The City College of New York

CSC 36000:
Modern Distributed Computing *with AI Agents*

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at City University of New York

October 27, 2025 CSC 36000

Today's Lecture

Reliable and Resilient Distributed AI

- Consistency-Availability Trade-off (already covered in class)
- Replication for High-Availability AI

Optimizing Performance and Efficiency

- The Energy Footprint
- Quantization
- Pruning

Live Coding Demonstration

Reliable and Resilient Distributed AI

Consistency
(Already Covered in Class)

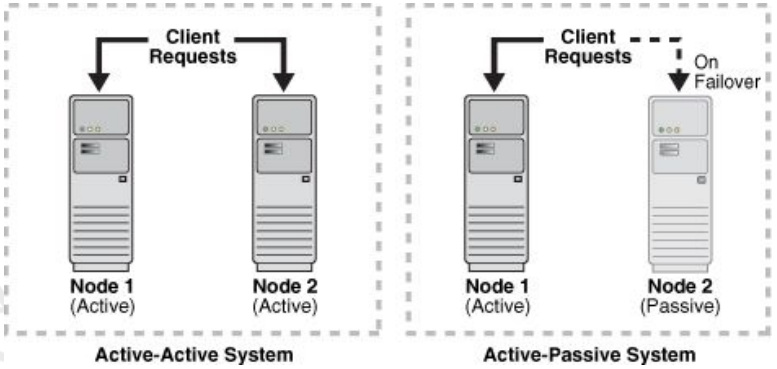
Replication for High-Availability AI

Achieving High Availability

- Another important consideration for AI-powered applications like TikTok is the ability to quickly serve *any* piece of content to *any* number of people located *all* over the world
- Multiple nodes also need to be able to serve the same data in case one goes down
- Both of these have a common solution: maintain *copies* of data across different nodes and locations!

Architecting for Uptime

Two primary architectural patterns for replication:



- **Active-Passive (Primary-Backup):** A single primary (active) node handles all traffic and write operations, and 1+ secondary (passive) nodes are kept in sync but remain on standby. One gets promoted if the primary fails.
- **Active-Active:** Multiple “hot” nodes simultaneously serve live traffic. A load balancer handles incoming requests and distributes them to the nodes. If one goes down, the load balancer simply stops sending it requests

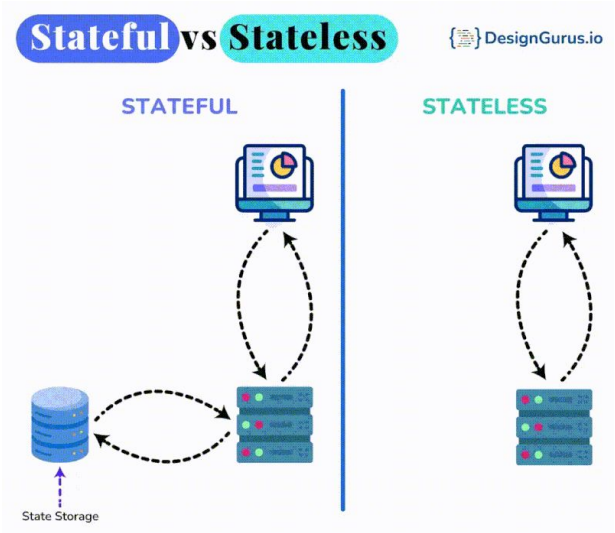
Stateful vs. Stateless

The architecture we choose often depends on whether we need to maintain or modify data over its lifetime (*stateful)* or whether we can treat each request independently (stateless)

Example (Stateful): Training an LLM involves periodically saving (checkpointing) the weights, optimizer, and training progress, making it highly stateful

Example (Stateless): Using an LLM for inference uses a static pre-trained model to generate a response to a user prompt. We don't change the model, so this is stateless

Which architecture would be good for each of these examples?



Data Synchronization

To keep replicas in sync, we need to consider a performance-consistency trade-off:

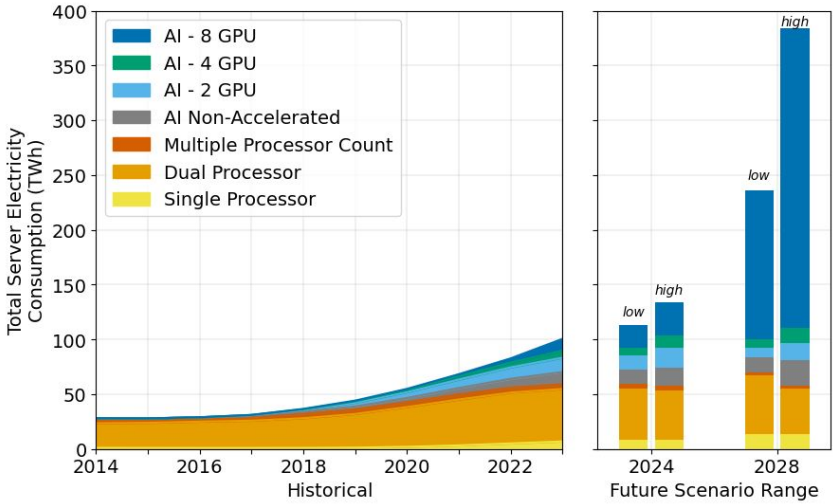
- **Synchronous Replication**: Write operations to the primary replica are not considered complete until it propagates to the secondary replicas
- **Asynchronous Replication:** The primary replica acknowledges the write even before it propagates, This makes it possible for data to be lost if a write is considered successful but the primary fails before the changes reach the secondary.

Optimizing Performance and Efficiency

The Energy Footprint

AI presents significant environmental and economic challenges for energy consumption.

When we design distributed AI systems, efficiency should be a top priority.

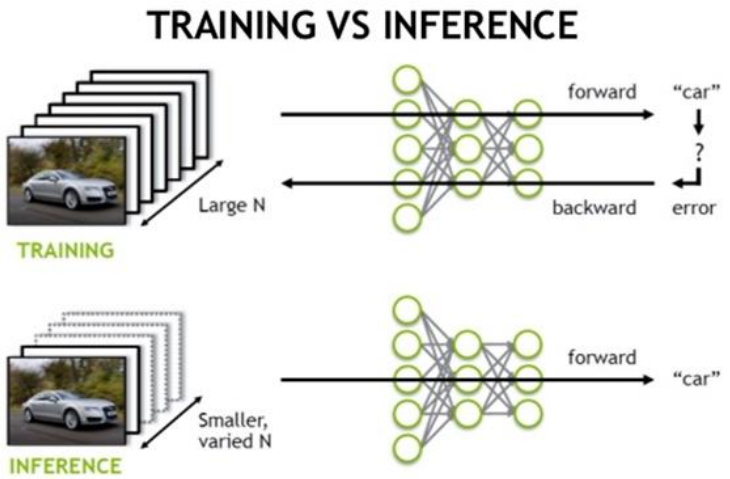


Source: https://iee.psu.edu/news/blog/why-ai-uses-so-much-energy-and-what-we-can-do-about-it

Training vs Inference

Two phases define how AI models use energy:

- **Training:** Models with billions or trillions of parameters take enormous amounts of energy and compute to train.
- **Inference:** The amount of energy for a single query is tiny. Not so much when this is scaled to billions of queries per day.



Strategies for Efficient AI

Often, making AI efficient goes hand-in-hand with adapting it to run on devices with few computational resources (e.g. smart glasses, phones, smart watches)

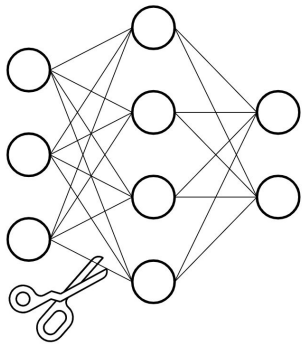
There are two main ways we can reduce the computational resources used by our AI:

- We can reduce the number of parameters (*pruning*)
- We can reduce the size of each parameter (*quantization*)

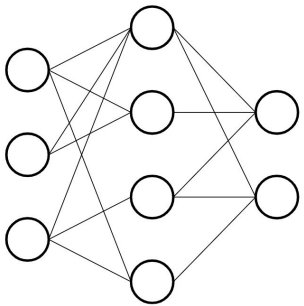
Pruning

When we use *pruning*, we remove parameters that are unimportant or redundant to the overall output of the model

- **Structured Pruning:** Remove whole blocks (e.g. channel in a ConvNet)
  - Creates a dense, but smaller, model
- **Unstructured Pruning:** Remove individual weights
  - Creates a sparse model that requires specialized hardware/software to realize inference speedup



Before pruning



After pruning

Pruning techniques often assume large networks rely heavily on particular small subnetworks for the bulk of their performance

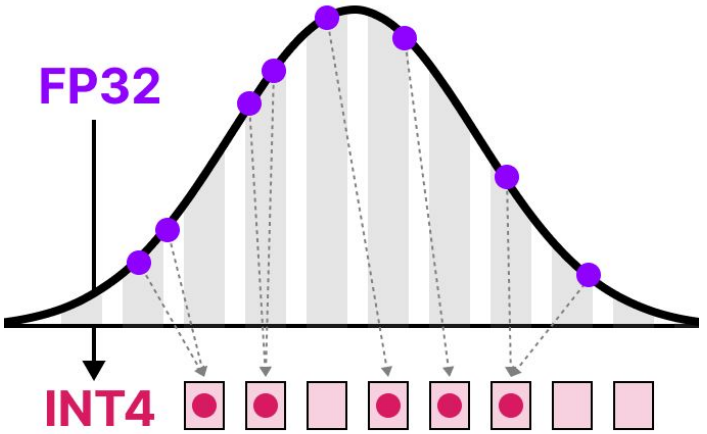
Quantization

By default, most machine learning model weights are represent with 32-bit floating points.

This allows for precise, rich calculations but is memory and energy intensive.

Quantization techniques convert these larger numbers into smaller numbers (e.g. four-bit integers) while attempting to preserve the original performance as much as possible (some performance usually lost)

An FP32 model quantized to INT4 can reduce the memory footprint by a factor of eight!



Questions?