- Selective Communication Overlap
- Effective hybrid parallelism strategy

See the *InternEvo* paper[1] for more details

Profiling Pretraining Workloads: GPU Memory Footprint

**Key Insight:**
A specialized partitioning mechanism is needed.

---

Profiling Evaluation Workloads: High GPU Idle Rate

- Source1 ➔ Model loading and data processing overhead
- Source2 ➔ Metric computing overhead. Up to 30 mins
- An example: 7B model evaluation on HumanEval

**Key Insight:** Improvement in the evaluation process of LLMs in **system-level**.

![Graph showing GPU time idle and system-level improvement](image-url)
Failure Analysis
Impact of Job Failures: Typical Failure Recovery Process

- LLM Jobs suffer from early job termination due to various failures
Impact of Job Failures: GPU Wastage and Progress Loss

From Checkpointing (①)
• Training get stuck during checkpoint saving (purple chunk)

From Failure Recovery (②–④)
• GPU time wastage (red chunk)
• Training progress loss (red x)

Key Insight: Build a system that minimizes the failure recovery overhead
Sources of Job Failures

- Failures Happen across the Stacks

User Script

```python
import torch
from models import InternLM

def train(args):
    model = InternLM()
    for epoch in range():
        train_one_epoch(model)

if __name__ == '__main__':
    train(args)
```

Framework

- InternEvo
- PyTorch
- Eval Framework

Infrastructure

- Scheduler
- Storage
- Network
- CPUs, Memory, NVLink
Job Failure Analysis

Workload Composition
- 1.3k pretraining jobs
- 31K evaluation jobs
- 550 debug jobs

Data Sources
- Runtime log: stderr & stdout
- Hardware monitoring data

Methodology
- Identify and categories failures of the failed jobs

<table>
<thead>
<tr>
<th>Category</th>
<th>Reason</th>
<th>Num</th>
<th>Avg. GPU Demands</th>
<th>Avg. TF (mins)</th>
<th>Total %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure</td>
<td>NVLinkError</td>
<td>54</td>
<td>800</td>
<td>868.1</td>
<td>30.25%</td>
</tr>
<tr>
<td></td>
<td>CUDAError</td>
<td>21</td>
<td>847</td>
<td>923.2</td>
<td>15.77%</td>
</tr>
<tr>
<td></td>
<td>NodeFailure</td>
<td>16</td>
<td>712</td>
<td>1288.8</td>
<td>14.30%</td>
</tr>
<tr>
<td></td>
<td>ECCError</td>
<td>12</td>
<td>680</td>
<td>1303.4</td>
<td>11.00%</td>
</tr>
<tr>
<td></td>
<td>NSError</td>
<td>12</td>
<td>758</td>
<td>549.6</td>
<td>4.53%</td>
</tr>
<tr>
<td></td>
<td>ConnectionError</td>
<td>147</td>
<td>29</td>
<td>51.9</td>
<td>3.44%</td>
</tr>
<tr>
<td></td>
<td>S3StorageError</td>
<td>10</td>
<td>422</td>
<td>2317.8</td>
<td>2.12%</td>
</tr>
<tr>
<td></td>
<td>NCCLTimeoutError</td>
<td>6</td>
<td>596</td>
<td>159.7</td>
<td>0.50%</td>
</tr>
<tr>
<td></td>
<td>NCCLRemoteError</td>
<td>3</td>
<td>1152</td>
<td>50.5</td>
<td>0.15%</td>
</tr>
<tr>
<td>Framework</td>
<td>DataloaderKilled</td>
<td>6</td>
<td>445</td>
<td>1580.6</td>
<td>4.38%</td>
</tr>
<tr>
<td></td>
<td>AttributeError</td>
<td>67</td>
<td>228</td>
<td>67.8</td>
<td>3.90%</td>
</tr>
<tr>
<td></td>
<td>OutOfMemoryError</td>
<td>14</td>
<td>572</td>
<td>323.8</td>
<td>3.28%</td>
</tr>
<tr>
<td></td>
<td>RunTimeError</td>
<td>65</td>
<td>441</td>
<td>66.4</td>
<td>1.72%</td>
</tr>
<tr>
<td></td>
<td>AssertionError</td>
<td>105</td>
<td>413</td>
<td>41.7</td>
<td>1.24%</td>
</tr>
<tr>
<td></td>
<td>ValueError</td>
<td>33</td>
<td>387</td>
<td>9.9</td>
<td>0.16%</td>
</tr>
<tr>
<td></td>
<td>ZeroDivisionError</td>
<td>5</td>
<td>499</td>
<td>14.5</td>
<td>0.03%</td>
</tr>
<tr>
<td></td>
<td>ModelLoadingError</td>
<td>104</td>
<td>8</td>
<td>2.6</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>DatasetLoadingError</td>
<td>5</td>
<td>1</td>
<td>1.6</td>
<td>0.00%</td>
</tr>
<tr>
<td>Script</td>
<td>FileNotFoundError</td>
<td>568</td>
<td>21</td>
<td>14.2</td>
<td>2.83%</td>
</tr>
<tr>
<td></td>
<td>OSError</td>
<td>266</td>
<td>8</td>
<td>9.6</td>
<td>0.28%</td>
</tr>
<tr>
<td></td>
<td>TypeError</td>
<td>620</td>
<td>18</td>
<td>0.9</td>
<td>0.06%</td>
</tr>
<tr>
<td></td>
<td>Others</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.08%</td>
</tr>
</tbody>
</table>
System for LLM

Fault-tolerant LLM Pretraining

Timely Feedback for Evaluation
Fault-tolerant Pretraining: Technique 1

Asynchronous Checkpointing
1. Store model states in host memory
2. A background process asynchronously save them to the storage

Improvements (blue line vs orange line)
• $58 \times$ checkpointing speedup ➔ less progress loss
Technique 2: LLM-assisted Diagnosis and Recovery

Job Start

Real-time Log Compression

Failure Occurs

Compressed Log & stderr

Rule-based Diagnosis

Match

Recoverable?

Yes

Automatic Recovery

No

Manual Recovery

Log Agent

Rule Writer

Failure Agent

Query Engine

Recovery Process

Failure Detection

Hint

Time
Timely Feedback for Evaluation: System Design

1. Reducing I/O Overhead
   - Download models once and load via PCIe

2. Async Metric Computation (CPU Only)

3. Batch Trails to Improve Throughput

**Improvement:** Reducing the makespan by 1.3 ~ 1.8 times
Characterization of Large Language Model Development in the Datacenter

Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaochen Bai, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, Yonggang Wen, Tianwei Zhang

1Shanghai AI Laboratory 2S-Lab, Nanyang Technological University 3NTU 4Peking University 5Shanghai Jiao Tong University 6SenseTime Research 7CUHK

Abstract
Large Language Models (LLMs) have presented impressive performance across several transformative tasks. However, it is non-trivial to efficiently utilize large-scale cluster resources to develop LLMs, often riddled with numerous challenges such as frequent hardware failures, intricate parallelization strategies, and imbalanced resource utilization. In this paper, we present an in-depth characterization study of a six-month LLM development workload trace collected from our GPU datacenter Acme. Specifically, we investigate discrepancies between LLMs and prior task-specific Deep Learning (DL) workloads, explore resource utilization patterns, and identify the impact of various job failures. Our analysis summarizes hurdles we encountered and uncovers potential opportunities to optimize systems tailored for LLMs. Furthermore, we introduce our system efforts: (1) fault-tolerant pretraining, which enhances fault tolerance through LLM-involved failure diagnosis and automatic recovery; (2) decoupled scheduling for evaluation, which achieves timely performance feedback via trial decomposition and scheduling optimization.

1 Introduction
Over the years, advances in LLMs have attracted significant attention from both academia and industry owing to their impressive performance and capabilities, such as ChatGPT [2] and GitHub Copilot [3]. However, due to their immense model sizes and extensive data demands, training such models necessitates a substantial computational infrastructure with thousands of accelerators [27, 68]. Hence, it is a common practice for tech companies and cloud providers to build large-scale GPU clusters to facilitate LLM development, especially after the popularity of ChatGPT. Nevertheless, it is non-trivial to perform efficient LLM development on such high-cost infrastructure. Developers often confront numerous issues and challenges, including frequent hardware failures [64, 96], intricate parallelization strategies [68, 113], unstable training progress [1, 110], long queuing delay [104], etc.

Developing LLMs is closely intertwined with the support of GPU clusters in various aspects. A thorough analysis of cluster workloads is essential for comprehending challenges and uncovering opportunities in designing systems tailored for LLMs. However, many conclusions and implications from existing DL workloads analysis works [38, 45, 97], conducted before the rise of LLMs, are not applicable to LLM development. This is primarily due to the divergent characteristics and requirements of LLMs:

(1) Paradigm Transition: DL workloads generally follow a task-specific paradigm that trains the model on domain-specific data to tackle a particular task (e.g., translation [18]). In contrast, LLMs follow an emerging paradigm that performs self-supervised training on broad data to generate a foundation model [39] and further adapts to a wide range of downstream tasks. This shift signifies a substantial divergence in the model development pipeline (e.g., pretraining [15], alignment [37]) and workload characteristics from prior DL workloads (§2.1).

(2) Tailored Software Stack: To accommodate the enormous model size of LLMs, a series of systems implement advanced techniques to optimize the execution of LLMs. For instance, Deepspeed [79], Megatron [68] and Alpa [113] accelerate the training via hybrid parallelism or state-sharding optimizer. As for model serving, Orca [104] and vLLM [51] improve throughput via iteration scheduling or memory management.

(3) Unified Architecture: Prior DL workloads usually employ various model architectures (e.g., CNN [54], RNN [18]) to address diverse tasks. In contrast, LLMs commonly embrace the Transformer [91] architecture, like BERT [31], GPT-3 [20], LLaMA [91] and PaLM [27]. The architectural homogeneity suggests a high level of uniformity in the LLM development pipeline and similarity across different datacenters.

To bridge this gap, we present an in-depth study of our operational experiences in the datacenter Acme of Shanghai AI Laboratory. It houses two distinct clusters, Seren and 8x2a/8, dedicated to LLM development and equipped with 4,704 A100 GPUs in total. Our analysis draws upon traces collected over a six-month period from March to August 2023, encompassing scheduler logs, infrastructure monitoring data, failure logs, and fine-grained profiling data. Our key findings and identified challenges can be summarized as follows:

• Shorter Job Duration and Unfair Queuing Delay: In contrast to the common stereotype that LLM workloads are usually long-term, the workloads in our datacenter exhibit 2.7–12.8× shorter average job duration compared to the DL workloads in previous traces [38, 45, 97]. This can be

More in the Paper

- More profiling results:
  - 123B model with 1k GPUs
  - Profiling MoE models
- Statistics on the workload categories
- Detailed failure analysis
- Environment impact of LLM development in Acme
**Datacenter Characterization**
LLM workload & resource utilization

**Resource Inefficiencies**
GPU time wastage of evaluation & pretraining workloads

**Failure Impacts**
Failures severely affect LLM development

**Systems Efforts for LLM**
Fault-Tolerant Pretraining System & Timely Feedback Evaluation

**InternLM**: https://github.com/InternLM

**Email**: wangzerui@pjlab.org.cn