Internal Report - Comparative Analysis of Distributed LLM Training Projects

Chris Esposo
Georgia Institute of technology
cesposo3@gatech.edu

9-28-2025

Abstract

Training large language models (LLMs) at scale demands distributed systems that can handle massive model states, high communication loads, and frequent failures. We survey representative projects spanning academic research, industrial open-source frameworks, and community initiatives, and assess each along five axes: (1) training scalability and parallelism, (2) resource allocation strategies, (3) fault tolerance, (4) real-world usage and benchmarking, and (5) codebase maturity and usability. We then compare these findings to our proposed project, *Democratizing Large Language Model Training in Open Science Environments*, and evaluate whether it constitutes a non-trivial academic contribution competitive with top ML systems venues.

1 Introduction

We examine how contemporary projects confront the central challenges of distributed LLM training: scaling across many devices, allocating heterogeneous resources, tolerating faults and churn, achieving credible real-world performance, and providing usable, maintainable code. Our goal is to position our proposal within this landscape and to articulate where it is novel, feasible, and impactful.

2 Academic Research Efforts

Recent research targets environments beyond traditional, homogeneous data centers, emphasizing heterogeneity, limited bandwidth, and unreliability.

SWARM Parallelism (ICLR 2022)

Ryabinin *et al.* introduced a model-parallel algorithm for *"swarms"* of poorly connected, heterogeneous, unreliable devices.[1] SWARM forms temporary randomized pipelines that are rebalanced on failures and reduces communication with compression-aware Transformer modifications. They trained a 1.1B-parameter Transformer (about 13B effective via weight sharing) on preemptible T4s with < 400 Mb/s network—a proxy for spot or volunteer nodes. *Scalability:* ∼1.1B params across low-end GPUs. *Resource allocation:* stochastic pipeline wiring with dynamic rebalance. *Fault tolerance:* on-the-fly pipeline restructuring. *Usage/Maturity:* research prototype (Hivemind-based) demonstrating feasibility more than SOTA throughput.

Decentralized Foundation Model Training (NeurIPS 2022)

Yuan *et al.* study training large models over decentralized, heterogeneous, low-bandwidth networks.[2] They propose a scheduler that partitions data/model-parallel *tasklets* across nodes using a cost model and evolutionary search; in geo-distributed simulations (8 cities, 3 continents) they report up to 4.8× speedups over prior baselines under extreme WAN conditions. *Focus:* explicit scheduling for heterogeneous/wide-area settings. *Maturity:* prototype-level with insights applicable to volunteer scenarios.

SpeedLoader (NeurIPS 2024)

Zhang & You redesign sharded training I/O for heterogeneous nodes, improving GPU utilization when offloading to CPU/NVMe.[3] They report MFU increases from ∼36% to ∼51% and 3×–30× speedups over prior offloading approaches in tested setups. *Focus:* intra-node heterogeneity and transfer scheduling; *fault tolerance:* not central; *maturity:* research-stage optimizations likely to influence production libraries.

Other Notable Threads

Work on asynchronous/Local-SGD for heterogeneous clusters, memory-efficient pipeline parallelism, microcheckpointing for low-overhead recovery, and privacy-preserving training (e.g., MPC) addresses specific subproblems but does not yet coalesce into a unified, open system for wide-area, untrusted training.

[1] Overview and paper: https://openreview.net

[2] Paper and reviews: https://openreview.net

[3] Proceedings: https://proceedings.neurips.cc

3 Industrial Open-Source Frameworks

DeepSpeed (Microsoft)

DeepSpeed enables *extreme-scale* training with ZeRO sharding (Stages 1–3) and *3D parallelism* (data, tensor, pipeline) and has supported trillion-parameter regimes.[4] ZeRO-Offload/Infinity extends memory with CPU/NVMe; DeepSpeed Elastic supports dynamic membership for spot/preemptible nodes. It is widely adopted (e.g., Megatron-DeepSpeed for BLOOM-176B). *Strengths:* performance, features, documentation. *Limitations:* assumes relatively homogeneous clusters; heterogeneity is largely manual.

Megatron-LM / Megatron-Core (NVIDIA)

Megatron introduced intra-layer tensor parallelism and high-performance pipeline parallelism; Megatron-Core refactors these into modular components.[5] It underpins many massive models (e.g., MT-NLG 530B) and adds resiliency extensions (e.g., NVRx) for failure detection/recovery. *Strengths:* maximal throughput on NVLink/NVSwitch clusters; *Limitations:* static partitioning; heterogeneity and WAN are not focal.

Alpa (JAX/OpenXLA)

Alpa automatically composes data, operator (tensor), and pipeline parallelism from a single-device JAX function using XLA graph partitioning.[6] It demonstrated strong scaling (dozens to 1000 GPUs with Ray) but targets fixed, homogeneous clusters; fault tolerance is via conventional checkpointing; the standalone project has been upstreamed/influential in OpenXLA.

Fully Sharded Data Parallel (FSDP, PyTorch)

FSDP shards parameters/optimizer states across ranks and all-gathers them just-in-time, enabling very large models within PyTorch core.[7] It is widely used (e.g., LLaMA training). *Strengths:* native, mature, interoperable; *Limitations:* expects roughly homogeneous shards; elasticity exists but is non-trivial.

4 Community-Driven and Collaborative Initiatives

Petals

Petals distributes *inference and low-bandwidth fine-tuning* over volunteer GPUs by assigning layers across peers with dynamic, fault-tolerant routing.[8] It has run BLOOM-176B and LLaMA-70B across the public swarm. *Strengths:* WAN-aware, churn-tolerant; *Limitations:* not aimed at from-scratch training of very large models.

Colossal-AI (HPC-AI Tech)

Colossal-AI consolidates hybrid parallelism (1D/2D/2.5D/3D tensor, pipeline, DP), memory systems (Gemini/PatrickStar), and FP8 into a cost-focused PyTorch stack.[9] It targets accessible performance on modest clusters. *Strengths:* flexible, evolving feature set; *Limitations:* primarily synchronous clusters; advanced fault tolerance and WAN heterogeneity are works in progress.

OpenLLM (BentoML)

OpenLLM simplifies *serving* open models via a clean API and Kubernetes-ready packaging.[10] It is orthogonal to distributed *training*, but relevant to democratizing LLM *usage*.

5 Structured Comparison of Key Features

6 Comparison to Our Proposed Project

Our project targets *wide-area, heterogeneous, and untrusted* environments—a qualitatively different regime than most industrial frameworks and broader in scope than existing community efforts.

Vision and scope. We aim to pool disparate resources (volunteer GPUs, multi-institution clusters) to train and fine-tune large models under dynamic availability, heterogeneity, and explicit trust constraints. This extends Petals-style WAN operation from inference/adapter training to *full or substantial* training while incorporating security guarantees largely absent from today's systems.

Training scalability. We do not expect to beat InfiniBand clusters on raw throughput—that is not our goal. Instead, we focus on *feasible* training at useful scales on commodity networks. SWARM and the NeurIPS'22 scheduler indicate that training billions of parameters over slow links is practical with appropriate algorithmic concessions (e.g., randomized pipelines, communication-aware scheduling). A compelling milestone is end-to-end training of a multi-billion-parameter model across tens of heterogeneous GPUs with acceptable time-to-accuracy.

Resource allocation via dynamic matching. We propose adapting Gale–Shapley deferred acceptance to continuously match *tasks* (e.g., pipeline stages, tensor shards, microbatches) with *nodes* according to evolving preferences (capabilities, bandwidth, reliability). Unlike heavy global optimizers

[4] Docs/overview: https://www.deepspeed.ai | Project: https://github.com/microsoft/DeepSpeed

[5] Repo: https://github.com/NVIDIA/Megatron-LM

[6] Repo: https://github.com/alpa-projects/alpa

[7] Intro/tutorials: https://pytorch.org/docs/stable/fsdp.html | HF guides: https://huggingface.co/docs/transformers/fsdp

[8] Project/paper: https://petals.ml | https://neurips.cc

[9] Project: https://github.com/hpcaitech/ColossalAI

[10] Project: https://github.com/bentoml/OpenLLM

(ILP/evolutionary search) or purely heuristic swarms, stable matching offers a light-weight reconfiguration primitive that can react to churn while avoiding unstable allocations. The research questions are its convergence/overhead under churn and its impact on end-to-end MFU and throughput relative to baselines.

Fault tolerance. In WAN volunteer settings, faults are the norm. We plan to combine: (i) micro- and layer-boundary checkpointing, (ii) redundant or speculative execution for critical stages, and (iii) elastic world-size adaptation with rapid re-matching when nodes arrive/depart. The target property is *graceful degradation*: progress continues with bounded slowdown under churn, avoiding job-wide aborts.

Security and privacy. We emphasize privacy-preserving training across untrusted nodes via selective cryptography (secure aggregation, secret sharing of sensitive states, limited homomorphic operations where tractable). Full homomorphic training remains impractical, but hybrid designs—e.g., encrypting gradients or masking activations where leakage risk is highest—can provide meaningful guarantees at tolerable overheads. This dimension is largely unexplored in mainstream LLM training frameworks and is a differentiator of our work.

Evaluation plan. A credible evaluation should demonstrate: (1) WAN training progress on real heterogeneous nodes (10–50 GPUs), (2) throughput/efficiency under synthetic churn, (3) scheduler ablations against static and heuristic baselines, and (4) security overheads vs. privacy gains on representative workloads. Even with smaller models, such results would constitute a substantive systems contribution.

Critical assessment. The project is ambitious: scheduling, elasticity, and security could each be a paper. For a top-tier venue, we will likely foreground one core contribution (e.g., dynamic stable matching for WAN LLM training) with a working system and rigorous evaluation, while scoping security to *selective* but demonstrable protections. Over time, this line of work admits multiple publishable slices (scheduler, FT protocols, secure aggregation at scale).

7 Conclusion

Industrial frameworks (DeepSpeed, Megatron, FSDP) excel on homogeneous clusters and now offer partial elasticity, but do not target WAN heterogeneity or untrusted settings. Community systems (Petals) prove WAN feasibility for inference and light fine-tuning; academic work (SWARM, decentralized schedulers, SpeedLoader) provides building blocks. Our project advances this frontier by (i) training at WAN scale with (ii) dynamic matching for heterogeneity/churn and (iii) explicit security guarantees. Even a partial realization—with end-to-end training on a heterogeneous WAN and solid ablations—would be a non-trivial contribution to ML systems and a strong basis for top-venue submission.

Notes on sources. Representative project pages and papers: DeepSpeed (https://www.deepspeed.ai); Megatron-LM (https://github.com/NVIDIA/Megatron-LM); Alpa (https://github.com/alpa-projects/alpa); FSDP docs (https://pytorch.org/docs/stable/fsdp.html); Petals (https://petals.ml / NeurIPS); Colossal-AI (https://github.com/hpcaitech/ColossalAI); SWARM / Decentralized schedulers / SpeedLoader (OpenReview/NeurIPS proceedings).

Table 1: Side-by-side comparison across five criteria. Abbrev.: DP=data parallel; TP=tensor parallel; PP=pipeline parallel; WAN=wide-area network.

Project	Training scalability & parallelism	Resource allocation strategies	Fault tolerance mechanisms	Real-world usage & benchmarks	Maturity & usability
DeepSpeed	3D DP+TP+PP; ZeRO (Stages 1–3) enables trillion-scale; offloading to CPU/NVMe extends capacity	Assumes homogeneous clusters; manual configuration of degrees; offload widens memory, not compute heterogeneity	Checkpointing + DeepSpeed Elastic (dynamic membership, rendezvous, resume)	Used in MT-NLG, BLOOM-176B (via Megatron-DeepSpeed); strong scaling on 1000s of GPUs	Production-grade, rich docs; plug-in style engine for PyTorch
Megatron(-Core)	State-of-the-art TP/PP kernels; mixture-of-experts, context/expert parallelism; extreme strong scaling	Static splits tuned to NVLink/NVSwitch clusters; limited heterogeneity/WAN focus	Periodic checkpoints; resiliency extensions (e.g., NVRx) for detection/recovery	Backbone for many frontier models; highest per-GPU throughput on NVIDIA platforms	Mature performance toolkit; improving modularity/docs
Alpa	Auto-composes DP+TP+PP from JAX/XLA; near-linear scaling (dozens–1000 GPUs)	Compiler-driven placement on fixed, mostly homogeneous devices; cluster known at compile time	Standard checkpointing; restarts on failures (no elastic layer)	Research deployments/demos; influence via OpenXLA auto-sharding	Research-grade; elegant in JAX, limited outside
FSDP (PyTorch)	Shards params/optim states; often combined with TP/PP for very large models	Evenly sharded ranks; limited built-in heterogeneity handling; optional CPU offload	PyTorch Elastic + sharded checkpoints; elasticity requires careful setup	Widely used in practice (e.g., LLaMA); strong ecosystem support	Core, stable feature; configuration can be nuanced
Petals	WAN pipeline across volunteer nodes for inference and adapter tuning; partial training feasible	Dynamic layer-to-node assignment with capacity-aware load balancing via DHT/coordinator	Designed for churn: heartbeats, rerouting, redundancy; layer-boundary recovery	Operational public swarm (BLOOM/LLaMA) with interactive latencies	Active open-source; specialized but user-friendly
Colossal-AI	Hybrid parallelism (1D/2D/2.5D/3D TP, DP, PP); FP8 and memory systems for efficiency	Config-driven; planner/booster apply parallel strategies; cluster-oriented, not WAN-heterogeneous	Standard checkpoint/restart; elastic/FT under active development	Adopted across startups/labs; strong cost/perf focus	Moderately mature; good docs; rapid evolution
OpenLLM	Serving-focused; not a distributed training framework	Orchestrates inference replicas; no training scheduler	Fault tolerance via cloud-native restarts; stateless serve	Gaining adoption for self-hosting LLM APIs	Production-ready for deployment (orthogonal to training)