The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

Distributed Intelligence for LLMs: Motivation, Methods, and Open Problems

Haoran Zhang

April 13, 2026

Electrical and Computer Engineering, The University of Texas at Austin

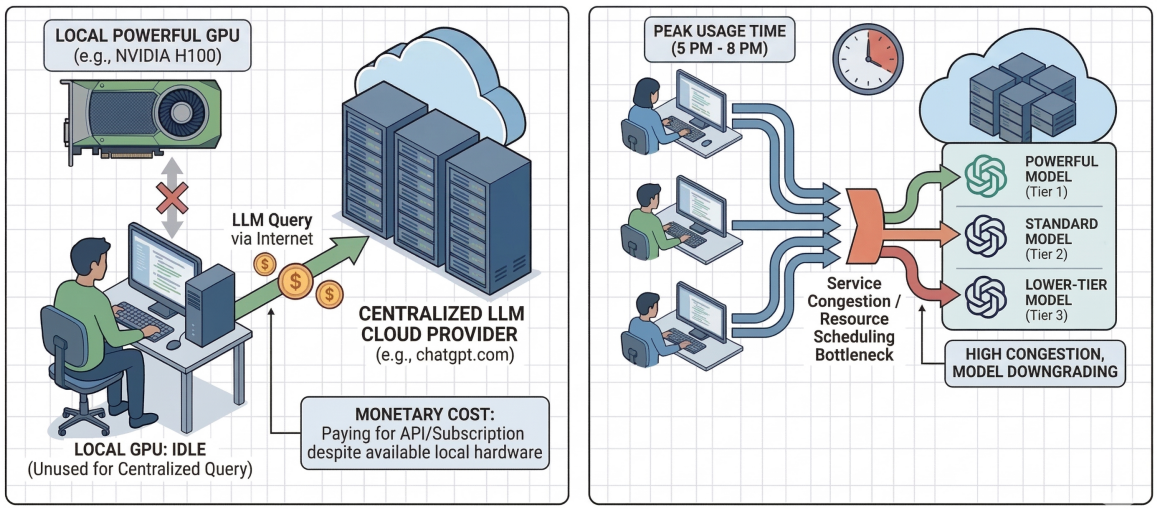
Outline

- Motivation: why distributed intelligence for LLMs?
- Two core problems:
  - Distributed inference via hierarchical routing
  - Distributed training via federated fine-tuning of LLMs
- Open problems and research lessons

The Reality of LLM Usage

- LLMs are now used for coding, writing, research, etc.
- Centralized deployment (like chatgpt.com) is constrained:
  - **Monetary cost and latency:**

Figure 1: **Constraints of Centralized LLM Deployment**



Subfigure (a): Monetary Cost Constraint

Subfigure (b): Latency and Model Allocation Constraint

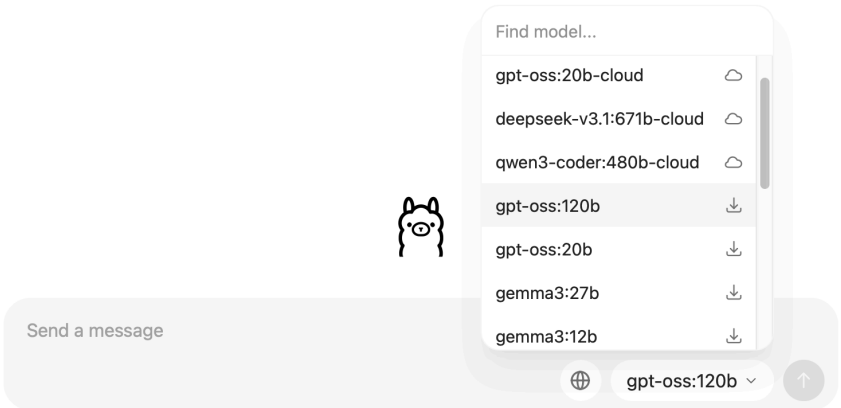
(figure generated by Gemini)

  - **Limited privacy:** Most AI companies employ users' chat data by default to train their models; some keep the data indefinitely [1]
- How can we make LLMs affordable and widely deployable?

[1] https://hai.stanford.edu/news/be-careful-what-you-tell-your-ai-chatbot

Real Example: Local LLM Deployment

- Tools such as **Ollama** make local LLM deployment much easier.
- Advantages of local deployment:
  - lower marginal cost,
  - offline usability, stronger privacy.



  DeepSeek-R1-Distill-Qwen-1.5B: 3.9GB VRAM (e.g., RTX 3060 12GB)
- Limitations of pure local deployment:
  - weaker models,
  - limited memory and compute to support complex tasks.
- Neither “all cloud” nor “all local” is ideal.

The Fundamental Tradeoff

Cloud Models

- Strong capability
- Expensive
- Privacy concerns
- Latency (query limit)

Local / Edge Models

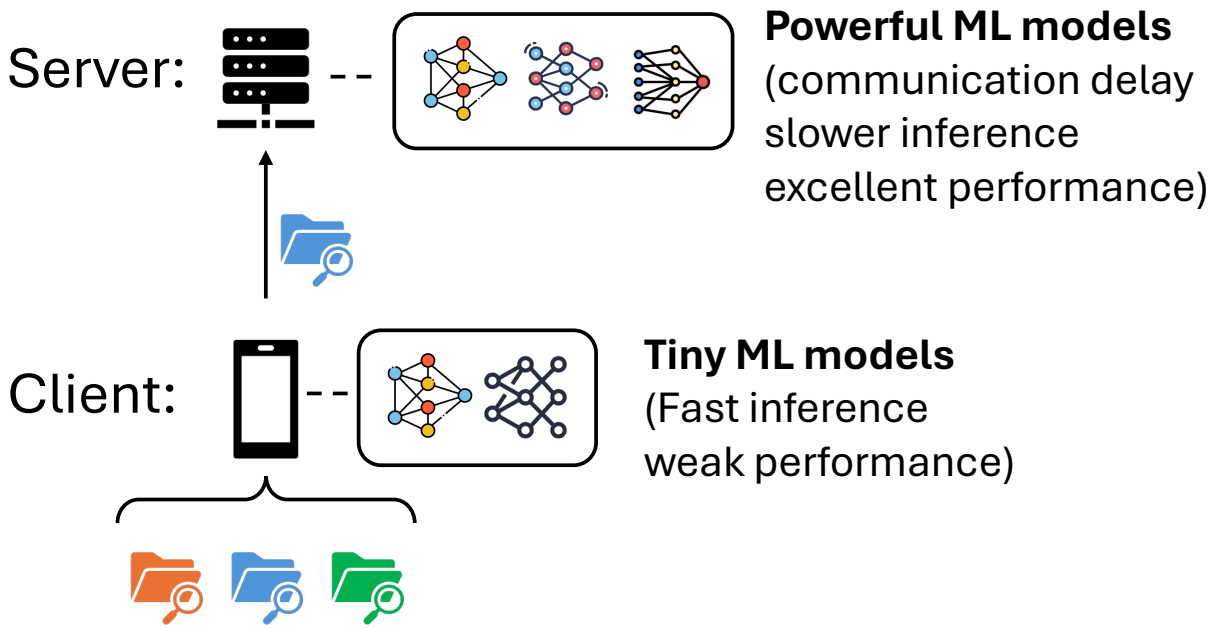
- Cheap at runtime
- Better privacy
- Lower latency
- Weaker model

The goal of distributed (networked) LLMs system:

- Find a middle point between cloud and local models
- Simple tasks executed locally
- Complex tasks executed remotely

Key Idea: Hierarchical Inference

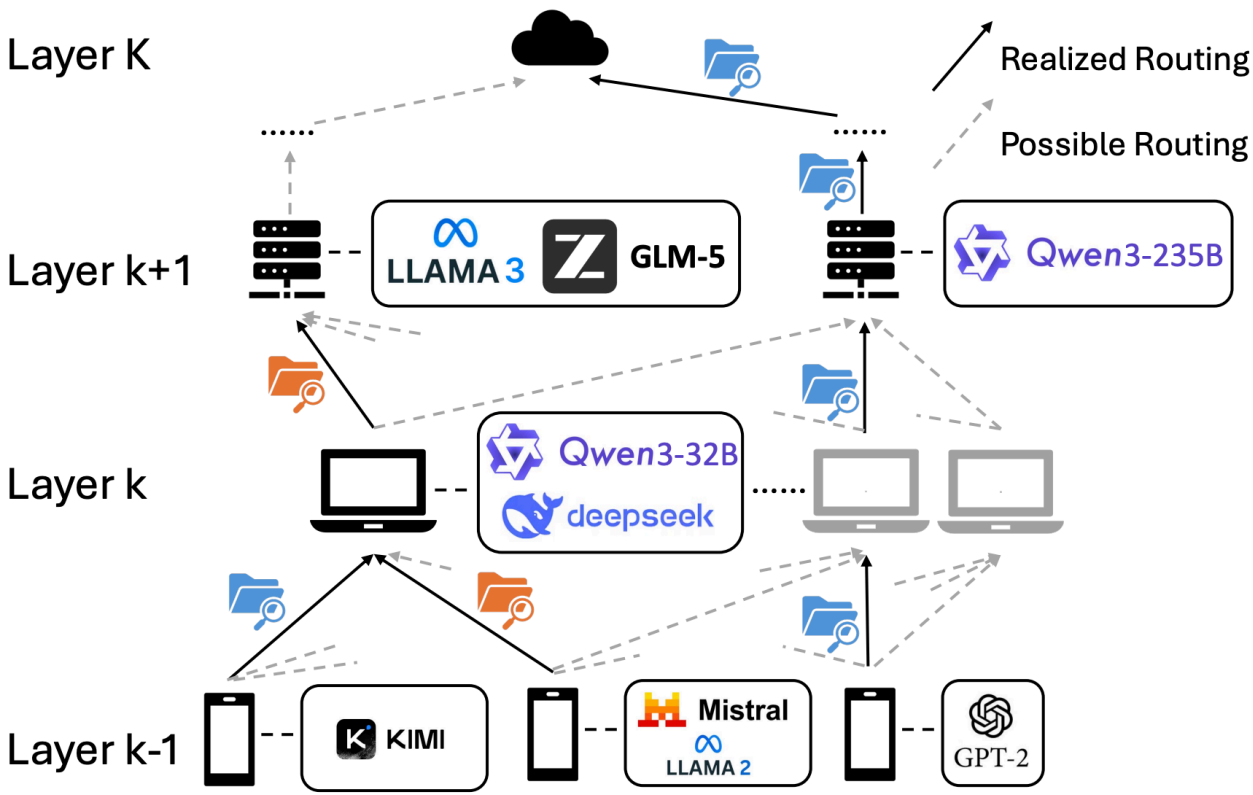
- Instead of relying on one model at one location, we can distribute inference across multiple nodes.
- Two-layer inference offloading has been studied[2], can we generalize it?



[2] Beytur, Hasan Burhan, et al. "Optimization of offloading policies for accuracy-delay tradeoffs in hierarchical inference." IEEE INFOCOM 2024.

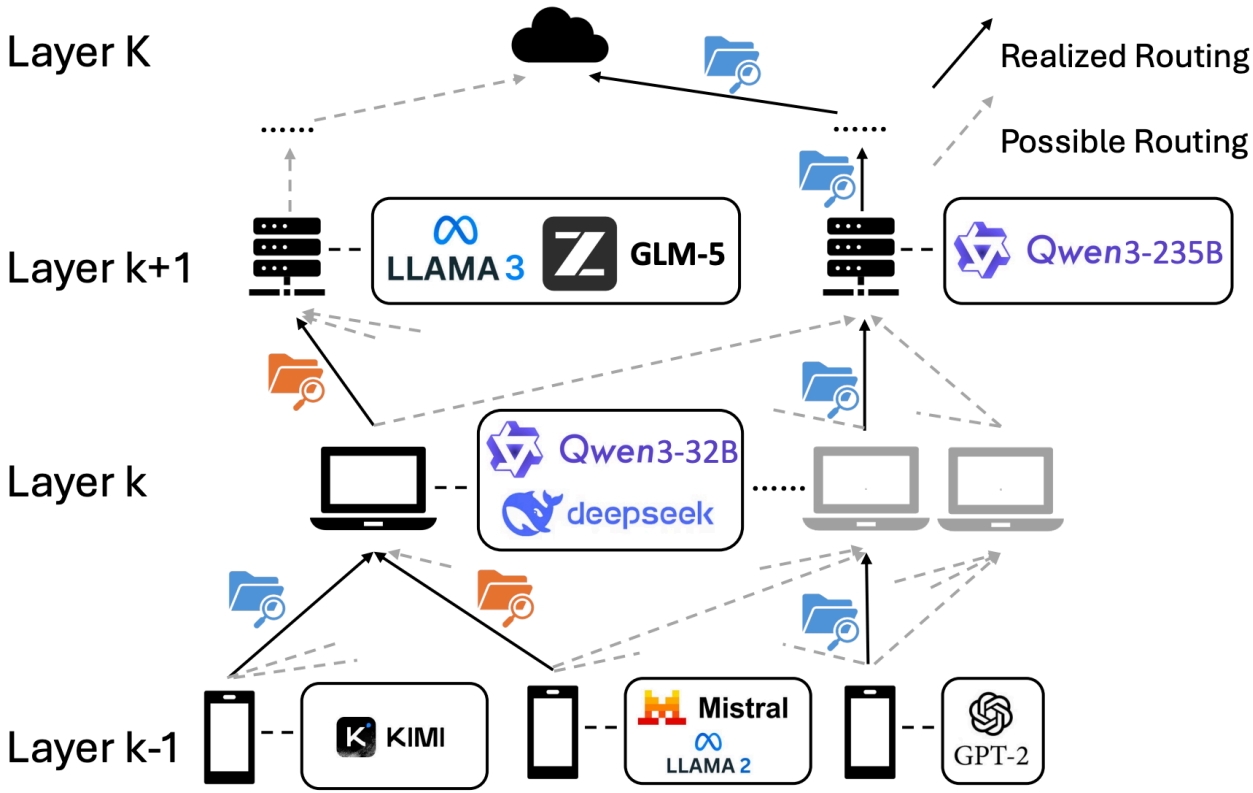
Key Idea: Hierarchical Inference

- Instead of relying on one model at one location, we can distribute inference across multiple nodes.
- These nodes may include: user devices, edge servers, enterprise servers, and cloud datacenters...



Key Idea: Hierarchical Inference

- **Objective:** task correctness $\min_\pi \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\sum_{j\in\mathcal{J}(t)} B(j,\pi)\right]$, $B(j,\pi)\in\{0,1\}$.
- **The most important thing:** How to formulate the problem?



How to formulate a research problem?

A smart problem formulation can finish 60% of the paper (maybe).

What is a good problem?

- How can we improve the model accuracy from 95% to 96%?
  - Too vague, you don't see the solution from the problem
- How can we train LLMs on a smartphone?
- How can we reduce the computational cost of LLM fine-tuning?
  - Why fine-tuning LLM is expensive? – too many parameters
  - How to reduce the population of "effective" parameters?
- How to determine the task offloading for each node?
  - What controls the offloading - the easiest solution, a threshold
  - How to place/learn the threshold for each node?
  - How to learn under non-differentiable settings?
  - When can we get feedback?
  - Any practical constraints? - communication

Multi-Armed Bandit (MAB) Problem

- K Slot Machines {1,2,...,K} (aka "Bandits" with "Arms").
- At each time step t=1,2,...,T: Pull an arm $a_t \in \{1,2,...,K\}$ and observe random reward (each arm is independent, and has some reward distribution which doesn't change over time).
- **Goal**: Maximize total (expected) reward after T time steps.

- **Problem**: At each time step, decide which arm to pull based on past history of rewards.

We have two arms: offload the task or not.

https://courses.cs.washington.edu/courses/cse312/22su/files/slides/L24_8-17_bandits.pdf

Contextual bandit view (per node)

Consider a node n of layer k with upstream nodes set $\mathcal{U}_n = \mathcal{N}_{k+1}$.

Context: realized confidence $z_n(j)$.

Experts suggest decisions. Define a set of experts $h \in \mathcal{H}$, each expert supports a threshold $\theta_h \in [0, 1]$. Define joint experts to handle multi-destination offloading:

$$e = (h, n') \in \mathcal{E}_n \triangleq \mathcal{H} \times \mathcal{U}_n,$$

$\pi_{(h,n')}$ (or π_e) means: *offload to n' if $z_n(j) < \theta_h$; otherwise ends locally.*

Node maintains expert weights $\{w_n^{(h,n')}\}_{(h,n')\in\mathcal{E}_n}$ (sum to 1).

Action probabilities. Offload to n':

$$p_n^j(n') = \sum_{h\in\mathcal{H}} w_n^{(h,n')} \mathbb{1}_{\theta_h > z_n(j)}.$$

Local termination:

$$p_n^j(0) = \sum_{e=(h,n')\in\mathcal{E}_n} w_n^e(t^j) \mathbb{1}_{\theta_h \le z_n(j)}.$$

Contextual bandit view (per node) - EXP4

Node maintains expert weights $\{w_n^{(h,n')}\}_{(h,n')\in\mathcal{E}_n}$ (sum to 1).

Action probabilities. Offload to n':

$$p_n^j(n') = \sum_{h\in\mathcal{H}} w_n^{(h,n')} \mathbb{1}_{\theta_h > z_n(j)}.$$

Local termination:

$$p_n^j(0) = \sum_{e=(h,n')\in\mathcal{E}_n} w_n^e(t^j) \mathbb{1}_{\theta_h \le z_n(j)}.$$

Exploration: mix with uniform: $(1-\lambda)p_n^j(a) + \lambda/(|\mathcal{U}_n|+1)$.

Expert weights are updated based on the cumulative loss $\hat{s}_n^{(h,n')}(t-1)$ observed up to the end of slot $t-1$:

$$w_n^{(h,n')}(t) = \frac{\exp\big(-\eta \hat{s}_n^{(h,n')}(t-1)\big)}{\sum_{e'\in\mathcal{E}_n} \exp\big(-\eta \hat{s}_n^{e'}(t-1)\big)}.$$

$$\hat{s}_n^{(h,n')}(t-1) = \sum_{t'=1}^{t-1} \sum_{j\in\mathcal{J}_n(t')} \hat{f}_n^{n'}(j,\pi_h) \mathbb{1}_{n\in\omega^{\boldsymbol{\pi},j}},$$

Contextual bandit view (per node) - EXP4

In practice, we do not observe the expert loss if we do not take the action as the expert suggested.

Naive unbiased estimator (importance weighting):

$$\hat{F}^{n'}_{\text{naive},n}(j,\pi_h) = \mathbb{1}_{\text{fb},n}(j)\frac{f_n^{n'}(j,\pi_h)}{\rho_n^{\boldsymbol{\pi}}(j)}$$

Recursion: $\rho_n^{\boldsymbol{\pi}}(j) = 1$ for the top layer (human judge), then $\rho_n^{\boldsymbol{\pi}}(j) = \sum_{a\in\mathcal{U}_n} p_n^j(a)\,\rho_a^{\boldsymbol{\pi}}(j)$.

Issue: in deep hierarchies, $\rho_n^{\boldsymbol{\pi}}(j)$ can be tiny $\Rightarrow$ variance scales like $1/\rho_n^{\pi}(j)$ (depth-amplified instability).

Variance-reduced unbiased estimator:

$$\hat{F}^{n'}_{\text{vr},n}(j,\pi_h) = \mathbb{1}_{\text{fb},n}(j)\frac{f_n^{n'}(j,\pi_h) - \bar{f}^{n'}_{n,y(j)}(\pi_h)}{\rho_n^{\boldsymbol{\pi}}(j)} + \bar{f}^{n'}_{n,y(j)}(\pi_h).$$

How to merge constraints into the loss

Constraints:

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[C_n^{\boldsymbol{\pi}}(t)\right] \le \gamma_n \tau.$$

γ_n is the maximum allowable long-term average resource consumption rate for node n

Use virtual queues to track debt. Define a virtual queue $Q_n(t)$:

$$Q_n(t+1) = \max\left\{Q_n(t) + C_n^{\pi}(t) - \gamma_n \tau,\ 0\right\}, \qquad Q_n(1) = 0.$$

Mean-rate stability of $Q_n(t)$: $\lim_{T\to\infty} \mathbb{E}[Q_n(T)]/T = 0$

Lyapunov function: $L(t) = \frac{1}{2} \sum_{n\in\mathcal{N}\setminus\mathcal{N}_1} Q_n(t)^2$.

Per-slot objective (control parameter $v > 0$):

$$\min_{\boldsymbol{\pi}} \sum_{n\in\mathcal{N}\setminus\mathcal{N}_1} q_n(t)\mathbb{E}\left[C_n^{\boldsymbol{\pi}}(t)\right] + v\mathbb{E}\left[\sum_{j\in\mathcal{J}(t)} B(j,\boldsymbol{\pi})\right]$$

Intuition: large queues $\Rightarrow$ penalize offloading more aggressively.

Results

Table 1: Offloading performance across multi-layer hierarchical systems. Our proposed VR-Ly-EXP4 outperforms all baseline methods across all settings, achieving the lowest inference error and the highest hit rate.

Methods	3-layer (4-2-1)			4-layer (8-4-2-1)			5-layer (16-8-4-2-1)		
	Feedback Rate	Hit Rate (↑)	Error Rate (↓)	Feedback Rate	Hit Rate (↑)	Error Rate (↓)	Feedback Rate	Hit Rate (↑)	Error Rate (↓)
Random	0.0146	0.0	0.4805±0.012	0.0017	0.0	0.4793±0.008	0.0002	0.0	0.4705±0.011
Round-Robin	0.0146	0.0	0.4627±0.014	0.0019	0.0	0.4603±0.002	0.0002	0.0	0.4693±0.004
Pure Local	0.0	0.0	0.4820±0.013	0.0	0.0	0.4803±0.001	0.0	0.0	0.4680±0.010
Ly-EXP4	0.2038	0.407	0.3433±0.012	0.1720	0.399	0.3313±0.003	0.1691	0.3949	0.3222±0.005
VR-Ly-EXP4-LocalLoss	0.2226	0.432	0.3207±0.006	0.2180	0.423	0.3117±0.007	0.2212	0.4346	0.3103±0.001
VR-Ly-EXP4	0.2120	**0.442**	**0.3172**±0.012	0.2142	**0.443**	**0.3065**±0.002	0.2089	**0.4456**	**0.2923**±0.001

Zhang, H., Cha, S., Beytur, H. B., Chan, K. S., de Veciana, G., & Vikalo, H. (2026). Online Learning for Multi-Layer Hierarchical Inference under Partial and Policy-Dependent Feedback.

Transition: Inference is Only Half the Story

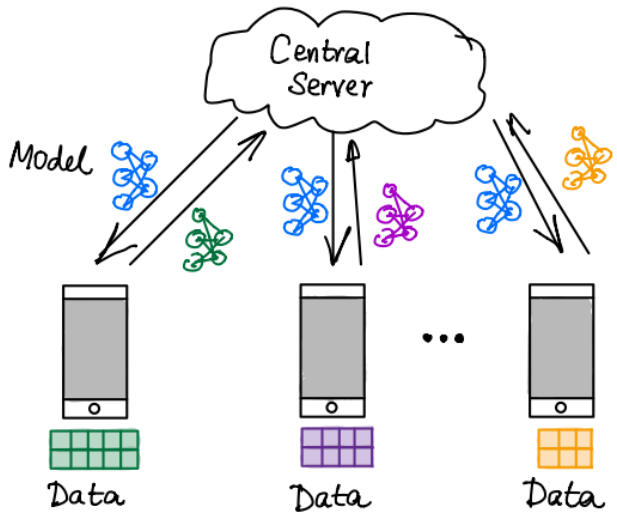
- So far, we discussed distributed inference for LLMs
- But there is another equally important question:

Can we train/fine-tune LLMs in a distributed manner?

- Your personalized model with your private data
  - E.g., let the model memorize your address, phone number...
- One user's private data can be limited, work with other users without sharing the privacy?
- This leads to the second part of the talk: **federated learning**.

Federated Learning

- Main Idea: Keep the data at the edge client, and bring the model training to the edge
- Sketch of the Algorithm:
  1. The aggregating server sends the current version of the model to available clients
  2. The clients train the model locally for a few iterations and send it back
  3. The server aggregates the models and goes back to step 1



https://www.andrew.cmu.edu/course/18-667/

Full-parameter fine-tuning is expensive...

LoRA for Parameter-efficient LLM Fine-tuning

- Suppose you have a pre-trained model weights denoted by a matrix $\mathbf{W}_0 \in \mathbb{R}^{d \times k}$, representing the key/query/value/output matrices in the transformer attention module
- The updated model after finetuning is $\mathbf{W}_0 + \Delta\mathbf{W}$, where $\Delta\mathbf{W} \in \mathbb{R}^{d \times k}$, can be found by computing gradients w.r.t. $\mathbf{W}_0$
- The Low-Rank Adaptation (LoRA) paper uses the intuition that updates to the parameter have a low intrinsic dimension and can be represents more compactly instead of the full $\mathbb{R}^{d \times k}$ matrix

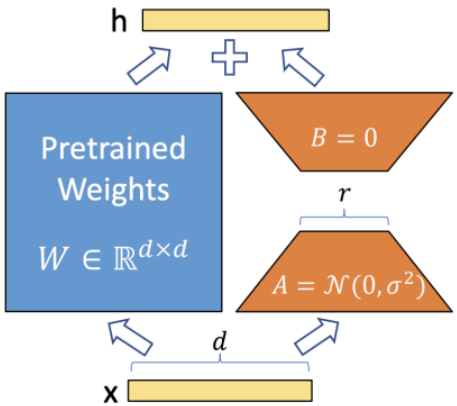


Figure 1: Our reparametrization. We only train A and B.

LoRA for Parameter-efficient LLM Fine-tuning

- LoRA represents the updates $\Delta\mathbf{W} = \mathbf{BA}$ where $\mathbf{B} \in \mathbb{R}^{d\times r}$ and $\mathbf{A} \in \mathbb{R}^{r\times k}$ where r is much smaller than d and k
- The elements of $\mathbf{A}$ are initialized randomly to Gaussian $\mathcal{N}(0, \sigma^2)$, and elements of $\mathbf{B}$ are initialized to 0.
- Thus their product $\Delta\mathbf{W}$ will get initialized to 0, meaning that the beginning of fine-tuning, the model parameters as same as the pre-trained weights $\mathbf{W}_0$

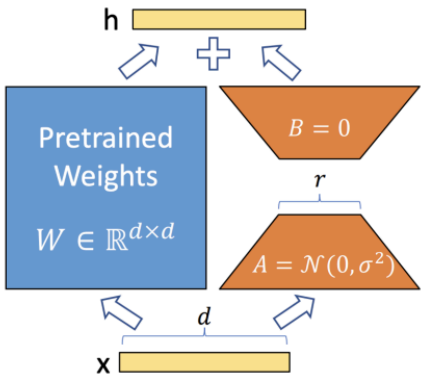


Figure 1: Our reparametrization. We only train A and B.

$d = k = 1024$, full-paramter training: around 10^6 trainable parameters

LoRA $r = 4$: $2 \times 1024 \times 4 \approx 8000$ trainable parameters

LoRA for Federated LLM fine-tuning

- What is the updated model $\mathbf{W}$ at the server?

$$\mathbf{W} = \mathbf{W}_0 + \frac{1}{m}\sum_{i=1}^{m} \mathbf{B}_i\mathbf{A}_i$$

- Now what does the server send back to the clients?
- Current methods send $\bar{\mathbf{B}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{B}_i$ and $\bar{\mathbf{A}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{A}_i$
- Clients initialize their $\mathbf{B}_i = \bar{\mathbf{B}}$ and $\mathbf{A}_i = \bar{\mathbf{A}}$ and proceed to the next training round

- This would imply that the updated global model at the client is

$$\mathbf{W} = \mathbf{W}_0 + \left(\frac{1}{m}\sum_{i=1}^{m} \mathbf{B}_i\right)\left(\frac{1}{m}\sum_{i=1}^{m} \mathbf{A}_i\right)$$

$$\neq \mathbf{W}_0 + \frac{1}{m}\sum_{i=1}^{m} \mathbf{B}_i\mathbf{A}_i$$

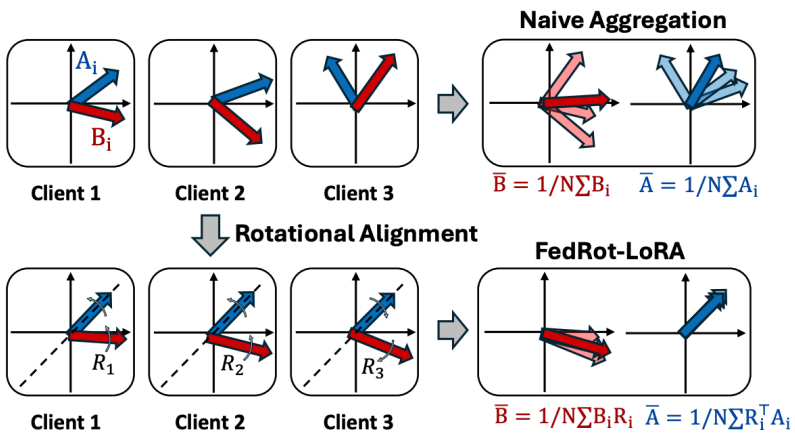
- Correcting for this mismatch is an open question

FedRot-LoRA

How can we improve the factor-wise aggregation?

$$\mathbf{W} = \mathbf{W}_0 + \left(\frac{1}{m}\sum_{i=1}^{m}\mathbf{B}_i\right)\left(\frac{1}{m}\sum_{i=1}^{m}\mathbf{A}_i\right)$$
$$\neq \mathbf{W}_0 + \frac{1}{m}\sum_{i=1}^{m}\mathbf{B}_i\mathbf{A}_i$$

Rotational invariance: $BA = (BR)(R^\top A)$



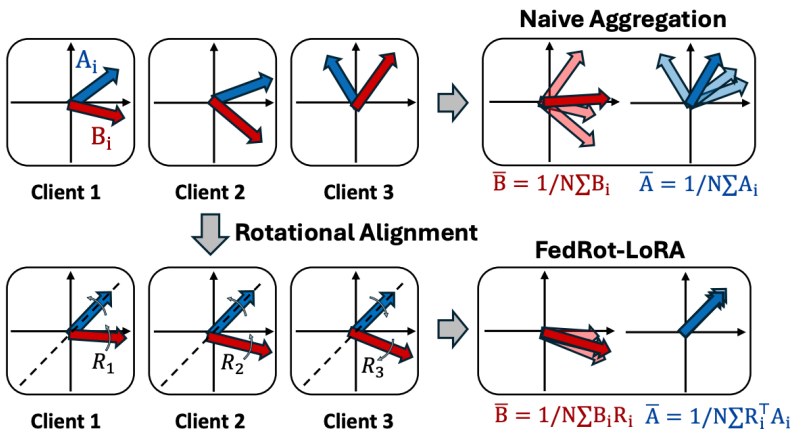
Zhang, H., Kim, D., Cha, S., & Vikalo, H. (2026). FedRot-LoRA: Mitigating Rotational Misalignment in Federated LoRA. arXiv preprint arXiv:2602.23638.

Future directions

How can we improve the factor-wise aggregation?

$$\mathbf{W} = \mathbf{W}_0 + \left(\frac{1}{m}\sum_{i=1}^{m}\mathbf{B}_i\right)\left(\frac{1}{m}\sum_{i=1}^{m}\mathbf{A}_i\right)$$
$$\neq \mathbf{W}_0 + \frac{1}{m}\sum_{i=1}^{m}\mathbf{B}_i\mathbf{A}_i$$

Rotational invariance: $BA = (BR)(R^\top A)$



- Reference model
- Beyond rotation optimization - rescaling

Research Advice: Define the Problem Well

Most important lesson
Defining the problem well is often more important than designing a complicated solution.

- A strong problem formulation helps clarify:
  - what is the real bottleneck,
  - what assumptions are reasonable,
  - what objective should be optimized.
- In many cases, once the problem is defined correctly, the solution direction becomes much more natural.

Thank You

Thank you!

Questions?

Haoran Zhang
haoranz@austin.utexas.edu