Efficient LLMs: DeepSeek Case

Incheon Paik
University of Aizu

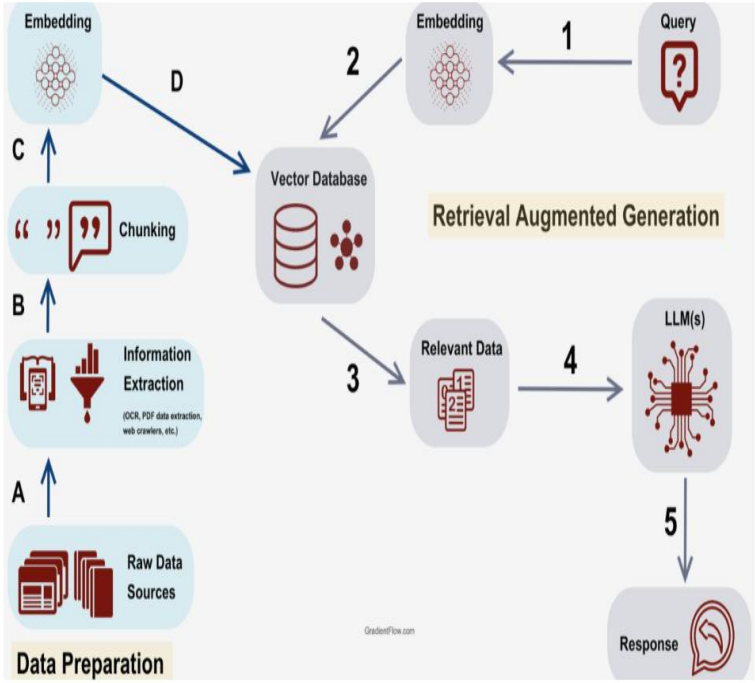
Contents

- RAG
- Mixture of Experts
- Distillation
- Quantization
- Reinforcement Learning from AI

Retrieval-Augmented Generation (RAG)

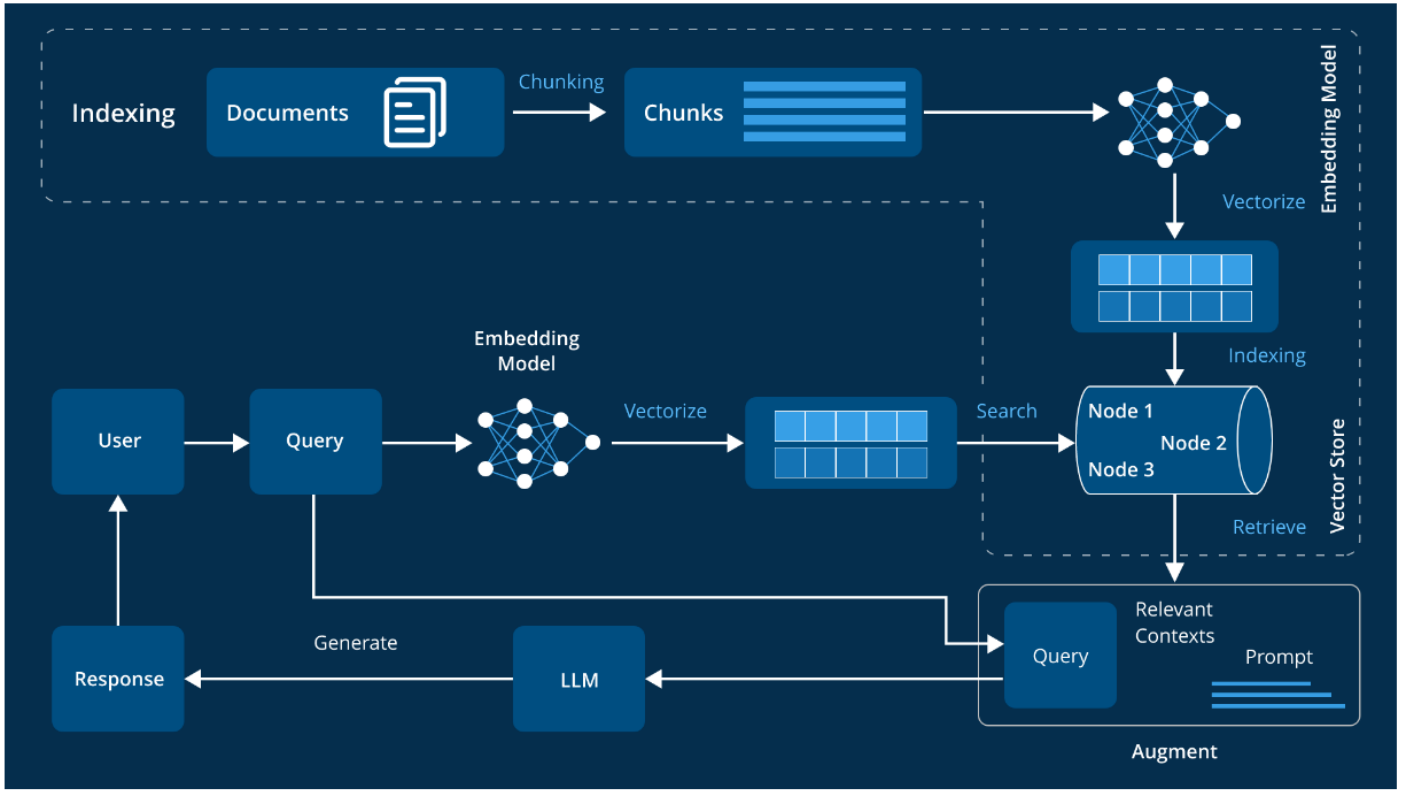
Why RAG? – Limitations of LLMs

- Hallucination: generating factually wrong answers
- Lack of up-to-date knowledge
- Weakness in domain-specific expertise (law, medicine, etc.)
- Need to integrate external knowledge
- Motivation
  - Reduce hallucinations via external knowledge
  - Enable up-to-date and time-sensitive answers
  - Domain-specific applications (legal, corporate, research QA)
  - Lower training cost (no need to memorize everything)



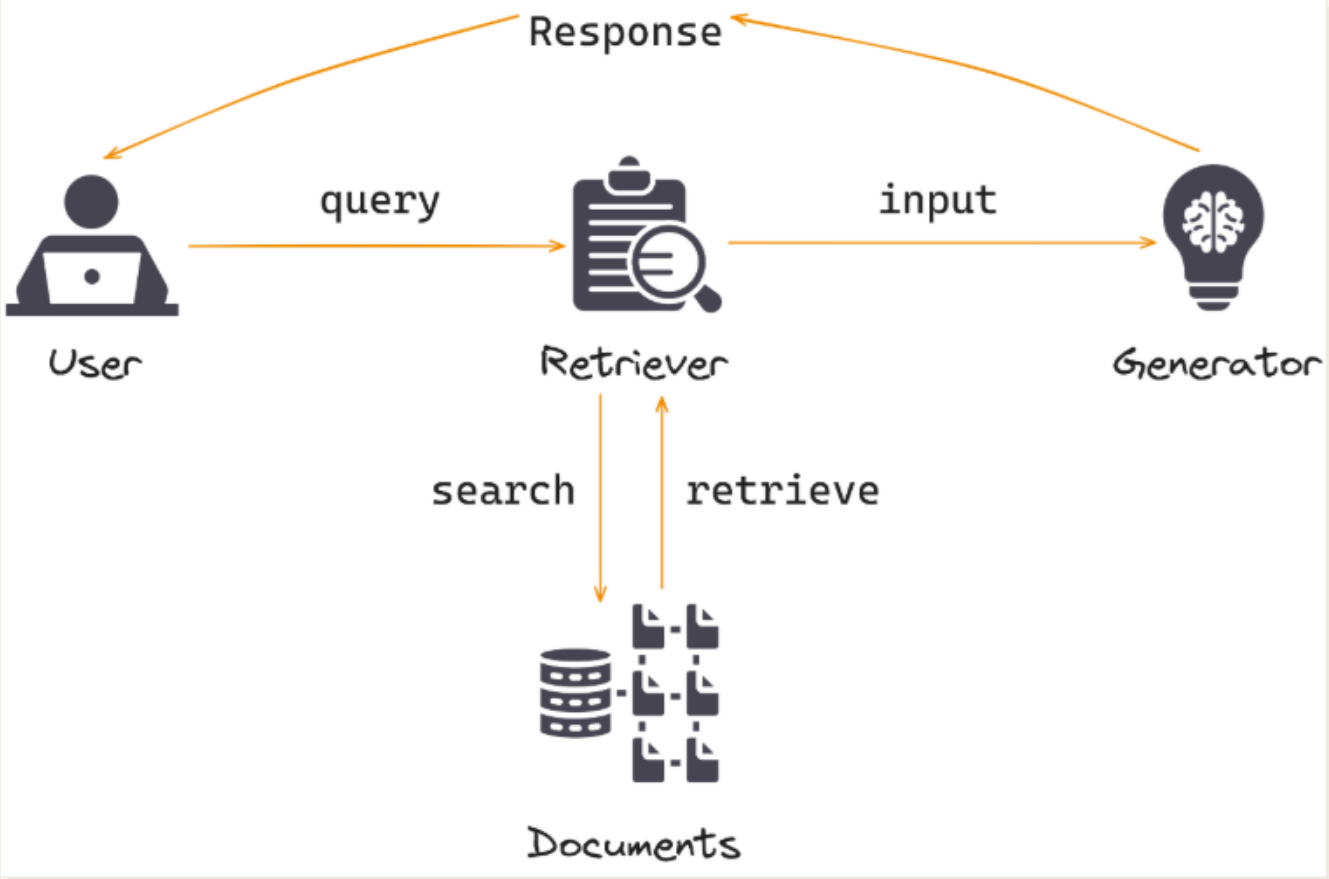
RAG Architecture

- Flow: Query → Retriever → Knowledge Base → Generator
- Retriever fetches relevant documents
- LLM integrates retrieved docs into response
- Key idea: Hybrid of Retrieval + Generation



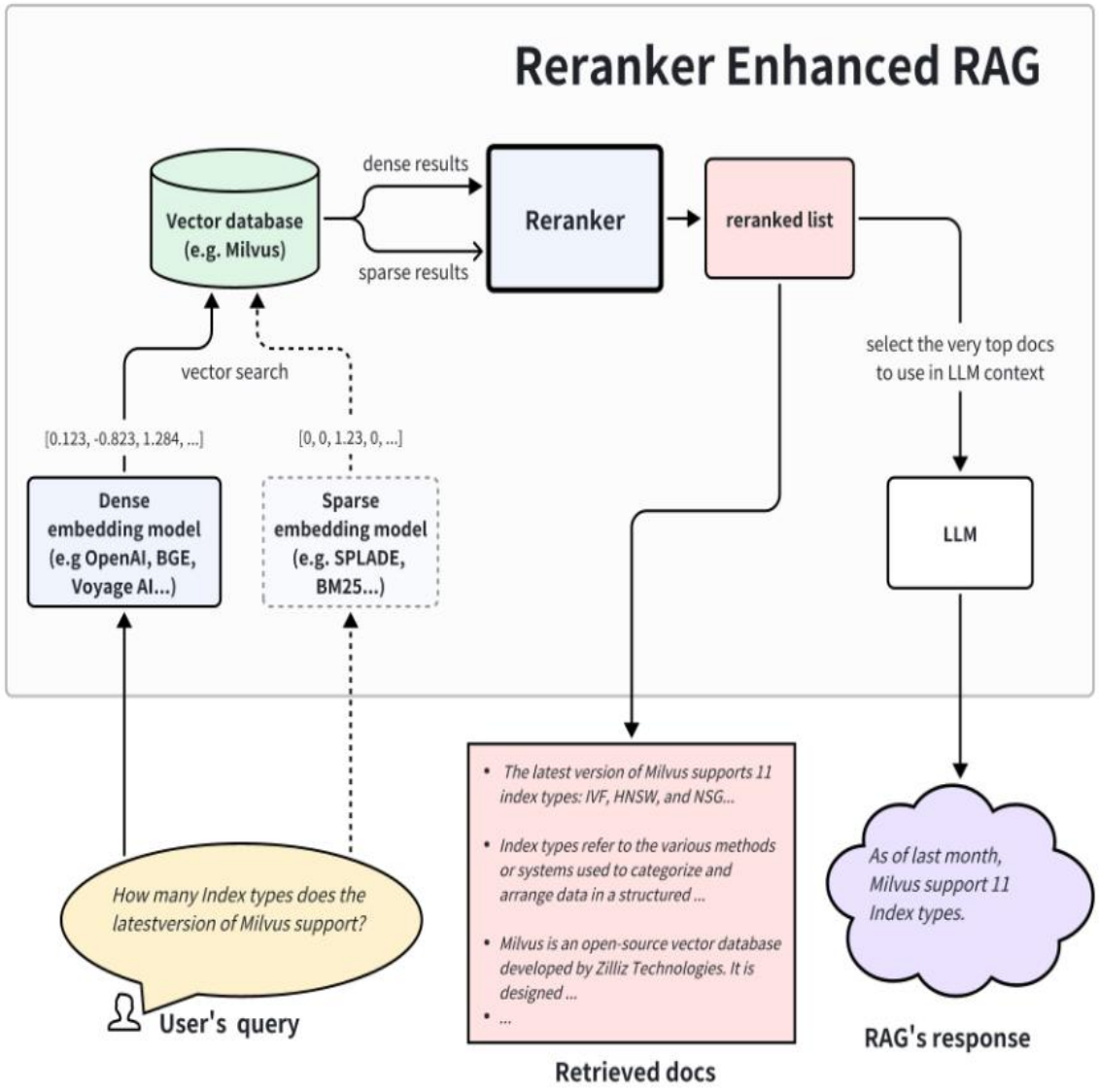
Retriever & Generator

- Retriever: Sparse (BM25) vs Dense (Embeddings: FAISS, Milvus)
- Generator: LLM generates answer conditioned on retrieved docs
- Advantage: more factual and domain-adapted answers



Key Techniques in RAG

- Retrieval methods: Sparse vs Dense
- Ranking & Filtering: Top-k selection, re-ranking
- Knowledge Base design: Vector DB, chunking strategies
- Efficiency: reduce retrieval latency, handle large-scale data

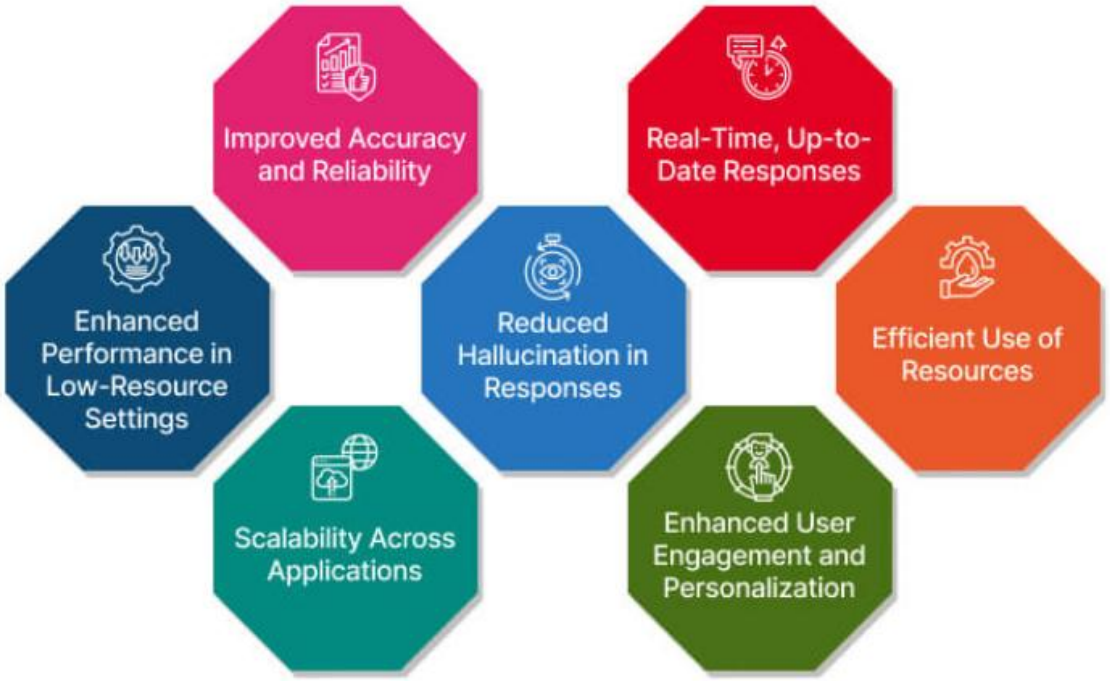


Applications of RAG

- Customer service: FAQ, chatbots
- Research: academic papers QA, legal document analysis
- Business: enterprise document QA
- News & real-time knowledge integration

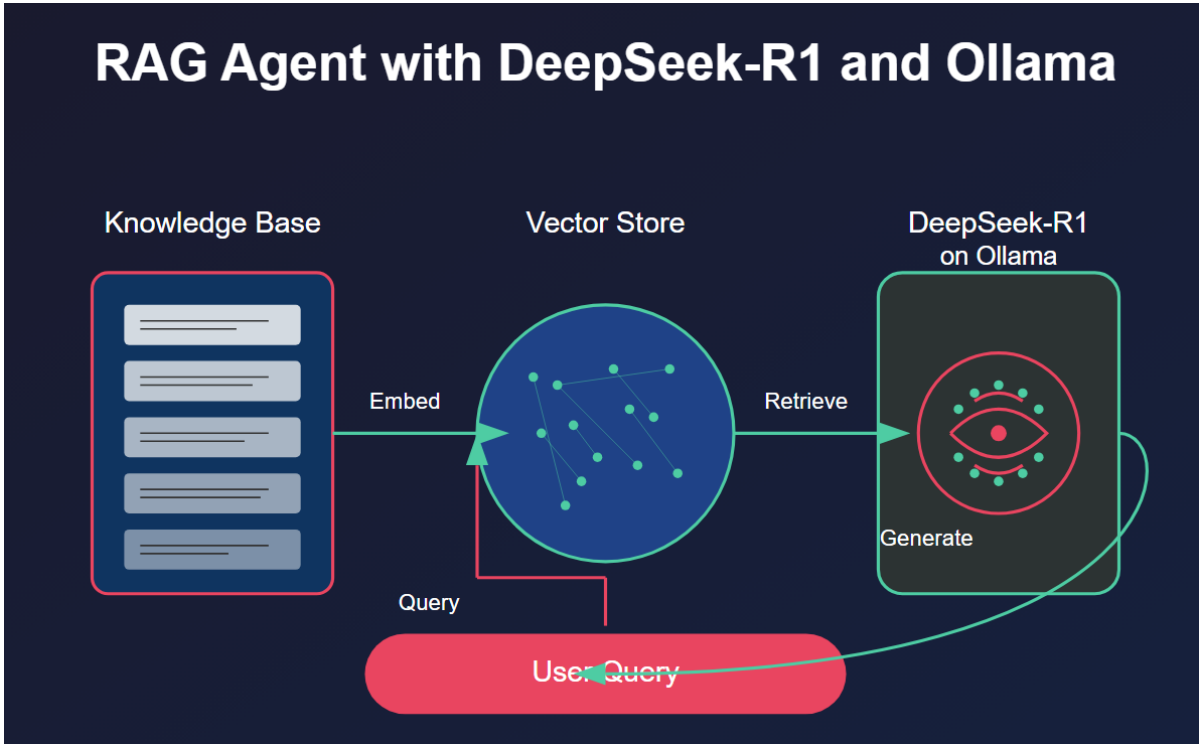
Benefits of RAG

- Reduce hallucinations
- Domain knowledge customization
- Access to latest information
- Lower training costs



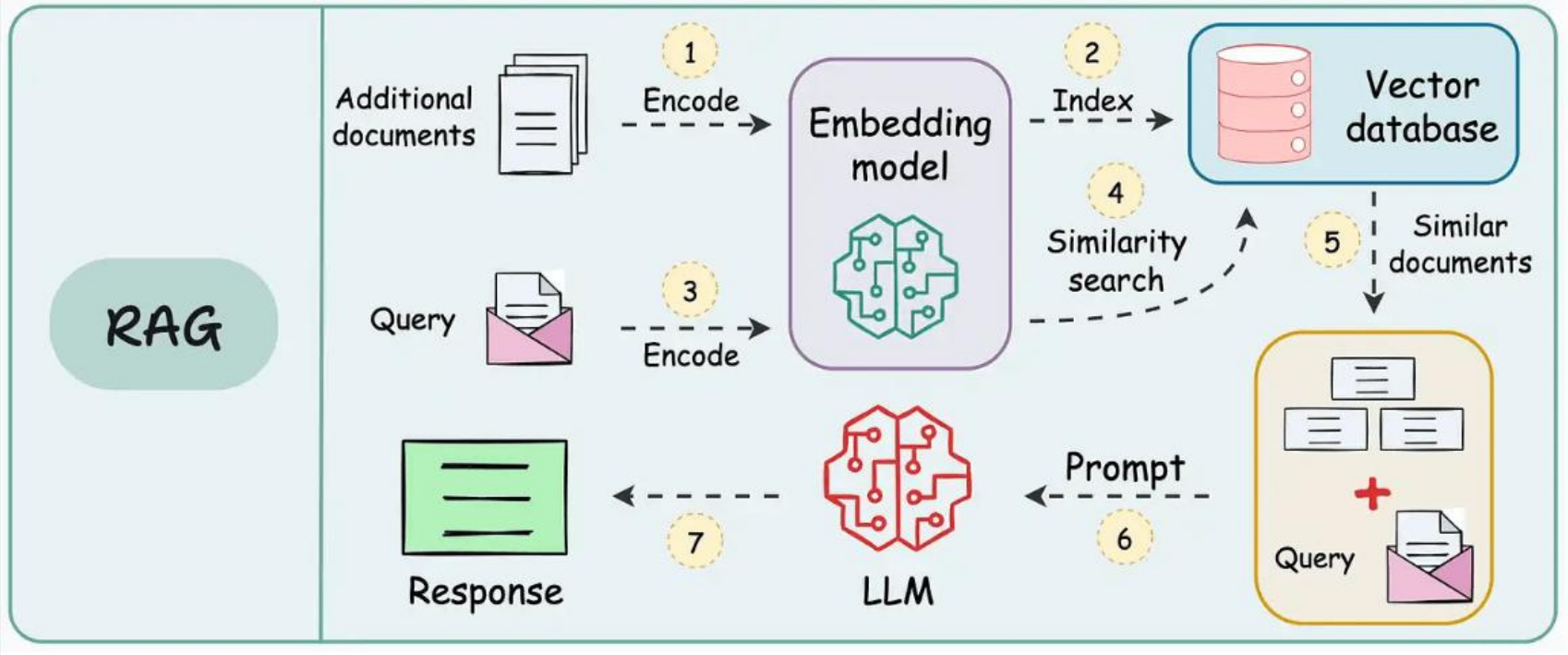
DeepSeek and Efficient RAG

- DeepSeek leverages RAG to achieve high performance with fewer parameters
- Efficiency points: optimized retrieval reduces computation cost
- External knowledge integration allows smaller models to compete



Summary for RAG

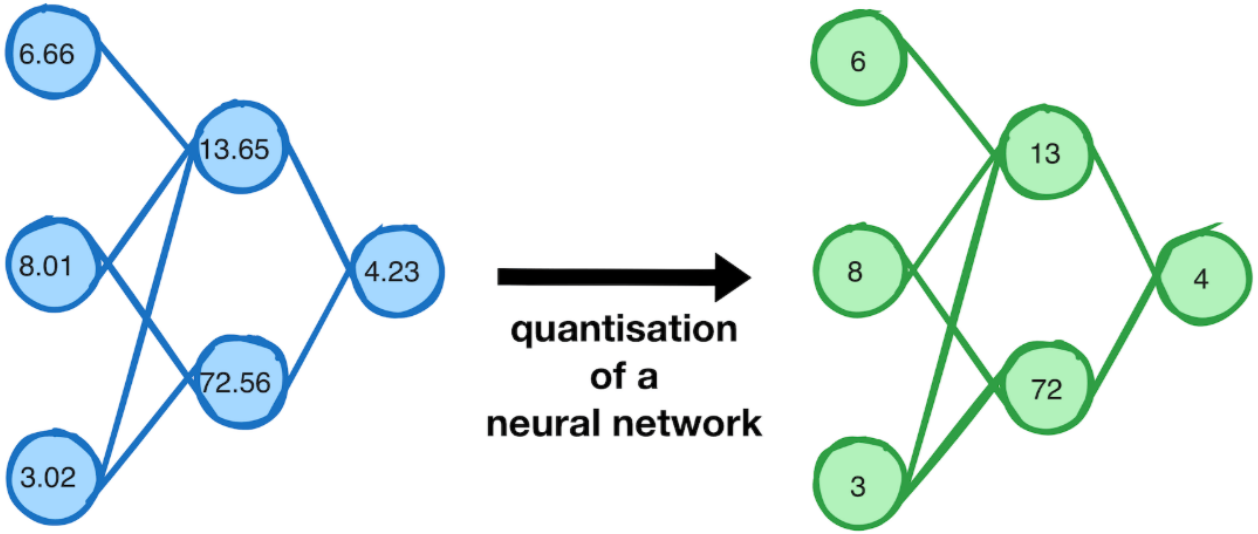
- RAG = Retrieval + Generation hybrid approach
- Solves limitations of LLMs: factuality, recency, domain adaptation
- Key factor behind Efficient LLMs such as DeepSeek



Quantization

What is Quantization? – The Basic Idea

- Representing model weights/activations with lower precision
- Example: FP32 → FP16, INT8, INT4
- Goal: reduce memory usage and computation cost
- Key benefit: faster inference, smaller deployment footprint

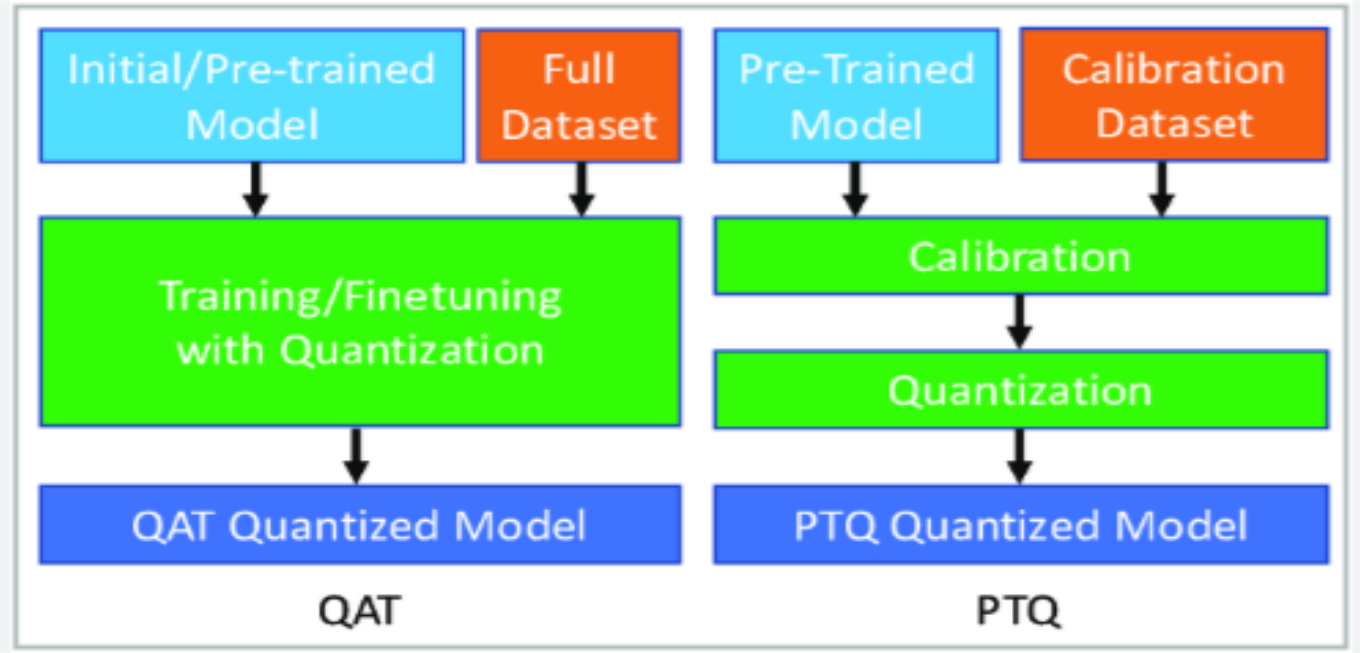


Why Quantization Matters in LLMs

- LLMs = billions of parameters → huge memory/computation demands
- Without quantization: need high-end GPUs, large VRAM
- With quantization: run on consumer GPUs, CPUs, or even mobile devices
- Essential for democratizing LLM usage

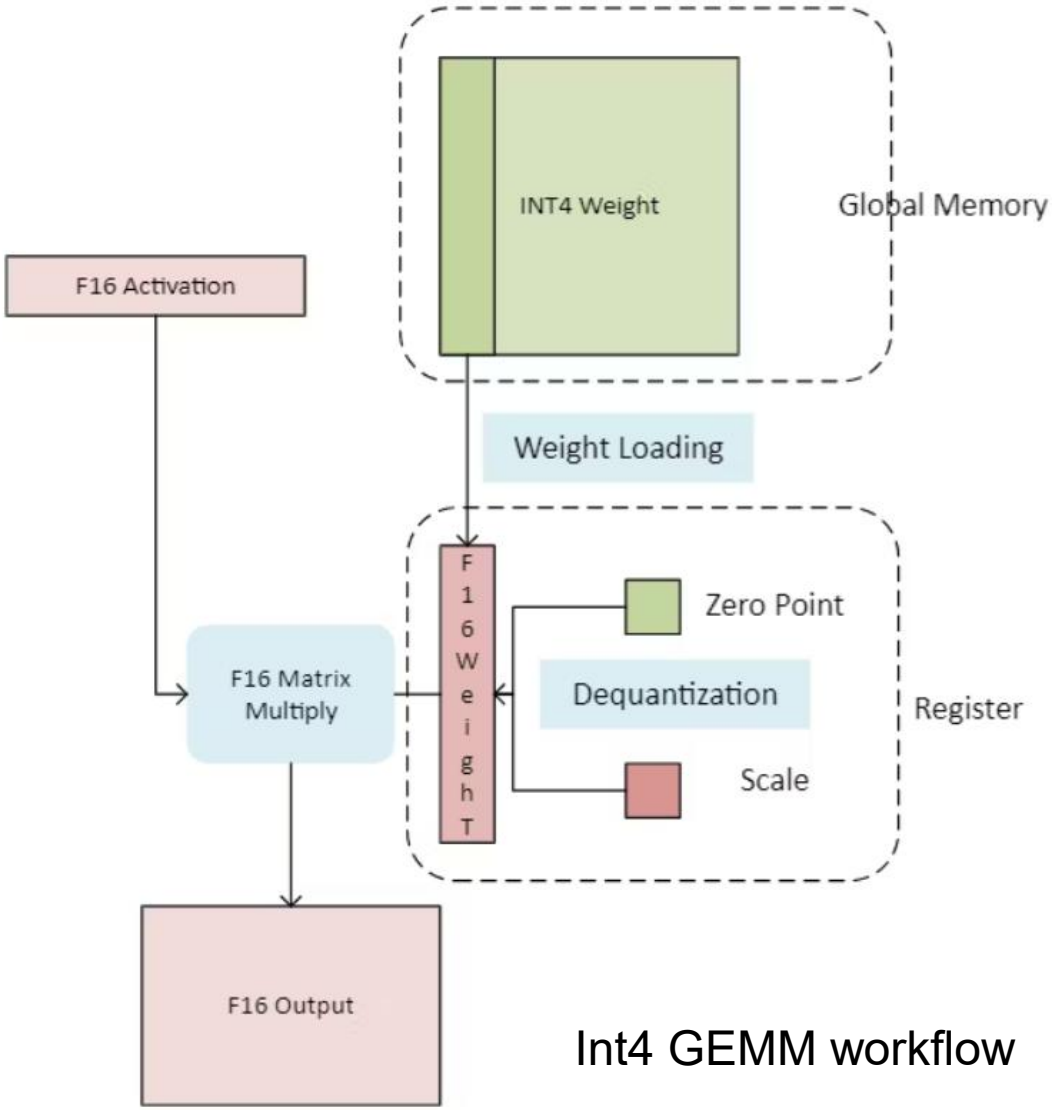
Types of Quantization (1)

- Post-Training Quantization (PTQ)
  - Apply quantization after model training
  - Easy to implement, but may lose accuracy
- Quantization-Aware Training (QAT)
  - Simulate quantization during training
  - More accurate but higher training cost



Types of Quantization (2)

- Weight-only Quantization
  - Compress weights only
  - Less accuracy loss, simpler
- Weight + Activation Quantization
  - Compress both weights and activations
  - Higher efficiency but risk of instability



Int4 GEMM workflow

Precision Levels and Trade-offs

- FP32: High precision, slow, large memory
- FP16 / bfloat16: Good balance (standard for training)
- INT8: Common for inference, strong compression
- INT4: Very compact, may harm accuracy if not optimized
- Binary: Extreme case, mostly research-level

		Recovery %	Average Score	MMLU 5-shot	MMLU CoT 0-shot	ARC-C 0-shot	GSM8k CoT 8-shot	HellaSwag 10-shot	Winogrande 5-shot	TruthfulQA 0-shot
8B	BF16	100.00	74.06	68.3	72.8	81.4	82.8	80.5	78.1	54.5
	W8A8-FP	99.31	73.55	68.0	71.6	81.2	82.0	80.0	77.7	54.3
	W8A8-INT	100.31	74.29	67.8	72.2	81.7	84.8	80.3	78.5	54.7
	W4A16-INT	98.72	73.11	66.9	71.1	80.2	82.9	79.9	78.0	52.8
70B	BF16	100.00	84.40	83.8	86.0	93.3	94.9	86.8	85.3	60.7
	W8A8-FP	99.72	84.16	83.8	85.5	93.5	94.5	86.6	84.6	60.6
	W8A8-INT	99.87	84.29	83.7	85.8	93.1	94.2	86.7	85.1	61.4
	W4A16-INT	99.53	84.00	83.6	85.6	92.8	94.4	86.3	85.5	59.8
405B	BF16	100.00	86.79	87.4	88.1	95.0	96.0	88.5	87.2	65.3
	W8A8-FP	100.12	86.89	87.5	88.1	95.0	95.8	88.5	88.0	65.3
	W8A8-INT	99.32	86.20	87.1	87.7	94.4	95.5	88.2	86.1	64.4
	W4A16-INT	99.98	86.78	87.2	87.7	95.3	96.3	88.3	87.4	65.3

Quantization Algorithms and Techniques

- Uniform vs Non-uniform Quantization
  - Uniform Quantization
    - Represents values using equal spacing between quantization levels.
    - Example: if the range is -1.0 to 1.0, split into 8 equal steps: -1.0, -0.75, -0.5, … 1.0.
    - Simple and fast to implement.
  - Non-uniform Quantization
    - Uses adaptive spacing depending on data distribution.
    - Example: if most values are near 0, place more quantization levels closer to 0.
    - Can preserve accuracy better.
- Dynamic vs Static Quantization
  - Dynamic Quantization
    - Adjusts quantization ranges during runtime based on input data distribution.
    - More flexible and adaptive, but adds runtime overhead.
  - Static Quantization
    - Fixes the quantization range before runtime, usually from training data statistics.
    - Faster and simpler at inference time.
    - Less adaptive to unseen data distributions.

Quantization Algorithms and Techniques

- LLM-specific methods: GPTQ, AWQ, SmoothQuant
  - GPTQ (GPT Quantization)
    - Weight-only quantization method for LLMs.
    - Maintains accuracy well, even with 4-bit quantization.
  - AWQ (Activation-aware Weight Quantization)
    - Takes activation distributions into account when quantizing weights.
    - Minimizes accuracy drop during inference.
  - SmoothQuant
    - Handles both weights and activations.
    - “Smooths” activations to make them easier to quantize, then applies weight quantization.
    - Widely used in very large LLMs.

Applications in LLMs

- Running 70B+ parameter models on single GPU via INT4
- Enabling on-device LLMs (smartphones, edge devices)
- Reducing inference costs in production (cloud services)
- Example: DeepSeek + quantization → competitive performance with fewer resources

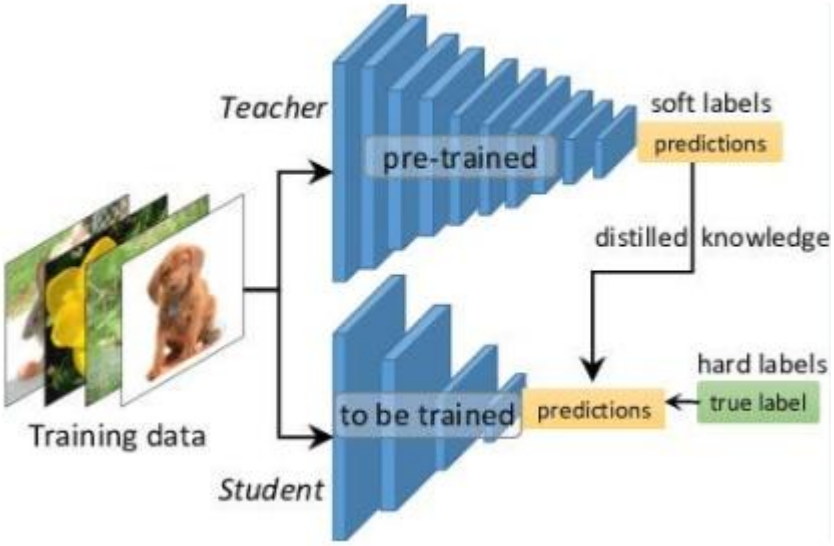
Benefits and Challenges

- Benefits
  - Smaller memory footprint
  - Faster inference speed
  - Cheaper deployment
- Challenges
  - Accuracy degradation risk
  - Sensitive to distribution of weights/activations
  - Requires algorithmic tuning (e.g., per-channel quantization)

Distillation

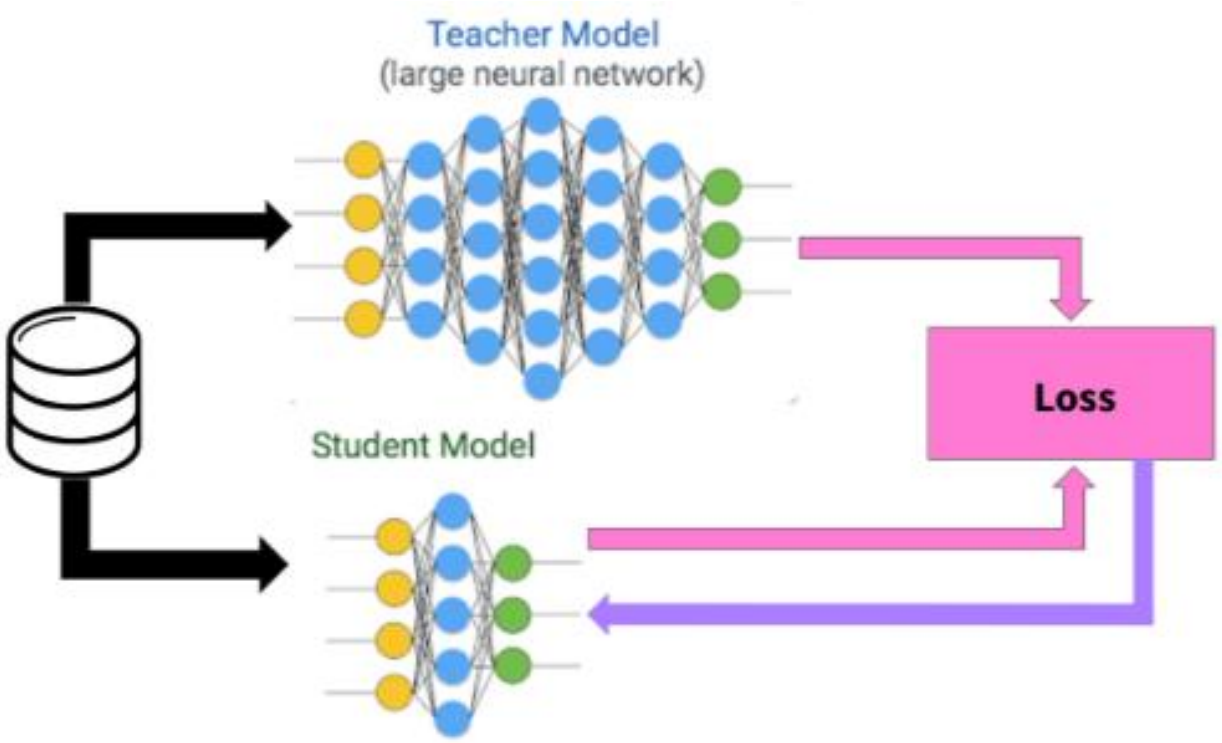
What is Knowledge Distillation? – The Concept

- Teacher Model (large, high-performance) → Student Model (smaller, efficient)
- Student learns to mimic teacher's predictions
- Uses both:
  - Hard labels (true answers)
  - Soft labels (teacher's probability distribution)
- Goal: compact model with near-teacher accuracy



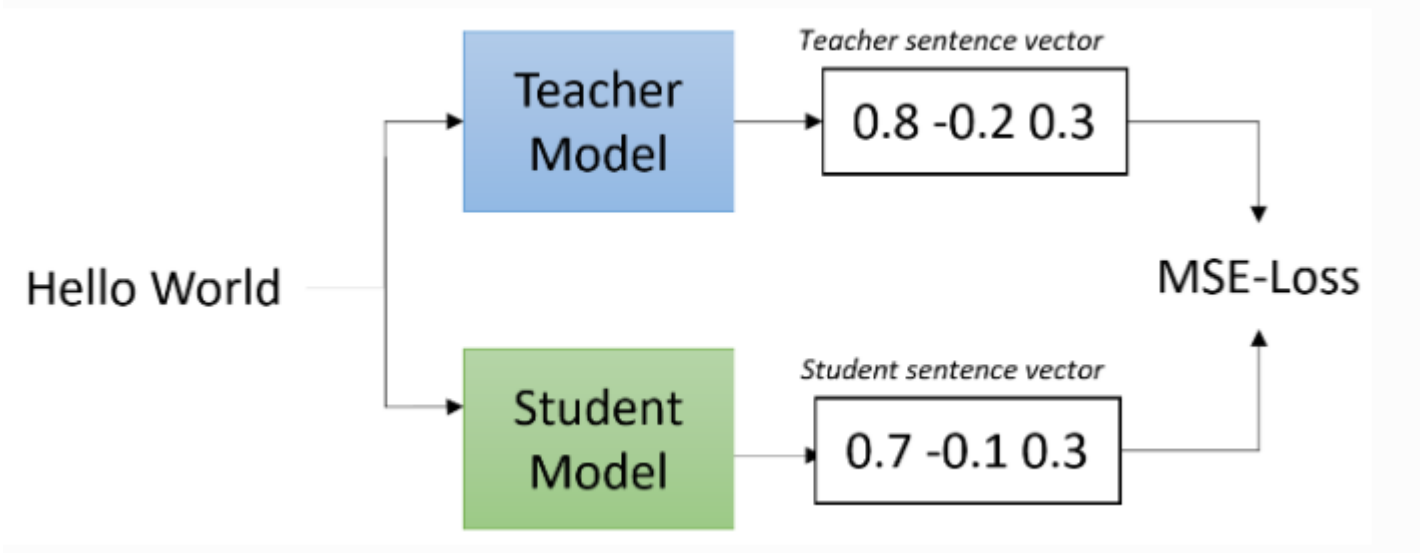
Why Distillation? – Motivation

- Large LLMs → expensive (compute, memory, inference time)
- Deployment challenges: cloud cost, latency, device limitations
- Need for smaller models without major accuracy loss
- Distillation: “Small but powerful”



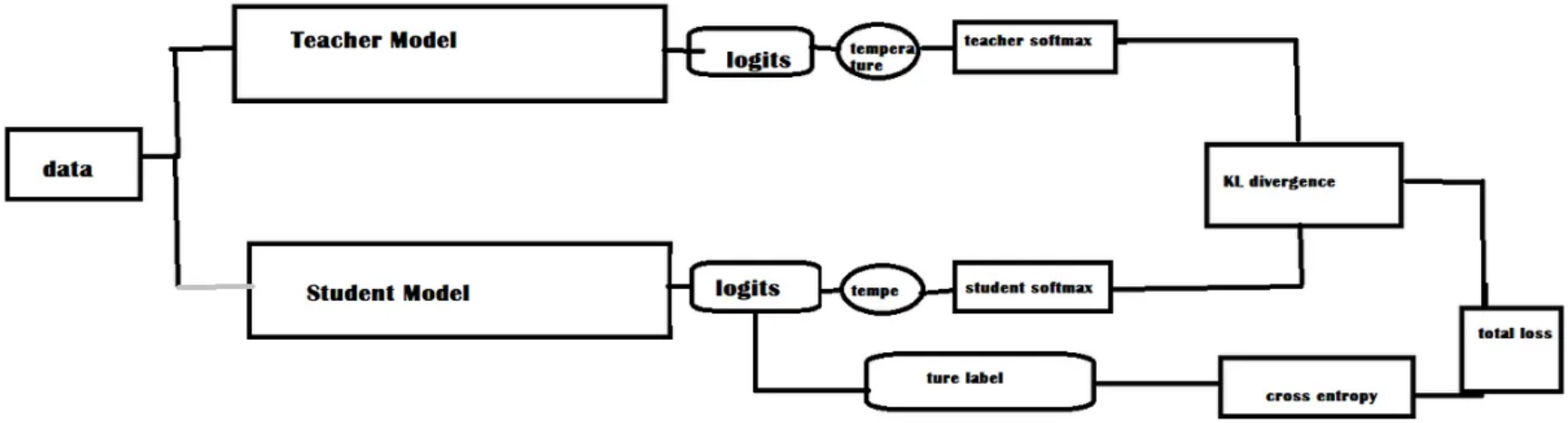
How Distillation Works – Core Mechanism (1)

- Teacher generates probability distribution for each output
- Example:
  - Cat: 0.92, Dog: 0.05, Fox: 0.03
  - True label: Cat (hard 1/0)
- Student learns richer supervision than 1/0 labels



How Distillation Works – Core Mechanism (2)

- Temperature scaling: smoothens teacher output for better transfer
- Loss function = Weighted sum of:
  - Cross-entropy with hard labels
  - KL-divergence between teacher and student soft labels
- Student learns both correctness and nuances



Types of Knowledge Distillation

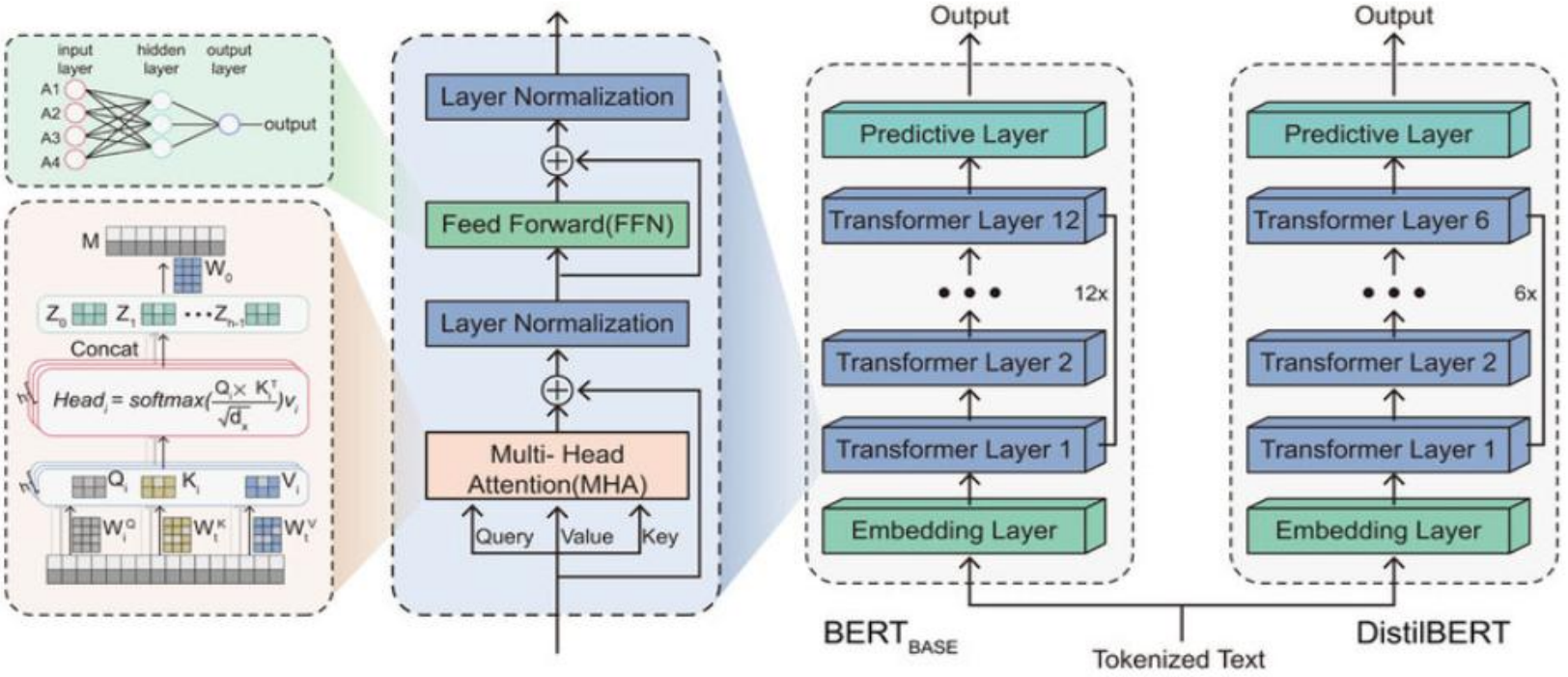
- Logit-based Distillation
  - Student mimics teacher's output distribution
  - The most common and widely used method (e.g., DistilBERT).
  - Often applies temperature scaling to smooth the teacher's outputs, making it easier for the student to learn.
  - Transfers not only the correct answer but also the teacher's "confidence in wrong answers," which improves generalization.
- Feature-based Distillation
  - Student mimics teacher's hidden layer features
  - Allows the student to inherit not only the final prediction but also the teacher's internal reasoning process and language/semantic knowledge.
  - Example: the student aligns with intermediate Transformer layer embeddings.
  - Particularly effective in NLP and vision models where contextual embeddings carry rich information.

Types of Knowledge Distillation

- Relation-based Distillation
  - Student mimics relations (attention maps, similarity patterns)
  - Focuses on preserving structural knowledge rather than raw values.
  - Example: student learns to reproduce teacher's attention distributions or pairwise token similarities.
  - Helps the student follow the teacher's reasoning patterns, not just outputs.
- In LLMs: often combined (hybrid)
  - Basic: Large Language Models (LLMs) often combine multiple types of distillation.
  - Logit-only distillation cannot capture reasoning, while Feature/Relation distillation can be computationally expensive.
  - Hybrid approaches balance final outputs + intermediate features + relational structure for best performance.
  - Recent research tailors hybrid strategies for specific tasks (e.g., QA, reasoning).

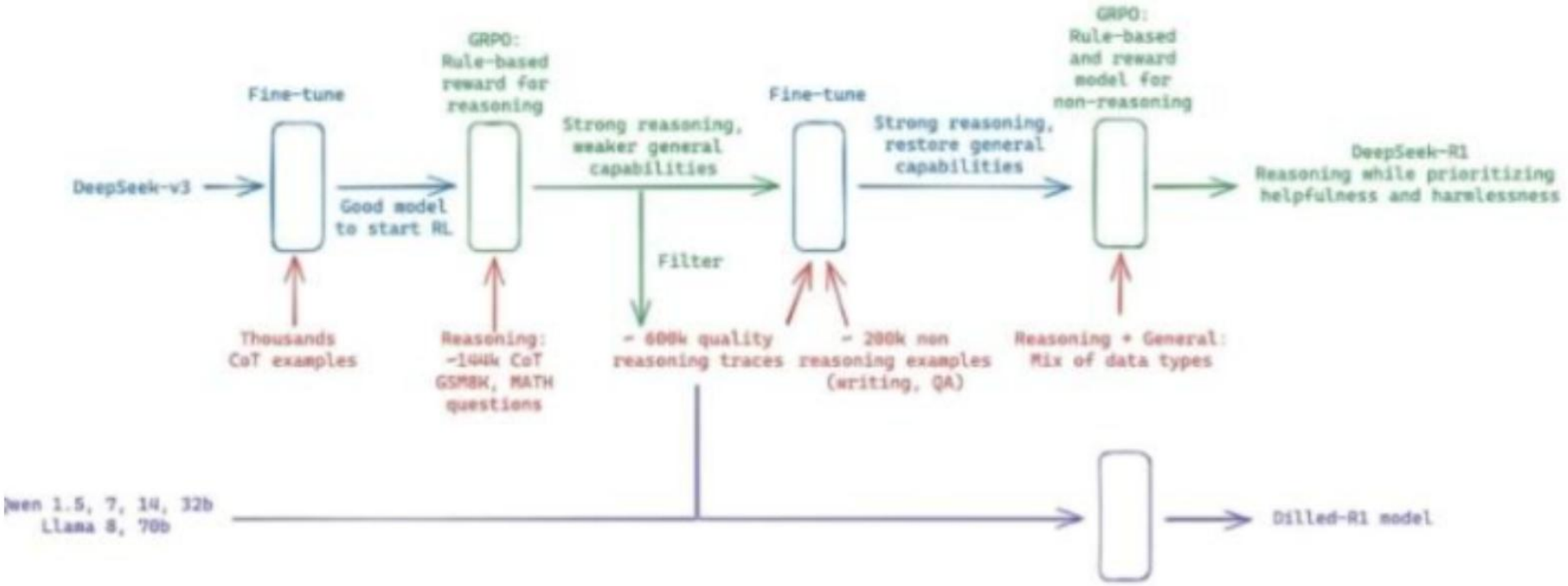
Applications in LLMs (1)

- DistilBERT
  - 40% smaller than BERT
  - 60% faster inference
  - Retains ~97% accuracy
- TinyBERT, MiniLM → other examples



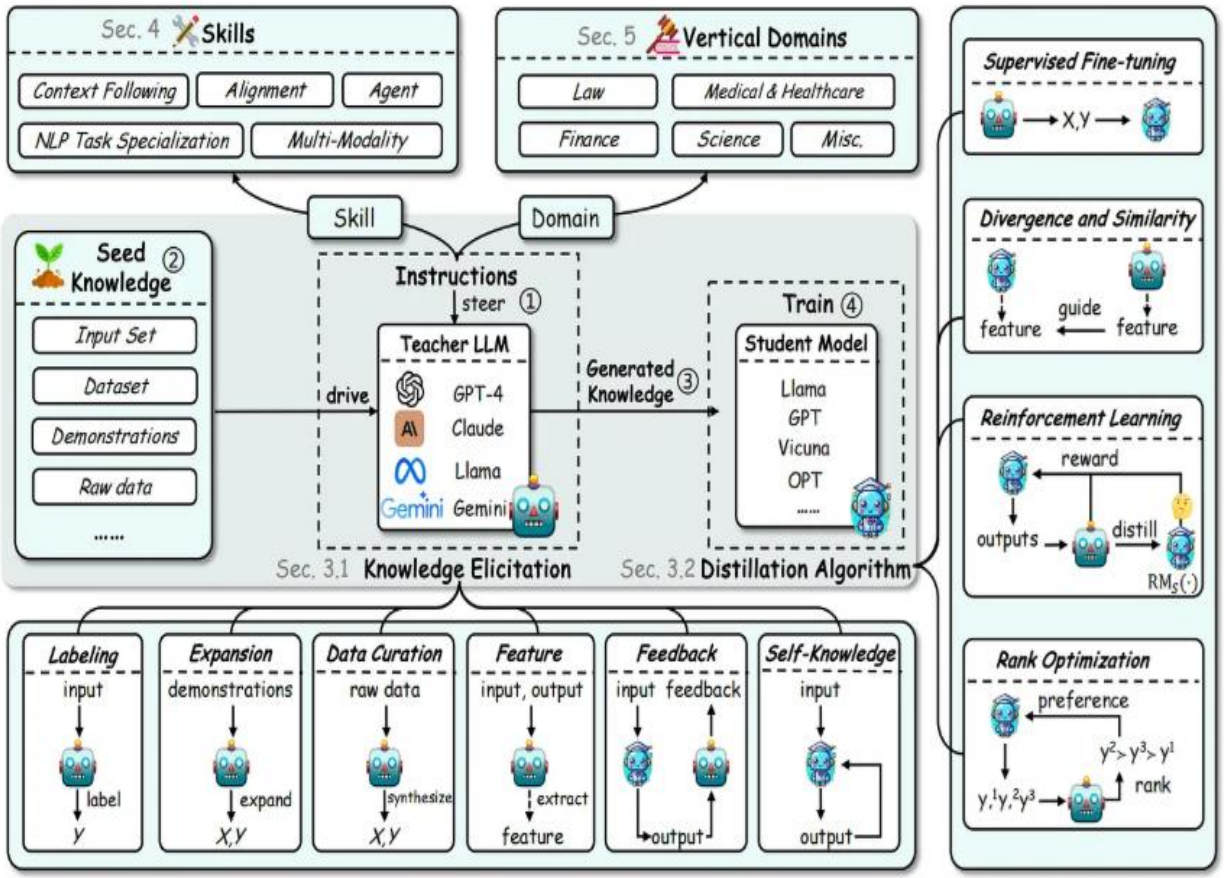
Applications in LLMs (2)

- GPT-family distillation: smaller GPT models distilled from GPT-3/GPT-4
- DeepSeek: efficient variants distilled from larger experts
- On-device LLMs: mobile, edge, IoT scenarios



Benefits and Challenges

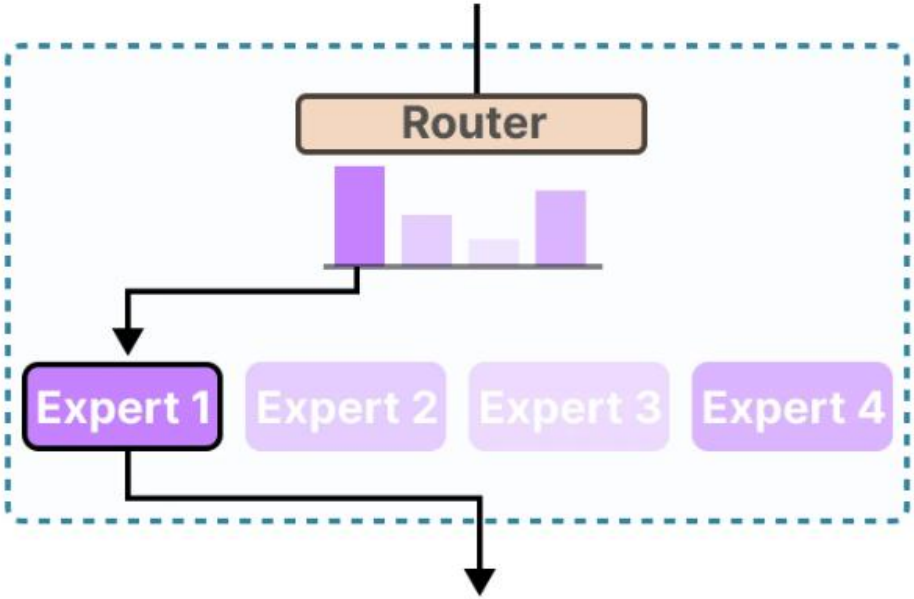
- Benefits
  - Smaller, faster models
  - Lower deployment cost
  - Better generalization via soft labels
- Challenges
  - Risk of losing nuanced capabilities
  - Sensitive to quality of teacher model
  - Limited transfer for complex reasoning



Mixture of Experts

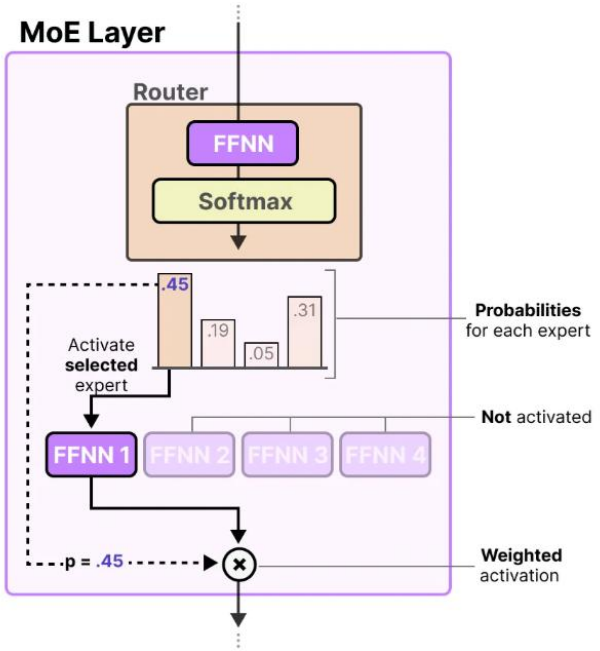
What is MoE? – The Concept

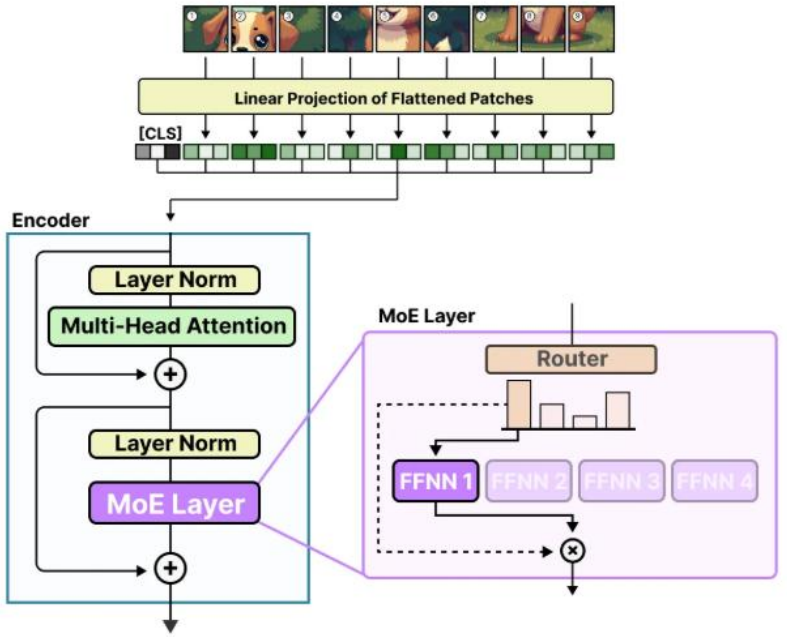
- Traditional dense models: all parameters active for every input
- MoE: model contains multiple “experts” (subnetworks)
- For each input, only a small subset of experts is activated
- Idea: scale model size without proportional compute increase



Why MoE? – Motivation

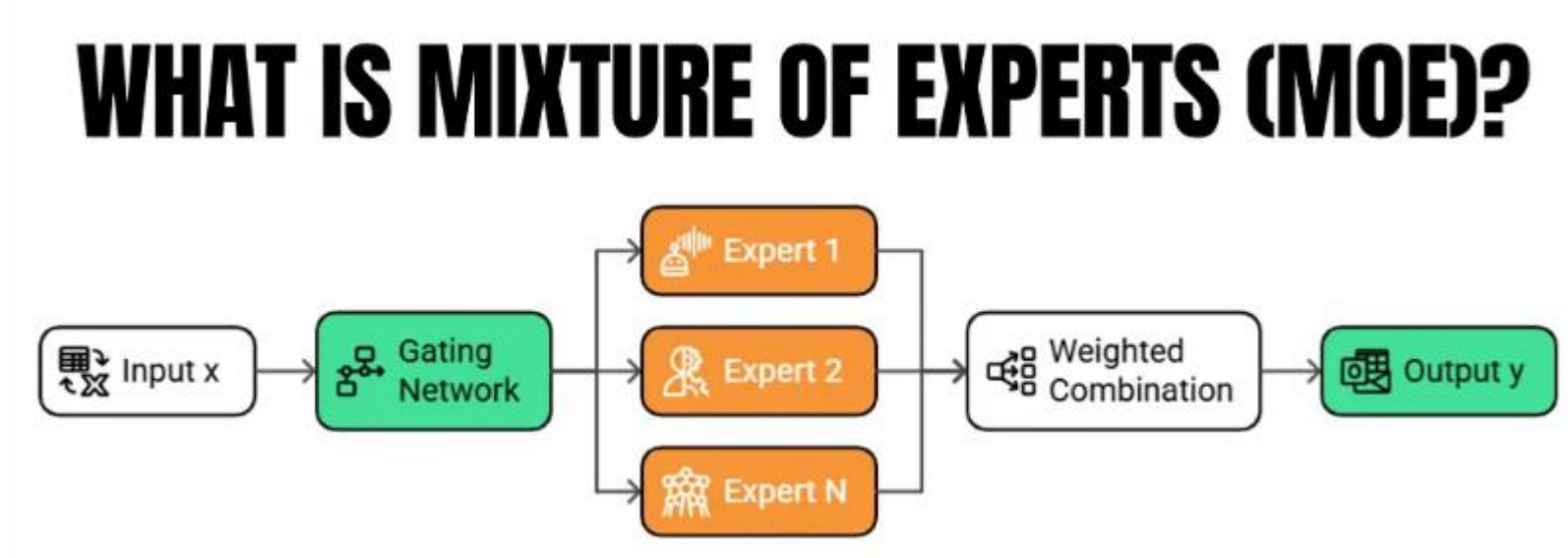
- Dense LLMs → compute and memory cost grow linearly with size
- Training trillion-parameter dense models = infeasible
- MoE allows:
  - Huge model capacity (knowledge storage)
  - Limited compute cost per inference step
- Achieves efficiency + scalability simultaneously





How MoE Works – Gating Mechanism

- Gating network chooses which experts to activate for each token
- Top-1 routing: send token to the single best expert
- Top-2 routing: send to two experts, average/merge results
- Challenge: prevent “expert collapse” (some experts overused, others idle)
- Solution: load balancing losses, routing regularization



How MoE Works – Gating Mechanism

- In a standard neural network, all parameters (all neurons/layers) are used for every input → very costly and inefficient.
- Mixture of Experts (MoE) prepares multiple expert networks in parallel, but for each input, only a few experts are activated.
- This way, the model can scale up the total parameter count while reducing actual computation per input.

- Router (Gate) Output

$$h(x) = W_r \cdot x$$

  - x: input token (e.g., word embedding)
  - W_r: weight matrix of the router/gating network
  - $h(x)$: logits (scores) for each expert

How MoE Works – Gating Mechanism

- Probability of Selecting an Expert (Softmax)

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_j^N e^{h(x)_j}}$$

- $p_i(x)$: probability that expert i should process input x
- N: total number of experts
- Softmax converts scores into probabilities

Example: "Expert 1 = 0.7, Expert 2 = 0.2, Expert 3 = 0.1 ..."

- Final Output (Weighted Sum of Top-k Experts)

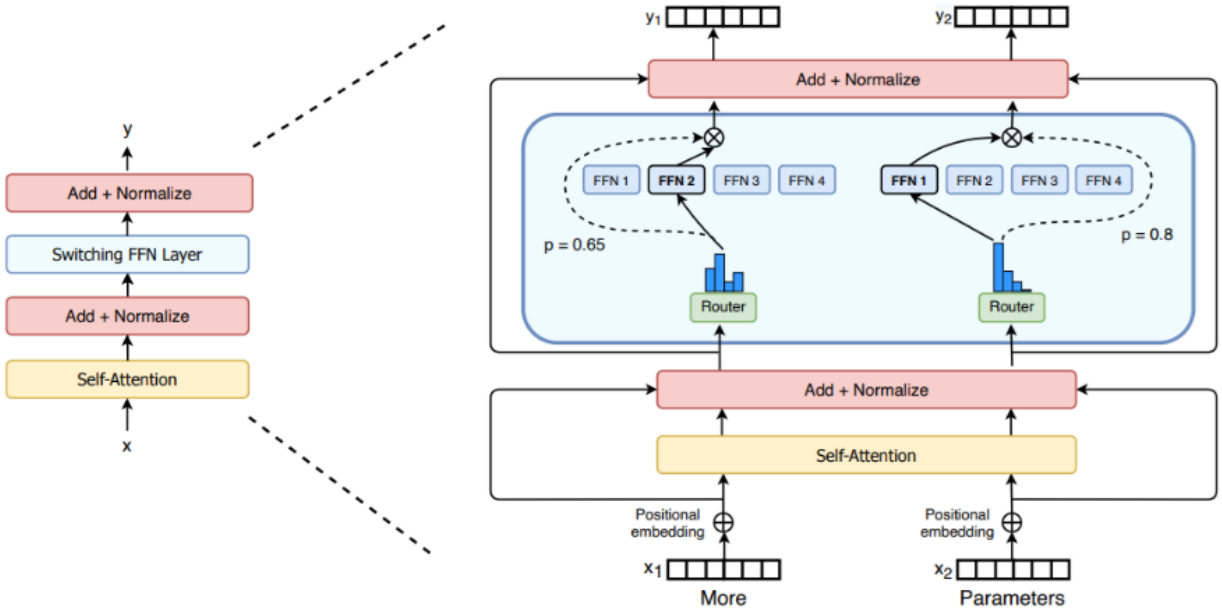
$$y = \sum_{i \in \tau} p_i(x) \cdot E_i(x)$$

- $E_i(x)$: output of the i-th expert
- τ: set of top-k selected experts (e.g., Top-2)
- Output is a **weighted combination** of the chosen experts

Example: If Expert 1 = 0.7, Expert 2 = 0.2, Expert 3 = 0.1,
Top-2 are chosen → Final output = $0.7E_1(x) + 0.2E_2(x)$.

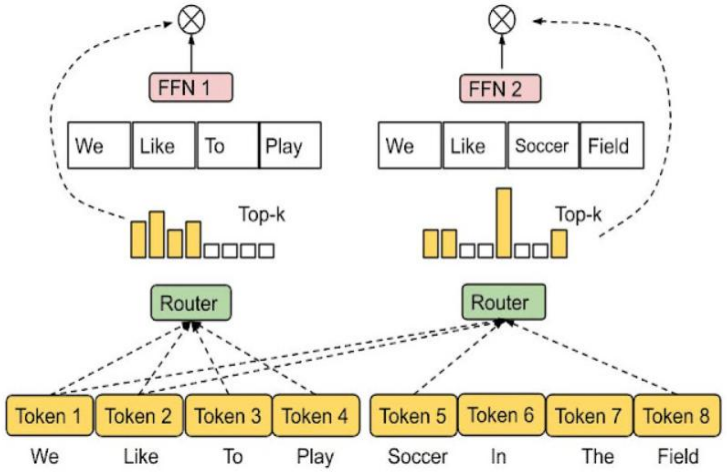
Design Variants (1)

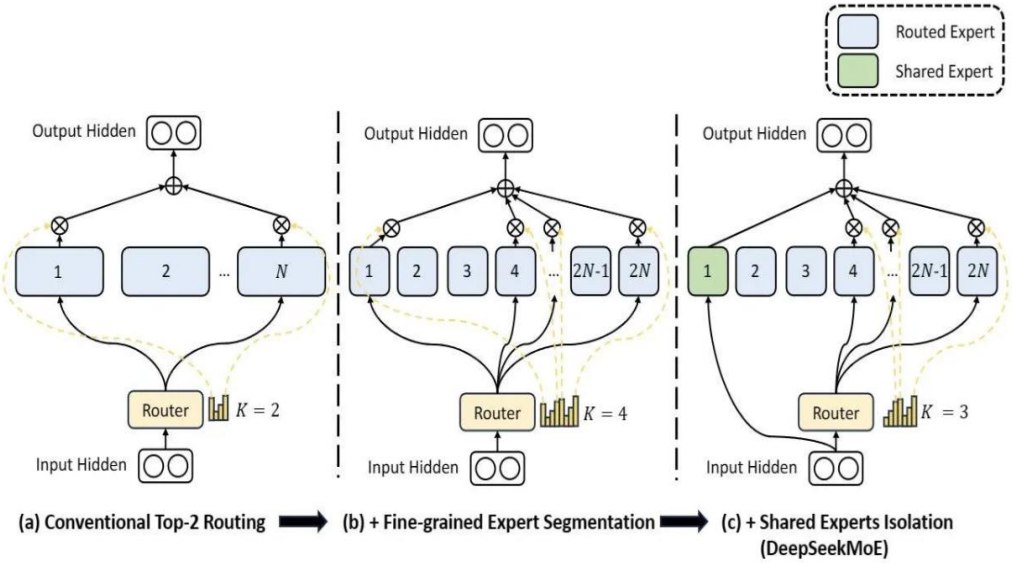
- Switch Transformer (Google, 2021)
  - Uses Top-1 routing for maximum efficiency
  - Scales to trillions of parameters with manageable compute
- GShard
  - Early large-scale distributed MoE by Google
  - Key step toward scaling language models



Design Variants (2)

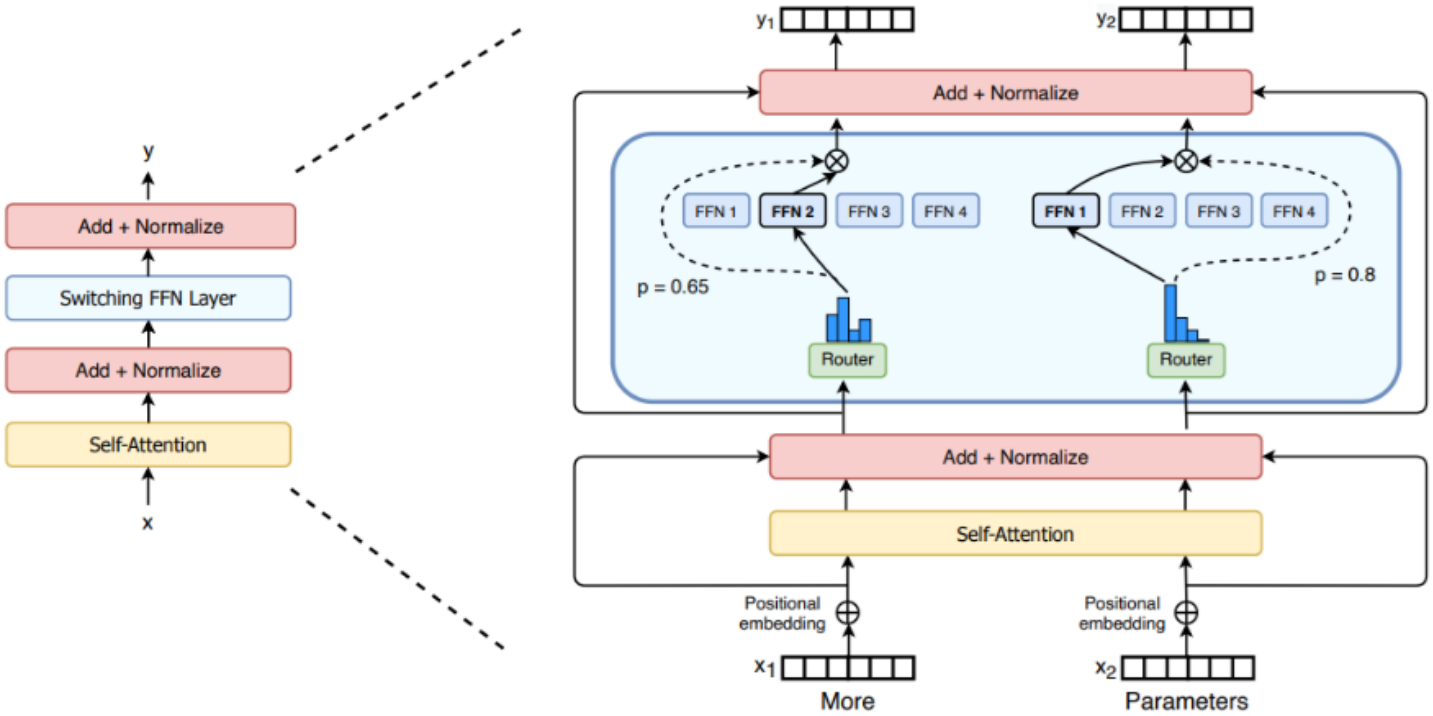
- GLaM (Google, 2022)
  - Mixture of Experts with fewer active parameters per input
  - Balanced efficiency vs performance
- DeepSeek-MoE
  - Hybrid design: shared experts + specialized experts
  - Improves stability and reduces redundancy
  - Key factor for DeepSeek's high efficiency





Applications in LLMs (1)

- Large-scale LLMs with MoE:
  - Google's Switch Transformer, GLaM
  - DeepSeek
  - Mixtral (open-source MoE)
- MoE made trillion-scale models practically trainable



Applications in LLMs (2)

- Efficiency gains:
  - In dense models, all parameters are activated for every input, but in MoE, only a small subset of experts (Top-k) is used → significantly reduces computation.
  - As a result, inference speed can be close to that of smaller dense models, while the capacity remains much larger.
  - Reduces memory bandwidth requirements, improving GPU utilization.
  - Gains are especially noticeable for long input sequences (e.g., documents, code).
- Deployment: enables cost-effective large models
  - Running models with billions or even trillions of parameters is extremely expensive.
  - MoE lowers actual inference costs (power, GPU time) because only a fraction of parameters are active.
  - Makes it more feasible to run large models not only in the cloud but also in on-premise servers.

Applications in LLMs (2)

- Deployment: enables cost-effective large models
  - Easier maintenance: only specific experts may need to be updated instead of retraining the whole model.
- Research frontier: combining MoE with quantization and distillation
  - MoE is already efficient, but combining it with quantization and distillation makes models even lighter.
  - Quantizing active expert weights to INT8/INT4 → further memory savings.
  - Distilling knowledge from a large MoE teacher model into a smaller student → makes deployment possible on limited hardware.
  - Current research focuses on “Efficient MoE + Quantization + Distillation” as a package for building cost-effective yet powerful LLMs.
  - DeepSeek leverages this strategy to compete with models like ChatGPT and Gemini.

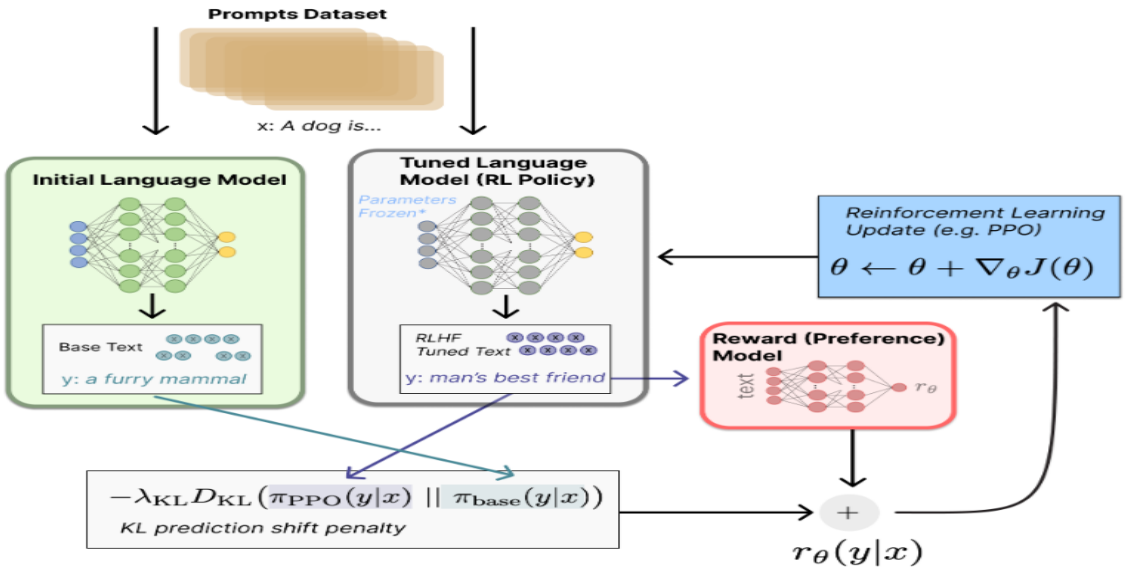
Benefits and Challenges

- Benefits
  - Huge capacity without proportional compute cost
  - Flexible specialization across experts
  - Improved efficiency for LLM scaling
- Challenges
  - Routing instability (expert imbalance)
  - Higher complexity in implementation
  - Training overhead for balancing losses

Model Alignment by Reinforcement Learning

RLHF – The Concept

- Reinforcement Learning with Human Feedback (RLHF)
- Standard pipeline:
  - Pretrain LLM on large corpus
  - Fine-tune with Supervised Fine-Tuning (SFT)
  - Train a Reward Model (RM) using human preference data
  - Optimize LLM policy using RL guided by reward model
- Purpose: align LLM outputs with human values/preferences



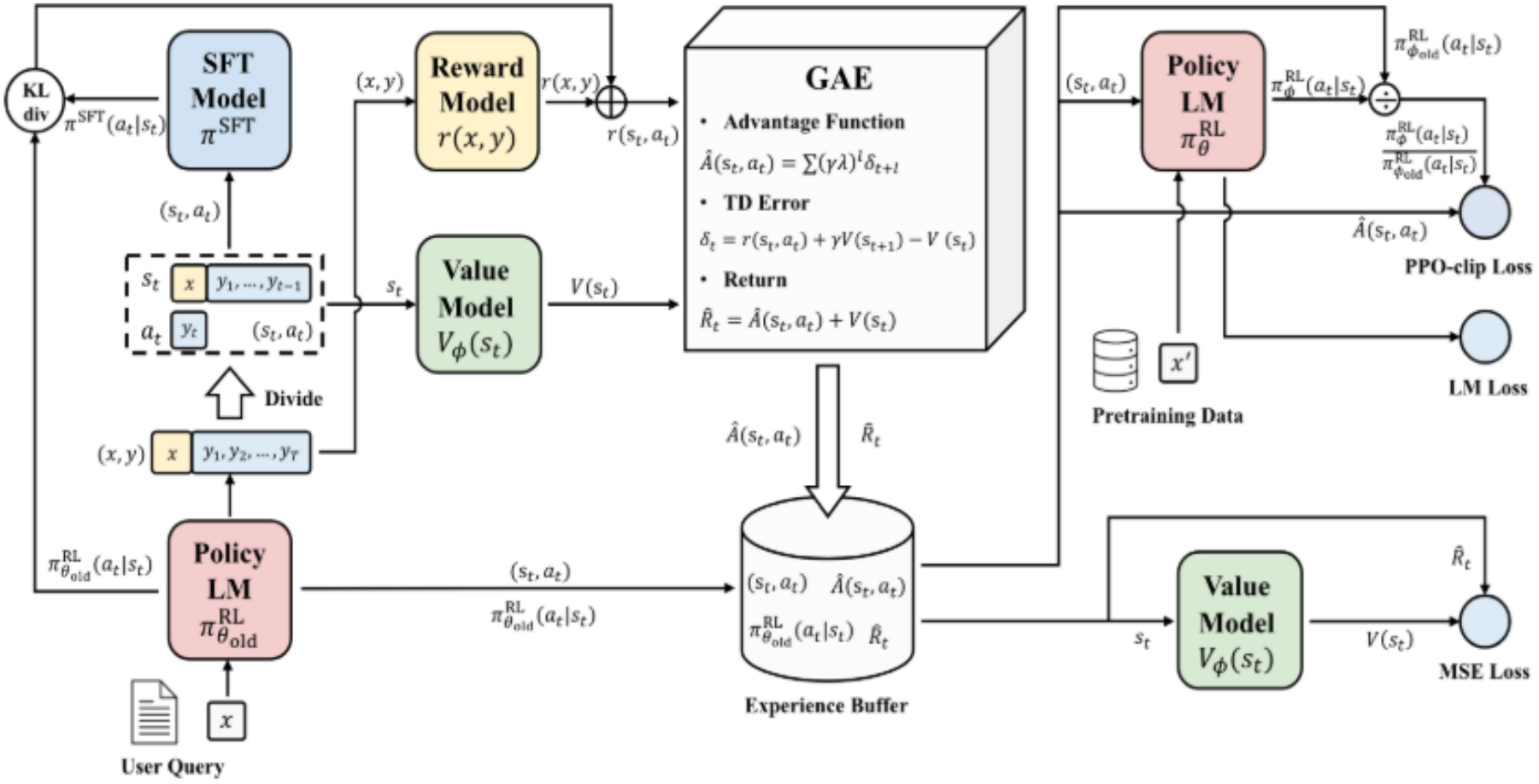
RLHF – Strengths and Limitations

- Strengths
  - Improves helpfulness, safety, alignment with user expectations
  - Provides structured feedback beyond raw labels
- Limitations
  - Human labeling = expensive, time-consuming
  - Reward model may be biased or inconsistent
  - Training is complex (RL adds instability, high compute cost)

PPO in RLHF

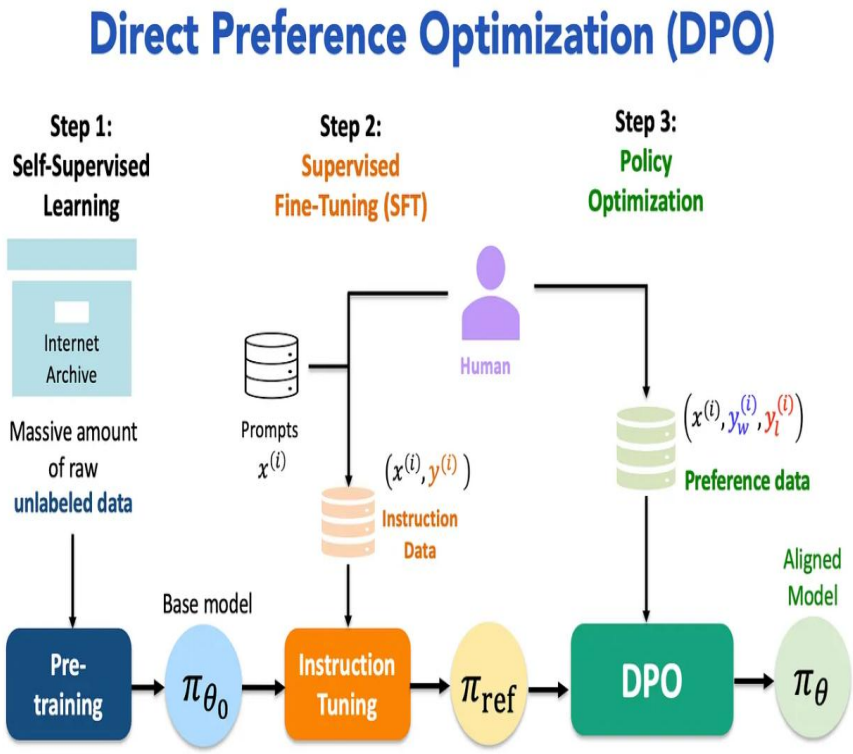
- Proximal Policy Optimization (PPO) = standard RL algorithm for RLHF
- Core idea:
  - Update model policy but constrain changes within a “trust region”
  - Avoids large, destabilizing updates
- Loss function includes:
  - Reward signal (from Reward Model)
  - KL penalty (keeps model close to supervised fine-tuned baseline)
- Widely adopted (used in OpenAI ChatGPT, Anthropic Claude, Google Gemini)

PPO in RLHF


KL div
SFT Model
Reward Model
GAE
Advantage Function
TD Error
Return
Policy LM
PPO-clip Loss
LM Loss
Pretraining Data
Divide
Value Model
Policy LM
Experience Buffer
Value Model
MSE Loss
User Query

DPO – Direct Preference Optimization

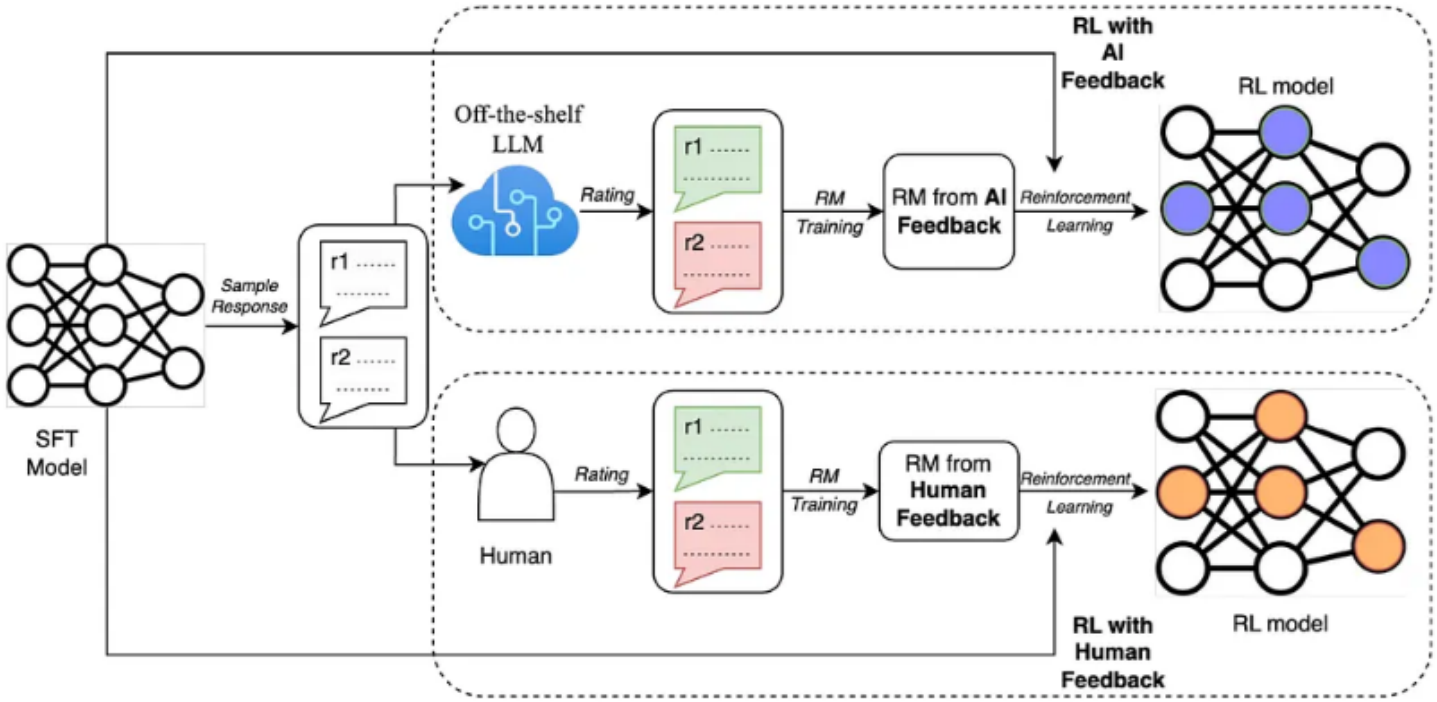
- Alternative to PPO-based RLHF
- **No explicit reward model, no RL loop**
- Directly optimizes LLM outputs against human preference data
- Uses pairwise comparisons: "preferred vs rejected" responses
- Advantages:
  - Simpler training (no reward model)
  - More stable, less computationally expensive
- Limitations:
  - Still requires high-quality preference datasets



Model Alignment by Reinforcement Learning – RLAIF (AI Feedback)

What is RLAIF? – The Concept

- Reinforcement Learning with AI Feedback
- Similar to RLHF, but replaces human annotators with AI models
- Stronger AI acts as a “teacher” providing preference judgments
- Goal: scalable and cost-effective model alignment

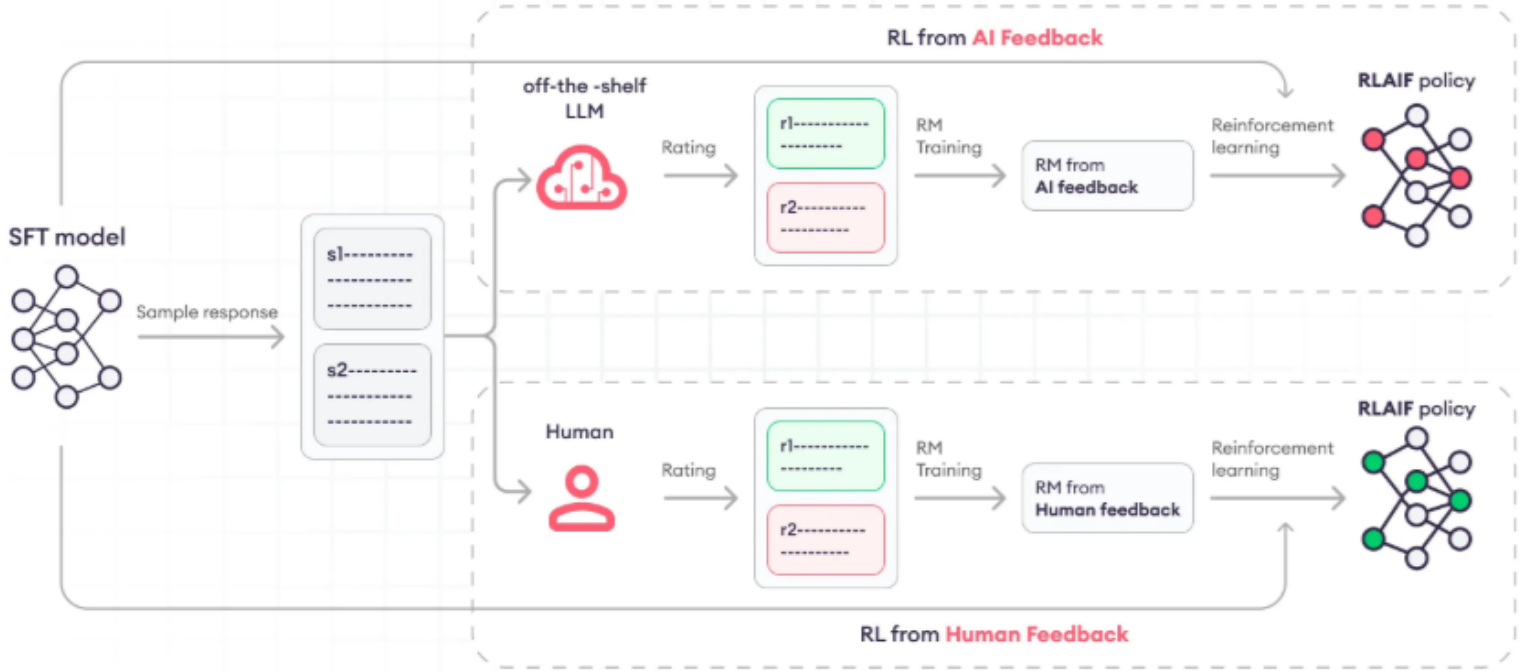


Motivation: Why RLAIF?

- RLHF challenges:
  - Human labeling = expensive, time-consuming
  - Difficult to scale for trillions of tokens
- RLAIF advantages:
  - Faster data generation
  - Lower cost than human feedback
  - Enables continuous, iterative alignment

RLAIF Pipeline – How It Works

- Collect responses from the target LLM
- Use a stronger AI model to rank or score responses
- Generate preference pairs (better vs worse answers)
- Train reward model or apply direct optimization (DPO-like)
- Update the target LLM with RL or preference optimization



Benefits of RLAIF

- Scalability: rapid collection of large feedback datasets
- Efficiency: reduce dependence on costly human annotators
- Consistency: AI feedback less noisy than multiple human raters
- Practicality: supports frequent model updates in production

Aspect	RLHF Benefits	RLAIF Benefits
Performance	Effective in engaging in human-like interactions. Adapts outputs to align with human preferences, applicable in conversation and summarization. It can produce harmful responses occasionally due to complex human desirability.	Comparable or superior to RLHF in tasks like summarization and harmless dialogue generation. Maintains helpfulness while making improvements in harmlessness and ethical alignment.
Subjectivity	Feedback is inherently subjective, reflecting biases and values of the human group providing feedback. This can lead to inconsistencies and may not represent diverse human values.	Reduces subjectivity by using AI-generated feedback guided by a constitution stating explicit principles. Provides a more objective and consistent feedback form grounded in defined ethical and safety standards.
Scalability	Collecting high-quality human preference labels is labor-intensive, time-consuming, and expensive, posing scalability challenges, especially for large and complex models.	Addresses scalability issues of RLHF by automating feedback with AI, allowing for the autonomous generation of large datasets. Reduces dependence on human labor, enhancing the scalability of the training process for large and complex models.

Challenges and Risks

- Feedback bias: supervising AI may propagate its own errors/biases
- Over-alignment: risk of “student” imitating teacher too closely → lack of diversity
- Quality control: hard to guarantee AI-generated preference labels
- Mitigation: combine AI feedback with smaller curated human datasets

Applications and Future Directions

- Industry adoption: DeepSeek, Google Gemini, Anthropic exploring RLAIF
- Hybrid approaches: AI feedback + selective human validation
- Future:
  - Chain-of-thought feedback from AI judges
  - Multi-agent feedback systems for robust evaluation
  - Towards sustainable, low-cost alignment at scale

Thank you!

Questions?