ARTIFICIAL INTELLIGENCE AND NEURAL NETWORKS

Lecture 10b – Efficient Large-Model Systems and Deployment

Chizhi Chris ZHANG

zhangchizhi@ciomp.ac.cn
Advanced Computing and Digital Technology Research Center
University of Chinese Academy of Sciences

Spring 2026

Today's Question

What we are trying to answer

How do we keep large AI models useful while reducing cost, latency, and operational risk?

Why this final NN lecture matters

Strong models are impressive, but impressive models are not enough. If a system is too slow, too expensive, too fragile, or too hard to monitor, it will not survive real deployment.

What changes from NN9

NN9 focused on multimodal reasoning and agents. NN10 focuses on the systems question behind them: how to make large-model capability practical in the real world.

From NN9 to NN10

Last time

We studied multimodal models and agent stacks that can see, reason, retrieve, and act.

Today

We ask what it takes to actually serve those systems at scale without letting cost and latency explode.

One sentence

NN9 asked how models become more capable. NN10 asks how those capabilities become sustainable.

Where the Cost Comes From

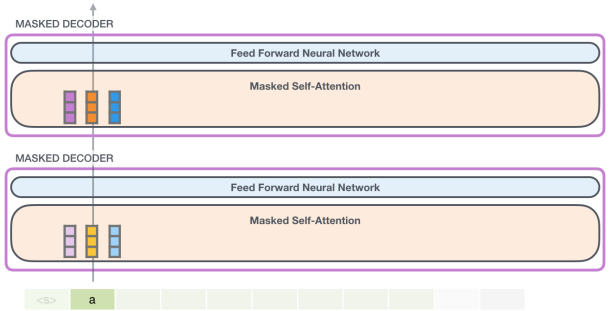
Main cost drivers

- more layers
- larger hidden states
- longer context
- repeated token generation

Why teams feel this fast

Every design choice that improves quality can also raise memory use, latency, and operating cost.

Attention Is Powerful and Expensive



Core operation

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

Why this matters

The same mechanism that makes transformers strong is also one of the main reasons long-context processing gets expensive.

Why Long Context Gets Expensive Fast

Rule of thumb

Self-attention memory and compute grow roughly with the square of sequence length.

What that means in practice

A model that feels comfortable on short prompts can become dramatically more expensive on long documents or long conversations.

This is why long-context marketing claims always need a systems question right behind them.

The Pipeline Costs More Than Pretraining

Before launch

Pretraining, tuning, safety work, evaluation, and deployment engineering all cost time and compute.

After launch

Serving, monitoring, caching, red-teaming, incident handling, and hardware refresh keep adding cost long after the first release.

The broader lesson

Pretraining is expensive, but it is not the whole bill. The system keeps charging you after the model looks finished.

A Slow System Feels Less Intelligent

A user-side reality

If an answer is excellent but arrives too slowly, users often experience the system as unreliable or frustrating rather than impressive.

Why this matters technically

Deployment quality is judged through waiting time, retry behavior, interruptions, and visible stability, not only through benchmark scores.

The practical lesson

Efficiency is part of perceived intelligence. A model that cannot respond within the task's time budget may fail even when its raw reasoning quality is strong.

Why Efficiency Is a Core Requirement

Without efficiency

Only a small number of organizations can afford to deploy strong systems broadly.

With efficiency

More products, institutions, and research groups can use capable AI under realistic budgets and latency targets.

Efficiency is not cosmetic optimization. It changes who can actually use the technology.

The Compression Toolbox

Three common levers

- quantization reduces numerical precision
- distillation trains a smaller student from a stronger teacher
- pruning removes parameters or structures with limited value

What all three are trying to do

Keep enough useful behavior for the real task while lowering memory, compute, or serving cost.

Quantization Intuition

Basic idea

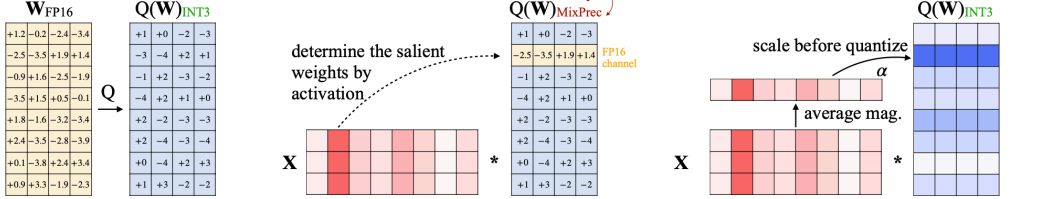
$$w \rightarrow \hat{w} = s \cdot q, \qquad q \in \mathbb{Z}$$

Read in words

Store approximate lower-precision weights instead of full-precision ones so inference moves less data and can run faster on suitable hardware.

The trick is to save cost without breaking the model's useful behavior.

Modern Quantization Is Selective



(a) RTN quantization (**PPL 43.2**)

(b) Keep 1% salient weights in FP16 (**PPL 13.0**)

(c) Scale the weights before quantization (**PPL 13.0**)

What changed from crude compression

Modern methods try to protect the most important weights or channels instead of compressing everything blindly.

Why this matters in practice

Teams want a model that still feels sharp after compression, not a cheap model that becomes obviously weaker on long reasoning or factual tasks.

What Can Go Wrong

Typical degradation

- factual quality drops
- long-form generation becomes less stable
- harder reasoning tasks suffer first

Typical mitigation

Layer-wise calibration, selective higher precision, and validating on the real tasks that matter for deployment.

Distillation Intuition

Teacher and student

A larger teacher guides a smaller student using outputs, preferences, or internal behavior traces.

Why teams use this

When a full-size teacher is too slow or expensive, distillation can preserve much of the useful behavior in a smaller model.

Distillation is one of the clearest examples of trading a little raw capability for a big systems win.

Why Teams Avoid Full Fine-Tuning Everywhere

The practical issue

If every domain, customer, or product variant requires a fully separate model, storage, training cost, and deployment complexity grow very quickly.

A common response

Teams often use lighter adaptation methods or small task-specific models so they do not have to rebuild the entire large model for every variation.

This is another reminder that deployment is about picking the smallest system that still solves the real problem well.

Mixture-of-Experts Starts from a Simple Idea

Dense model

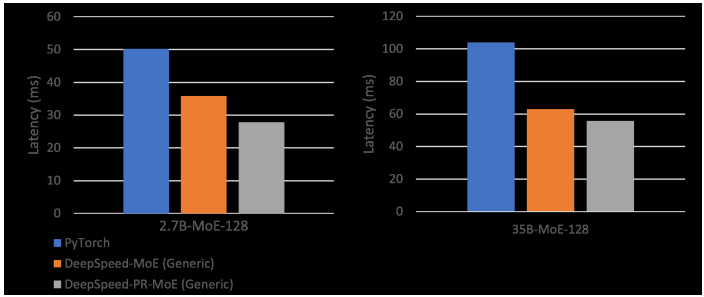
Every token activates the same large block of parameters.

MoE model

A router chooses only a subset of experts for each token, so not all parameters are active every time.

The goal is larger capacity without paying the full dense-compute price for every token.

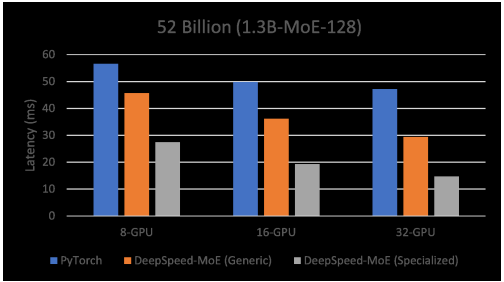
A MoE Architecture Example



How to read the diagram

Routing decides which expert subnetworks should handle each token. Capacity goes up, but routing and communication become part of the problem.

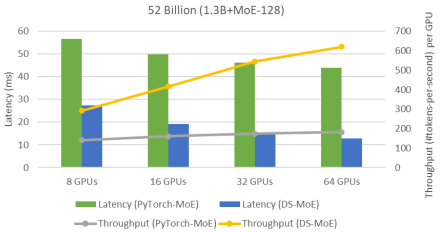
Sparse Capacity Looks Attractive



Why teams care

Sparse activation can raise total parameter capacity without making every token pay the same dense cost.

Latency and Throughput Still Decide



Deployment reality

Router overhead, communication cost, and expert imbalance can eat into the theoretical efficiency gain.

What the user actually feels

The system may be elegant on paper, but an eight-second wait for a short reply still feels broken to the person on the other side.

MoE Failure Modes

Common issues

- expert collapse
- hotspot routing
- unstable training dynamics

Typical fixes

Load-balancing losses, router regularization, and scheduling that respects hardware constraints.

A Large-Model Serving Pipeline

Three stages users never see

- prefill: encode the prompt and build the key-value cache
- decode: generate tokens iteratively
- post-process: apply formatting, filtering, and safety steps

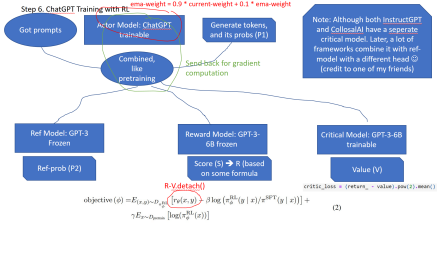
What users actually feel

Users experience only the total wait. They do not care which internal stage caused the delay.

A useful comparison

A long prompt may spend most of its time in prefill. A long answer may spend most of its time in decode. Both are “slow” from the user’s point of view.

RLHF Needs Infrastructure Too



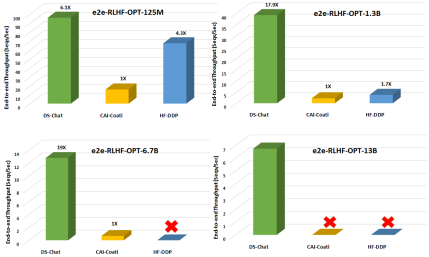
What this picture reminds us

Alignment quality depends on scalable infrastructure, not only on elegant objective functions.

What teams discover the hard way

Preference data has to be collected, cleaned, versioned, and linked to training runs. Without that plumbing, even a good idea about alignment cannot scale.

End-to-End System View



What this picture is really saying

Data, training, serving, monitoring, and safety processes must be designed together. A weak link anywhere in the chain can damage the whole product.

Latency Budgeting

Latency sources

- model compute
- memory movement
- network overhead
- tool-call waiting time

Design objective

Meet user-visible latency targets while preserving answer quality and staying inside the cost budget.

Caching Helps, but It Is Not Free

Common options

- KV cache reuse
- prompt prefix cache
- retrieval result cache

Tradeoff

Aggressive caching improves speed, but can increase staleness, inconsistency, or awkward interactions with personalized state.

Pick the Smallest System That Works

A common deployment mistake

Use the biggest model for every request, even when many requests are short, repetitive, or low-stakes.

A better strategy

Route easy work to a cheaper path and reserve the largest model for the cases that truly need deeper reasoning, longer context, or stronger generation quality.

This is not just a budget issue. It also improves responsiveness and makes the whole service easier to scale.

Efficiency Work Can Change Behavior

Technical fact

Compression or routing changes can alter model behavior, so alignment and safety must be revalidated after optimization.

Process implication

Efficiency work and safety work are coupled. One cannot simply happen after the other and then be forgotten.

This is one of the most important practical lessons in modern deployment.

What Reliability Testing Must Cover

Three questions every test plan should answer

- capability: does the system solve the normal task?
- robustness: what happens under noisy, vague, or adversarial input?
- safety: does it violate policy, hallucinate badly, or trigger harmful actions?

Why all three matter

A system can look strong in a polished demo and still fail badly once the input becomes messy or the stakes become real.

When Humans Must Stay in the Loop

Typical triggers

- high uncertainty
- medical or legal context
- access to sensitive data
- irreversible actions

Goal

Automate where safe, escalate where accountability must remain human.

A concrete rule

Summarizing a draft contract may be automated. Signing the contract or giving legal advice still belongs to a qualified person.

Observability Is Part of Trust

What to log

- prompt and response metadata
- tool-calling traces
- latency and cost signals
- safety intervention events

Why this matters

Without observability, teams cannot debug failures, trace incidents, or govern the system responsibly.

System Design Rules

Rule 1

Optimize for end-to-end quality per dollar, not isolated model score.

Rule 2

Treat latency, observability, and fallback behavior as first-class design targets.

Rule 3

Re-check alignment and reliability after every major optimization pass.

Summary

- Large-model deployment is limited as much by system cost and latency as by raw model quality.
- Quantization, distillation, and sparse routing are all ways of trading some simplicity for better efficiency.
- Serving is a pipeline problem, not only a model problem.
- Optimization can change behavior, so reliability and safety must be revalidated after efficiency work.
- Modern AI engineering ends where model design, infrastructure, observability, and governance meet.

After the Course

What should stay with you

When you see a strong model, also ask what it costs, how it is served, how it is monitored, and what happens when it fails. Real deployment is where technical maturity shows up.

A useful habit

Choose the smallest reliable system that solves the real task, and keep checking quality, safety, and accountability after every optimization step.

Thank You