Distributed LLM Inference

on Systems of Small GPU Chips

Challenges, Strategies, and Engineering Trade-Offs

February 2026

Table of Contents

1. Introduction

Large language models (LLMs) have grown to scales where a single GPU can no longer hold the full model in memory. While high-end accelerators like the NVIDIA A100 (80 GB) or H100 can host medium-sized models, many practitioners are turning to clusters of smaller, more affordable GPUs—such as consumer-grade cards with 8–24 GB of VRAM—to perform inference at lower cost. This architecture introduces a rich set of engineering challenges that span memory management, inter-device communication, scheduling, and numerical precision.

This report provides an in-depth treatment of the key challenges encountered when distributing LLM inference across many small GPU chips connected by high-speed links, along with the most widely adopted solutions in both research and production systems.

2. Fundamental Constraints

2.1 Memory Capacity per Chip

The dominant constraint in distributed inference is per-chip memory. An LLM's memory footprint is determined by its parameter count, the chosen numerical precision, and the KV-cache required for autoregressive decoding. A 70-billion-parameter model stored in FP16 occupies roughly 140 GB for weights alone—far exceeding any single consumer GPU. The KV-cache grows linearly with batch size and sequence length, adding further pressure.

Model Size	FP16 Weight Memory	GPUs Needed (24 GB)	GPUs Needed (12 GB)
7B	~14 GB	1	2
13B	~26 GB	2	3
70B	~140 GB	6–8	12–16
405B	~810 GB	34–40	68–80

2.2 Interconnect Bandwidth and Latency

Distributed inference transforms a compute-bound problem into a communication-bound one. Every forward pass requires exchanging activations or partial results between chips. The ratio of computation to communication—the arithmetic intensity—determines whether the GPUs spend more time computing or waiting on data transfers.

High-speed links such as NVLink (up to 900 GB/s bidirectional on H100), NVSwitch, and custom interconnects (e.g., Google's ICI for TPUs) are designed to mitigate this bottleneck. However, commodity setups using PCIe (32–64 GB/s per link) or even networked solutions like InfiniBand (up to 400 Gb/s per port) introduce significantly higher latencies and lower bandwidths per chip, making communication optimization critical.

2.3 The Inference Workload Profile

LLM inference has two distinct phases with very different computational profiles. The prefill phase processes the entire input prompt in parallel and is compute-bound, resembling a training forward pass. The decode phase generates tokens one at a time autoregressively, is memory-bandwidth-bound, and underutilizes GPU compute. This asymmetry means that a parallelism strategy ideal for prefill may be suboptimal for decode, and vice versa.

3. Parallelism Strategies for Inference

3.1 Tensor Parallelism (TP)

Tensor parallelism splits individual weight matrices across GPUs. For a transformer's attention and feed-forward layers, the large matrix multiplications are partitioned column-wise or row-wise so each GPU computes a shard of the output. The partial results are then combined via an *all-reduce* collective operation.

This is the most common strategy when GPUs are connected by high-bandwidth links (e.g., NVLink within a single node). It keeps latency low because every GPU participates in every layer. However, TP requires an all-reduce at every transformer block—typically two per layer (one after the attention projection, one after the feed-forward network)—making it extremely sensitive to interconnect bandwidth. On PCIe-connected GPUs, TP beyond 2-way often becomes counterproductive because communication time dominates.

3.2 Pipeline Parallelism (PP)

Pipeline parallelism assigns different layers of the model to different GPUs. GPU 0 holds layers 0–15, GPU 1 holds layers 16–31, and so on. Activations flow forward through the pipeline, with each GPU sending its output to the next.

PP's communication pattern is point-to-point rather than collective, so it tolerates lower-bandwidth links better than TP. The principal drawback is pipeline bubbles: during the decode phase, only one GPU at a time is active for a given request unless the system interleaves multiple requests across pipeline stages. This makes PP highly dependent on request-level batching to achieve good utilization.

3.3 Sequence Parallelism (SP)

Sequence parallelism distributes the sequence dimension across GPUs. Each chip processes a contiguous chunk of the input sequence. This is particularly useful for very long contexts, where the KV-cache alone can exceed a single chip's memory. Variants like Ring Attention circulate KV blocks through a ring topology, allowing each GPU to compute attention against the full context without holding it all locally. Communication involves passing KV blocks to the next neighbor in the ring, overlapping with computation on the current block.

3.4 Hybrid and Expert Parallelism

Production systems almost always combine strategies. A typical large-scale deployment might use TP within a node (where NVLink bandwidth is high) and PP across nodes (where only network bandwidth is available). For Mixture-of-Experts (MoE) models, expert parallelism places different experts on different GPUs, with an all-to-all communication pattern to route tokens to the appropriate expert. This is well-suited to small GPUs, since each chip only needs to hold a subset of experts.

3.5 Comparison of Parallelism Approaches

Strategy	Communication Pattern	Bandwidth Sensitivity	Latency Impact	Best Fit
Tensor (TP)	All-reduce per layer	Very high	Low (all GPUs active)	Intra-node, NVLink
Pipeline (PP)	Point-to-point	Low	Higher (bubbles)	Inter-node, PCIe
Sequence (SP)	Ring / neighbor	Moderate	Moderate	Long sequences
Expert (EP)	All-to-all routing	Moderate–high	Low per expert	MoE models

4. Communication Challenges and Solutions

4.1 The Communication Bottleneck

In tensor-parallel inference, each transformer layer triggers collective communication. For a model with L layers, the decode phase alone involves 2L all-reduce operations per generated token. At a TP degree of 8 on PCIe 4.0 (approximately 32 GB/s per direction), the cumulative time spent on these transfers can exceed the compute time by an order of magnitude for small batch sizes.

4.2 Overlapping Communication and Computation

The primary mitigation strategy is to overlap transfers with computation. Frameworks like DeepSpeed, Megatron-LM, and vLLM schedule NCCL operations on separate CUDA streams so that while one chunk of a matrix multiply is being reduced, the GPU is already computing the next chunk. Achieving high overlap requires careful kernel scheduling and often benefits from splitting operations into smaller tiles that can be pipelined.

4.3 Quantized Communication

Reducing the precision of data transmitted between chips—for example, sending activations in FP8 or INT8 rather than FP16—halves the bytes on the wire with minimal accuracy loss. Some systems apply separate quantization to the communication path versus the compute path, compressing activations before transmission and dequantizing on receipt.

4.4 Topology-Aware Placement

When chips are not fully connected (e.g., in a multi-node setup where intra-node bandwidth far exceeds inter-node bandwidth), it is critical to assign tensor-parallel groups to chips that share the fastest links and confine pipeline stages to the inter-node boundary. Topology-aware scheduling can reduce cross-node traffic by 50–90% depending on the architecture.

5. Memory Optimization Techniques

5.1 KV-Cache Management

The key-value cache is often the largest per-request memory consumer during decode. For a 70B model serving 2048-token sequences at FP16, the KV-cache per request can exceed 2–3 GB. Managing this efficiently is essential.

- **PagedAttention (vLLM):** Instead of pre-allocating contiguous memory for each sequence, vLLM borrows the concept of virtual memory paging from operating systems. KV-cache blocks are allocated on demand in small, fixed-size pages, eliminating internal fragmentation and enabling near-100% memory utilization. This has become the de facto standard for serving frameworks.
- **Prefix Caching:** When multiple requests share a common system prompt, the KV-cache for that prefix can be computed once and reused. This reduces redundant computation and memory for shared-prefix workloads like chatbots.
- **KV-Cache Compression:** Techniques like grouped-query attention (GQA) and multi-query attention (MQA) reduce the size of the KV-cache at the model architecture level. Post-hoc quantization of cached keys and values (e.g., to FP8 or INT4) can further halve or quarter memory usage with modest accuracy trade-offs.

5.2 Weight Quantization

Quantizing model weights is the most direct way to fit a model onto fewer or smaller chips. Common approaches include GPTQ, AWQ, and GGUF formats, which compress weights to 4-bit or even lower precision while preserving output quality through calibration. A 70B model in 4-bit quantization requires only about 35 GB—within reach of two 24 GB GPUs rather than six to eight in FP16.

The trade-off space involves accuracy degradation at extreme quantization levels, particularly for reasoning-intensive tasks. Recent advances in mixed-precision quantization keep sensitive layers (e.g., the first and last transformer blocks) at higher precision while aggressively quantizing middle layers.

5.3 Offloading and Heterogeneous Memory

When GPU memory is insufficient even after quantization, weights or KV-cache blocks can be offloaded to host CPU RAM or even NVMe storage. FlexGen demonstrated that carefully scheduling prefetching, computation, and offloading across a GPU–CPU–SSD hierarchy can achieve reasonable throughput for offline inference, though latency per token increases. For

interactive serving, partial offloading—keeping hot layers on GPU while cold layers reside in CPU RAM—can be effective, especially if the PCIe bandwidth is high enough to overlap fetches with computation.

6. Scheduling and Batching

6.1 Continuous Batching

Traditional static batching pads all sequences to the longest in the batch and waits for all to finish before accepting new requests. Continuous (or iteration-level) batching, pioneered by the Orca system and adopted by vLLM, TensorRT-LLM, and others, allows new requests to join the batch at every decode step. This dramatically improves GPU utilization and throughput, as short sequences no longer block long ones.

6.2 Disaggregated Prefill and Decode

Since prefill is compute-bound and decode is memory-bandwidth-bound, some systems assign different chip groups to each phase. A cluster of GPUs handles prefill in parallel, then hands off the KV-cache to a separate set of GPUs optimized for decode. This avoids the problem of prefill computation starving decode of resources (or vice versa) and allows each pool to be tuned independently. Frameworks like Splitwise and DistServe implement this pattern.

6.3 Speculative Decoding

Speculative decoding uses a small draft model (which can run on a single small GPU) to generate candidate token sequences, which the large target model then verifies in a single batched forward pass. If the draft model's predictions are largely correct—which they often are for routine text—multiple tokens can be accepted per step, reducing the number of expensive large-model forward passes. In a distributed system, this means fewer rounds of inter-chip communication for the large model.

7. Challenges Specific to Small-Chip Systems

7.1 Higher Parallelism Degree

Fitting a large model onto many small chips requires a higher parallelism degree than using fewer large chips. A 70B FP16 model on 12 GB GPUs might require 12–16 way parallelism versus 2–4 way on 80 GB GPUs. Each additional degree of parallelism increases the number of communication operations and the collective synchronization overhead, sometimes superlinearly.

7.2 Load Imbalance

Not all layers of a transformer are equal in size. Embedding layers, output projection layers, and the first/last transformer blocks may have different memory footprints and compute

requirements. With many small chips, the granularity of partitioning increases, and slight imbalances in layer assignment can leave some chips idle while others are overloaded. Automated partitioning tools (like Alpa or FlexFlow) search the parallelism configuration space to minimize such imbalances.

7.3 Fault Tolerance and Stragglers

A system with many chips has a higher probability that any single chip fails or slows down during inference. Since distributed inference typically requires synchronization at every layer, one slow chip can bottleneck the entire system. Solutions include redundant compute (running the same shard on two chips and taking the faster result), asynchronous pipeline designs that tolerate bounded delays, and checkpoint-restart mechanisms that can route around a failed chip.

7.4 Interconnect Heterogeneity

In practice, small-chip systems often have heterogeneous interconnects. Within a single machine, GPUs might be connected via NVLink or PCIe at different bandwidths depending on topology. Across machines, Ethernet or InfiniBand adds another tier. The parallelism strategy must be co-designed with the physical topology to avoid placing high-communication workloads on slow links.

8. Software Ecosystem and Frameworks

Framework	Key Capabilities	Notes
vLLM	PagedAttention, continuous batching, TP/PP, speculative decoding	De facto open-source standard for LLM serving; strong multi-GPU support
TensorRT-LLM	Fused kernels, FP8/INT4, TP/PP, inflight batching	NVIDIA-optimized; best latency on NVIDIA hardware
DeepSpeed-Inference	TP, heterogeneous offload, ZeRO-Inference	Good for resource-constrained environments; supports CPU offloading
llama.cpp / GGUF	CPU + GPU split, 2–6 bit quantization, RPC-based distribution	Lightweight; popular for consumer hardware; supports networked GPU splitting
Megatron-LM	TP/PP/SP, large-scale training and inference	Reference implementation for parallelism strategies
exo / Petals	Peer-to-peer distributed inference across heterogeneous devices	Designed for volunteer/consumer GPU clusters; tolerates heterogeneity

9. Performance Engineering Considerations

9.1 Profiling and Bottleneck Analysis

Effective distributed inference requires profiling both compute and communication. NVIDIA Nsight Systems can visualize kernel execution alongside NCCL operations on a timeline, revealing whether the system is compute-bound, memory-bandwidth-bound, or communication-bound for each phase. The goal is to ensure that communication is fully overlapped with useful computation.

9.2 Kernel Fusion and Flash Attention

Kernel fusion reduces the number of GPU kernel launches and intermediate memory reads/writes. Flash Attention, in particular, is critical for distributed inference: by computing attention in a tiled, online fashion without materializing the full attention matrix, it reduces per-chip memory consumption from $O(n^2)$ to $O(n)$ in sequence length. This directly increases the batch size or sequence length a small chip can support, improving overall throughput.

9.3 Choosing the Right Configuration

The optimal configuration depends on the specific hardware topology, model architecture, and workload characteristics (average sequence length, batch size, latency requirements). There is no one-size-fits-all answer. As a starting heuristic for small-chip clusters: use tensor parallelism within the highest-bandwidth domain (intra-node), pipeline parallelism across the lower-bandwidth boundary (inter-node), and tune the batch size to fill pipeline bubbles. Automated search tools can explore this space more systematically.

10. Conclusion and Recommendations

Distributing LLM inference across many small GPU chips is not only feasible but is becoming an increasingly practical alternative to monolithic large-GPU deployments. The core engineering challenge is managing the tension between per-chip memory limits and inter-chip communication overhead. The key recommendations for practitioners building such systems are as follows.

- **Start with quantization.** Reducing weight precision to 4–8 bits dramatically reduces the number of chips required and thus the communication overhead.
- **Match parallelism to topology.** Use tensor parallelism only across high-bandwidth links; fall back to pipeline parallelism where bandwidth is limited.
- **Adopt PagedAttention.** Efficient KV-cache management is non-negotiable for serving workloads; vLLM or equivalent is the baseline.
- **Use continuous batching.** Static batching leaves enormous throughput on the table in multi-user serving scenarios.
- **Profile relentlessly.** The optimal configuration is hardware- and workload-specific. Measure time-to-first-token and inter-token latency separately, and optimize each phase independently.

- **Consider disaggregation.** For high-throughput serving, separating prefill and decode onto specialized chip pools can yield significant efficiency gains.
- **Plan for failure.** More chips means more failure surface. Build in monitoring, health checks, and graceful degradation from the start.

The field is evolving rapidly, with new frameworks, quantization methods, and architectural innovations (such as linear attention and state-space models that eliminate the KV-cache entirely) continuing to shift the trade-off landscape. A distributed small-chip inference system designed with modularity and configurability in mind will be best positioned to adopt these advances as they mature.