Generative and Agentic AI in Practice

DS 246 (1:2)

Lecture Topics

Phase 1: Foundations (Weeks 1-8, leading to Midterm)

Week 6 (Sep 10): Finetuning LLMs, Methods of Finetuning, Model Compression

- Topics: finetuning paradigms, parameter-efficient methods (classic + new), compression strategies, and cutting-edge efficient architectures like MoR.

Five-Level Framework (recap)

Five-Level Framework for Working with LLMs

4. Fine-Tuning (Parameter-Efficient Methods)

Adapting pre-trained models to specific tasks by training additional parameters while keeping the base model frozen.

Popular Methods

- LoRA (Low-Rank Adaptation): Adds trainable low-rank matrices (~0.1-1% of original parameters)
- QLoRA: Quantized version using 4-bit precision for memory efficiency
- DoRA: Weight-decomposed adaptation for better performance Adapters: Small trainable modules inserted between layers

Advantages

- Reduces hallucinations
- Provides up-to-date information
- Enables domain-specific responses
- Source citation capability

Example Applications

- Domain-specific chatbots, specialized translation, custom writing styles

Introduction to Finetuning

Introduction to Finetuning

Why Finetuning Matters

- Adapt LLMs to domain-specific tasks (biomedical, legal, finance).
- Personalization for organizations or user groups.
- Boosts performance without retraining from scratch.

Deployment Challenges

- High compute cost (billions of parameters).
- Large memory footprint (GPU/TPU requirements).
- Latency issues in real-time applications.
- Environmental & financial cost of large-scale training.

Introduction to Finetuning

Trade-Offs

- Full Finetuning → high accuracy, but costly.
- Parameter-Efficient Finetuning → lighter, faster, but may sacrifice some performance.
- Balance between accuracy, efficiency, and scalability.

Introduction - Challenges in Finetuning LLMs

Compute Cost

- Training requires large GPU/TPU clusters.
- High energy consumption & monetary cost.

Memory Usage

- Billions of parameters → huge VRAM needs.
- Limits scalability on smaller devices.

Introduction - Challenges in Finetuning LLMs

Latency

- Slow inference for real-time applications.
- Bottlenecks in multi-user deployment.

Catastrophic Forgetting

- Finetuning on new tasks may overwrite general knowledge.
- Loss of performance on original/pretrained capabilities.

2.

Finetuning Methods

Finetuning Methods - Full Finetuning

Updates all parameters of a pretrained model, allowing it to adapt fully to a specific downstream task. It's like rewriting the entire textbook for a new course.

Pros

- Highest Task Adaptation: Achieves the best performance because the entire model is optimized.
- Maximum Flexibility: The model can learn intricate, task-specific patterns that a frozen backbone couldn't.

Cons

- Extremely Resource-Intensive: Requires significant computational power and memory (GPU hours, VRAM), making it very expensive.
- Potential for Catastrophic Forgetting: The model may forget the general knowledge it learned during pretraining.

Finetuning Methods - Full Finetuning

Updates all parameters of a pretrained model, allowing it to adapt fully to a specific downstream task. It's like rewriting the entire textbook for a new course.

Pros

- Highest Task Adaptation: Achieves the best performance because the entire model is optimized.
- Maximum Flexibility: The model can learn intricate, task-specific patterns that a frozen backbone couldn't.

Cons

- Extremely Resource-Intensive: Requires significant computational power and memory (GPU hours, VRAM), making it very expensive.
- Potential for Catastrophic Forgetting: The model may forget the general knowledge it learned during pretraining.

Finetuning Methods - Partial Finetuning

Freeze the majority of the model's layers (the "backbone") and only trains a new "head" or the final few layers.

Pros

- Reduced Resource Needs: Significantly lowers computational costs and memory requirements, making it more accessible.
- Preserves Pretrained Knowledge: By freezing the backbone, it retains the general features and knowledge learned during pretraining.

Cons

- Limited Adaptability: Performance can be constrained since the core features of the model can't be changed to better suit the new task.
- Suboptimal Performance: May not reach the same level of accuracy as full finetuning, especially for tasks that are very different from the original pretraining data.

Parameter-Efficient Finetuning (PEFT)

- A set of techniques to adapt Large Language Models (LLMs) to new tasks.
- Tunes only a small subset of parameters, freezing the rest.
- Saves significant compute resources: time, memory, storage.
- Achieves performance comparable to full finetuning.

Why PEFT Matters

- Scalability → enables finetuning billion-parameter LLMs on modest hardware.
- Efficiency → fewer parameters to train, faster adaptation.
- Reusability → can store multiple lightweight adapters for different tasks.
- Deployment friendly → reduces inference overhead compared to full FT

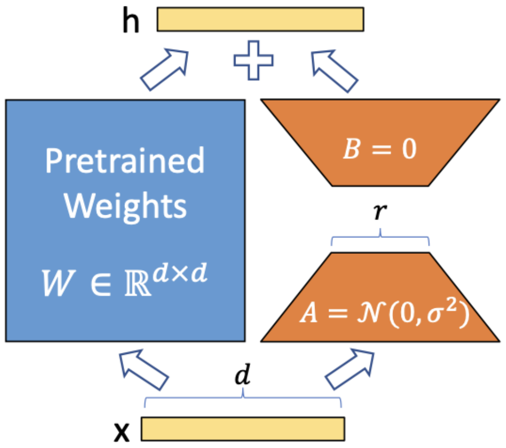
Parameter-Efficient Finetuning (PEFT)

Common PEFT Methods

- Adapters → insert small, trainable bottleneck layers.
- Prefix / Prompt Tuning → learn soft prompts or prefix tokens to guide the model.
- LoRA (Low-Rank Adaptation) → inject low-rank matrices into weight layers.
- BitFit → update only bias terms, all other parameters frozen.

Parameter-Efficient Finetuning: LoRA (Low-Rank Adaptation)

- LoRA is an innovative PEFT method that freezes the original, pre-trained weights of a model.
- Instead of directly updating the main weight matrix, let's call it W, LoRA injects a small, low-rank matrix update.
- This update is created by multiplying two much smaller matrices, A and B.



https://arxiv.org/pdf/2106.09685

Parameter-Efficient Finetuning: LoRA (Low-Rank Adaptation)

The adapted weight, W', is calculated as:

$$W' = W + \Delta W = W + AB$$

Where:

- W is the original, frozen weight matrix.
- $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$ are the trainable low-rank matrices.
- The rank r is a hyperparameter that's much smaller than the dimensions of the original matrix ($r \ll \min(d, k)$). This is the key to LoRA's efficiency.
- Only matrices A and B are trained, while the original W remains fixed. This means that a model with billions of parameters might only need a few million trainable parameters for a new task.

Pros and Cons of LoRA

Pros

- **Extreme Efficiency:** Drastically reduces the number of trainable parameters, leading to lower GPU memory usage and faster training.
- **Storage-Friendly:** Since only the small matrices A and B are saved for each task, we can store multiple fine-tuned versions of a large model with minimal disk space.
- **No Inference Latency:** After training, the matrices A and B can be merged back into the original weight matrix W to create a new, single weight matrix W′.

Pros and Cons of LoRA

Cons

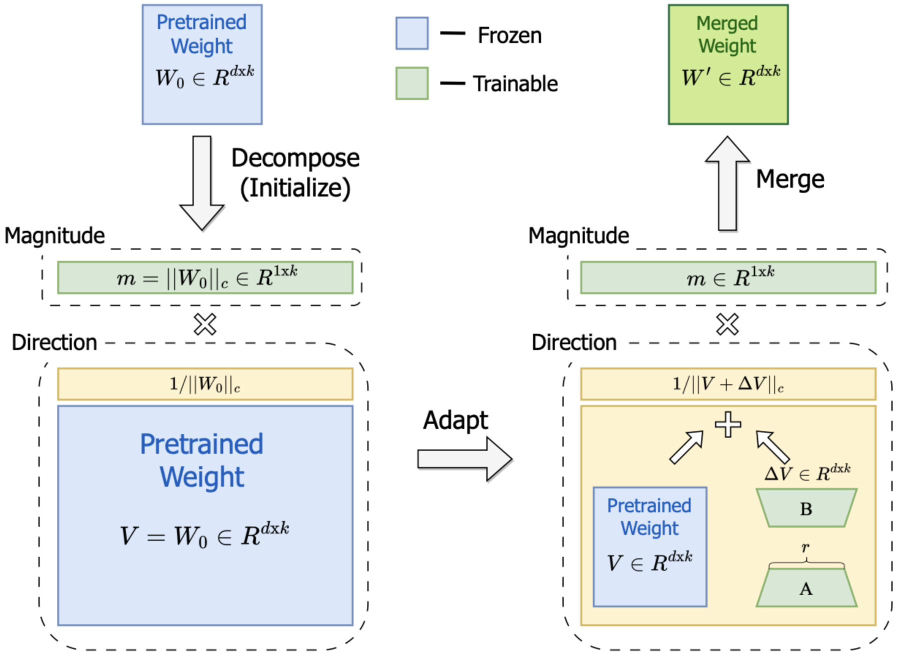
- **Performance Trade-off:** While LoRA often achieves performance comparable to full fine-tuning, it might not reach the absolute highest level of accuracy for certain complex tasks, as it constrains the model's ability to make large-scale changes to its weights.
- **Hyperparameter Tuning:** The performance of LoRA is sensitive to the choice of the rank r. Finding the optimal value requires some experimentation and tuning.

Parameter-Efficient Finetuning:
DoRA (Weight-Decomposed Low-Rank Adaptation))

- Designed to bridge the performance gap between LoRA and full fine-tuning.
- It achieves this by recognizing that fine-tuning involves two distinct kinds of changes to a model's
    - adjusting the magnitude and

$$W' = \underline{m}\frac{V + \Delta V}{||V + \Delta V||_c} = \underline{m}\frac{W_0 + \underline{BA}}{||W_0 + \underline{BA}||_c}$$

where $m = ||W_0||_c$ and $V = W_0$



https://arxiv.org/pdf/2402.09353

Parameter-Efficient Finetuning: Recent advances:

- EDoRA (SVD-initialized, extremely efficient).
- BiDoRA (bi-level optimization).
- MiLoRA (mixture of LoRA experts with prompt-aware routing).
- Expert-based methods: PERFT, ESFT, S'MoRE, MoRE.

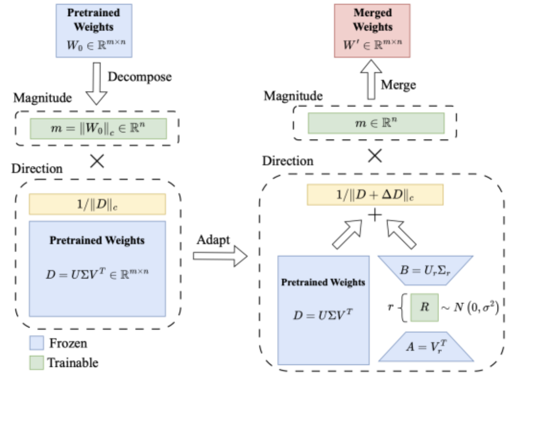


Figure 1. An overview of EDoRA

https://arxiv.org/pdf/2501.12067

3.

The Challenge of Scale

The Challenge of Scale

Why Model Compression Matters for LLMs

- **Deployment Bottleneck:** LLMs are too big for efficient deployment, requiring massive compute, memory, and energy.
- **The Goal:** Make models smaller and faster with minimal performance loss.
- Key Benefits:
  - Reduced Costs: Drastically cuts memory, latency, and operational compute costs.
  - Expanded Access: Enables on-device inference on consumer hardware (mobile, edge devices) and facilitates scalable cloud deployment.

The Challenge of Scale

Why Model Compression Matters for LLMs

- **Main Approaches**
  - Pruning
  - Quantization
  - Knowledge Distillation
  - Low-Rank Factorization

Core Compression Techniques

Pruning

- Remove redundant / less important parameters.
- **Types:**
  - Unstructured Pruning → remove individual weights (high compression, but limited hardware speedup).
  - Structured Pruning → remove neurons, filters, or attention heads (hardware-friendly, faster inference).
- **When Applied:**
  - During training → pruning-aware training.
  - After training → post-training pruning.
- Methods:
  - Magnitude-based → prune smallest weights.
  - Saliency-based → prune least impactful weights.

Core Compression Techniques

Quantization

- Reduce precision of parameters/activations (e.g., FP32 → INT8/INT4).
- **Techniques:**
  - Post-Training Quantization (PTQ) → fast, no retraining, some accuracy loss.
  - Quantization-Aware Training (QAT) → simulates quantization during training, better accuracy.
  - Mixed-Precision Quantization → assign bit-widths based on layer sensitivity.
- Popular Methods:
  - 8-bit quantization (efficient, widely used).
  - Bfloat16, NF4 (NormalFloat 4-bit) in QLoRA (for LLMs).

Core Compression Techniques

Knowledge Distillation

- Train a smaller student model to mimic a larger teacher model.
- **Forms:**
  - Logit Distillation → match teacher's output probabilities.
  - Feature Distillation → match intermediate representations.
- Advantages:
  - Student can be much smaller yet retain accuracy.
  - Teacher & student can have different architectures.

Core Compression Techniques

Low-Rank Factorization

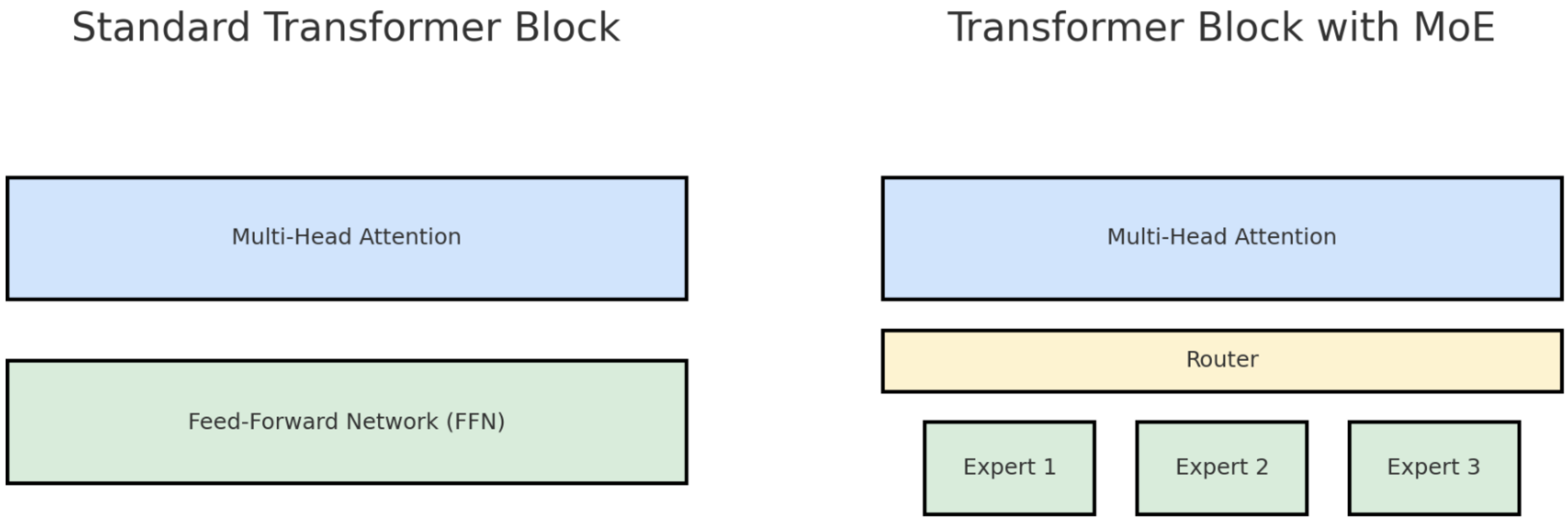
- Approximate large weight matrices with products of smaller, low-rank matrices.
- **Technique:**
  - Singular Value Decomposition (SVD) → factorize weights.
- **Benefits:**
  - Significant reduction in trainable parameters.
  - Forms the basis of LoRA (Low-Rank Adaptation) in PEFT.
- Use Cases:
  - Compression (storage + compute savings).
  - Efficient finetuning of LLMs.

https://arxiv.org/pdf/2407.06204

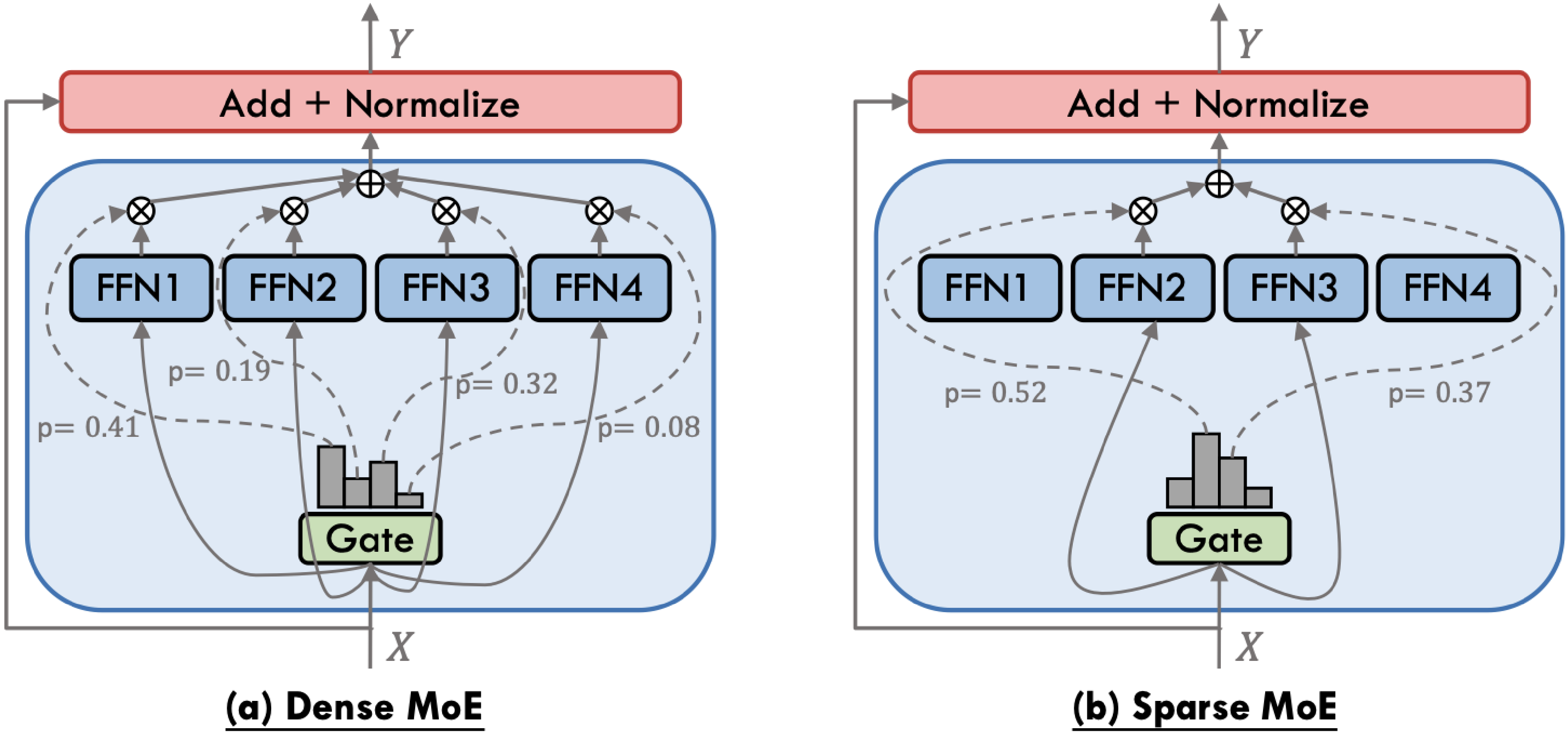
3.

Beyond Finetuning

Beyond Finetuning: Mixture of Experts



Beyond Finetuning: Mixture of Experts



https://arxiv.org/pdf/2407.06204

Beyond Finetuning: Mixture of Experts

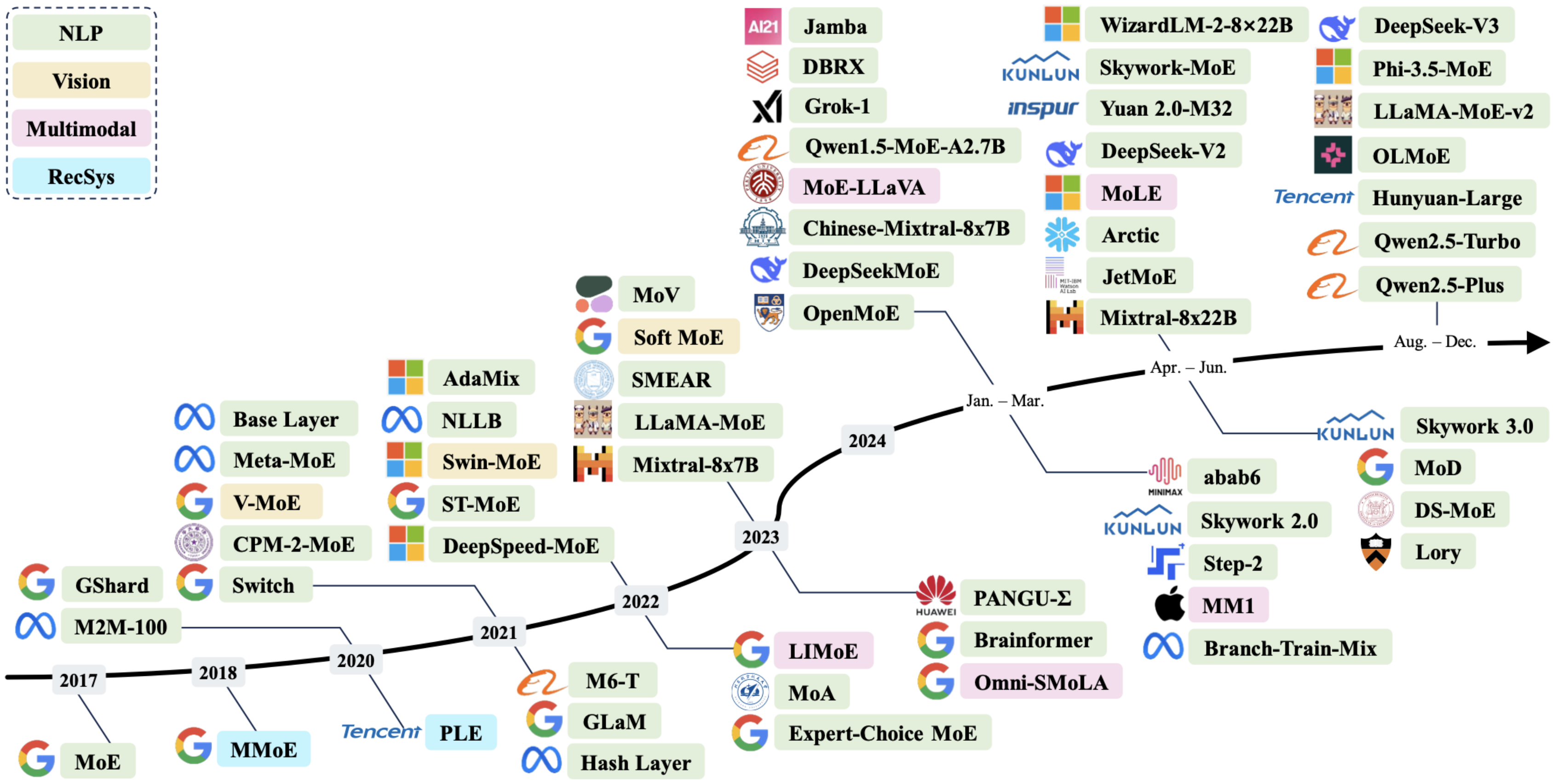


Fig. 1. A chronological overview of several representative mixture-of-experts (MoE) models in recent years. The timeline is primarily structured according to the release dates of the models. MoE models located above the arrow are open-source, while those below the arrow are proprietary and closed-source. MoE models from various domains are marked with distinct colors: Natural Language Processing (NLP) in green, Computer Vision in yellow, Multimodal in pink, and Recommender Systems (RecSys) in cyan.

https://arxiv.org/pdf/2407.06204

Beyond Finetuning: Mixture-of-Recursions (MoR)

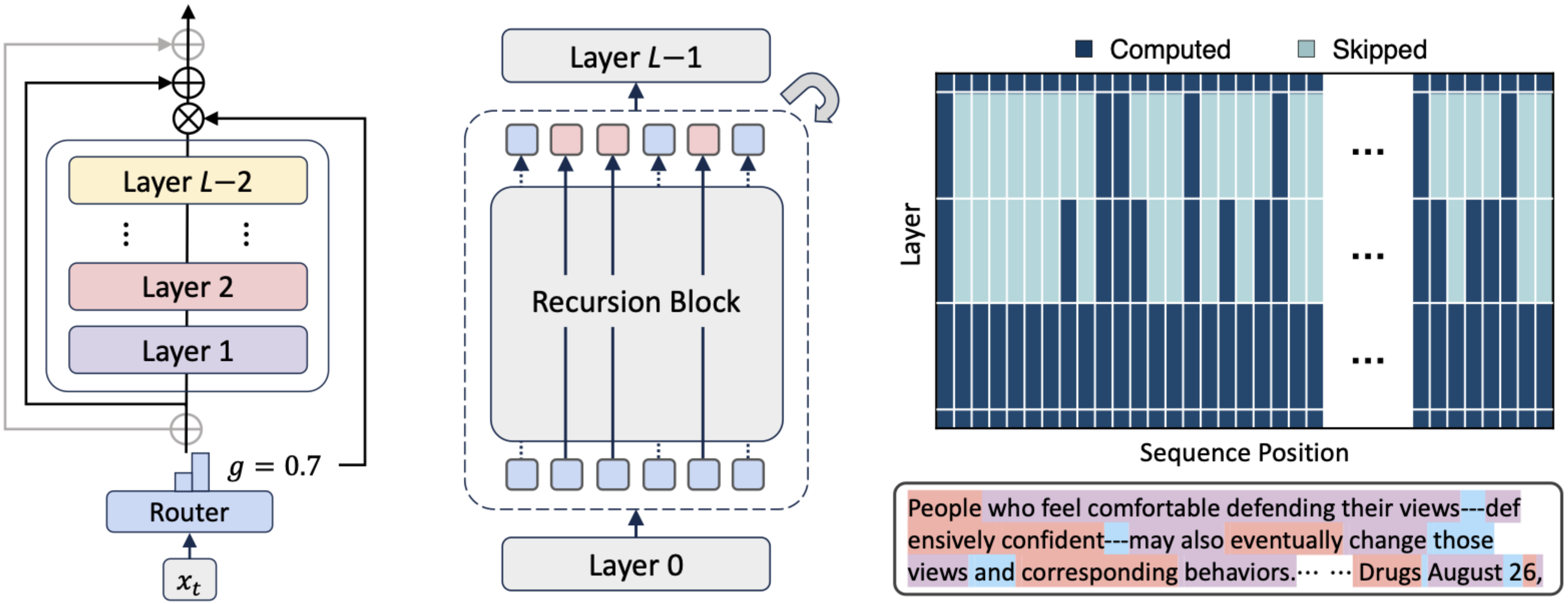


Figure 1: Overview of Mixture-of-Recursions (MoR). (*Left*) Each recursion step consists of a fixed stack of layers and a router that determines whether each token should pass through or exit. This recursion block corresponds to the gray box in the middle. (*Middle*) The full model structure, where the shared recursion step is applied up to N_r times for each token depending on the router decision. (*Right*) An example routing pattern showing token-wise recursion depth, where darker cells indicate active computation through the recursion block. Below shows the number of recursion steps of each text token, shown in colors: 1, 2, and 3.

https://arxiv.org/pdf/2507.10524