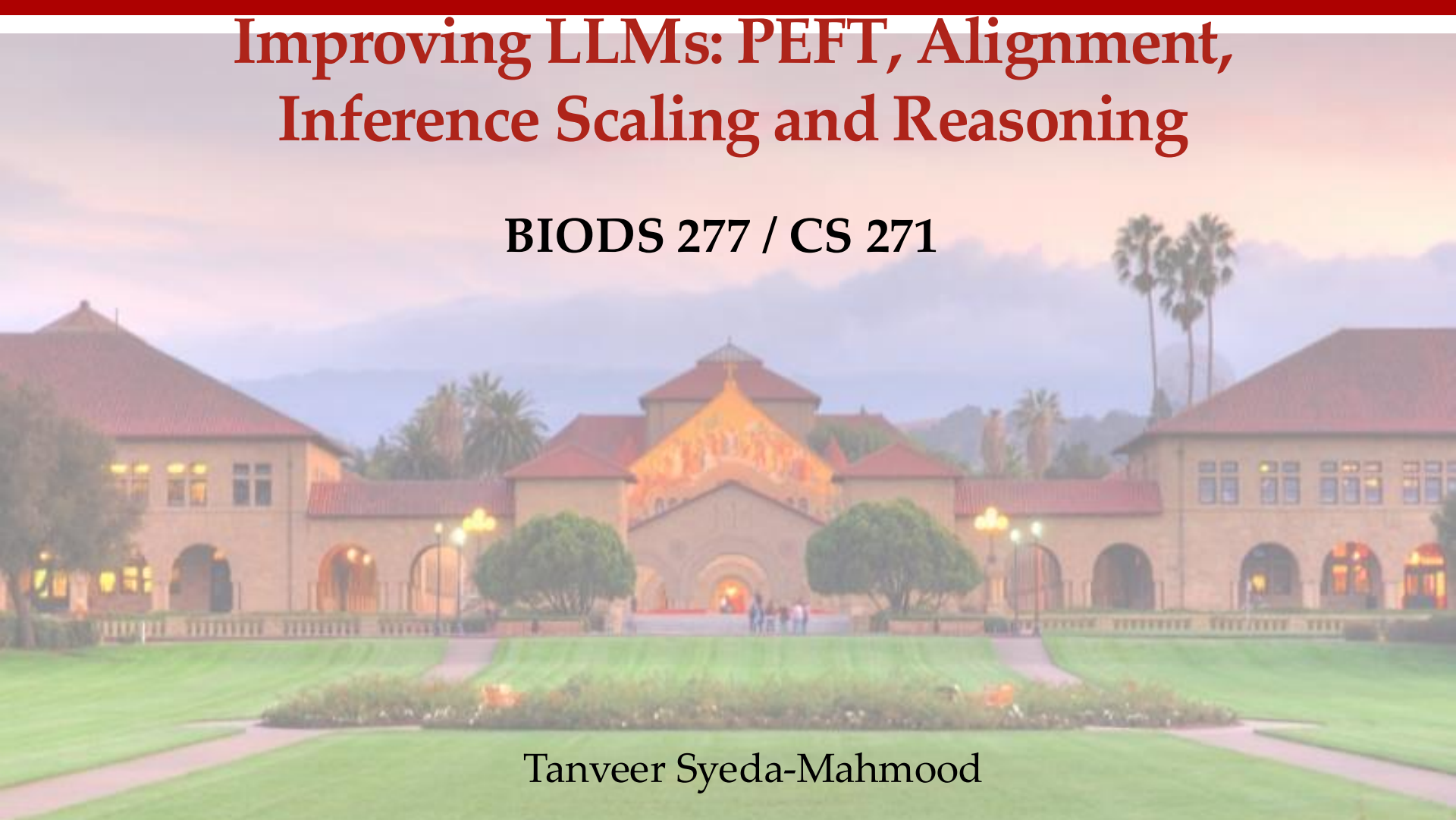
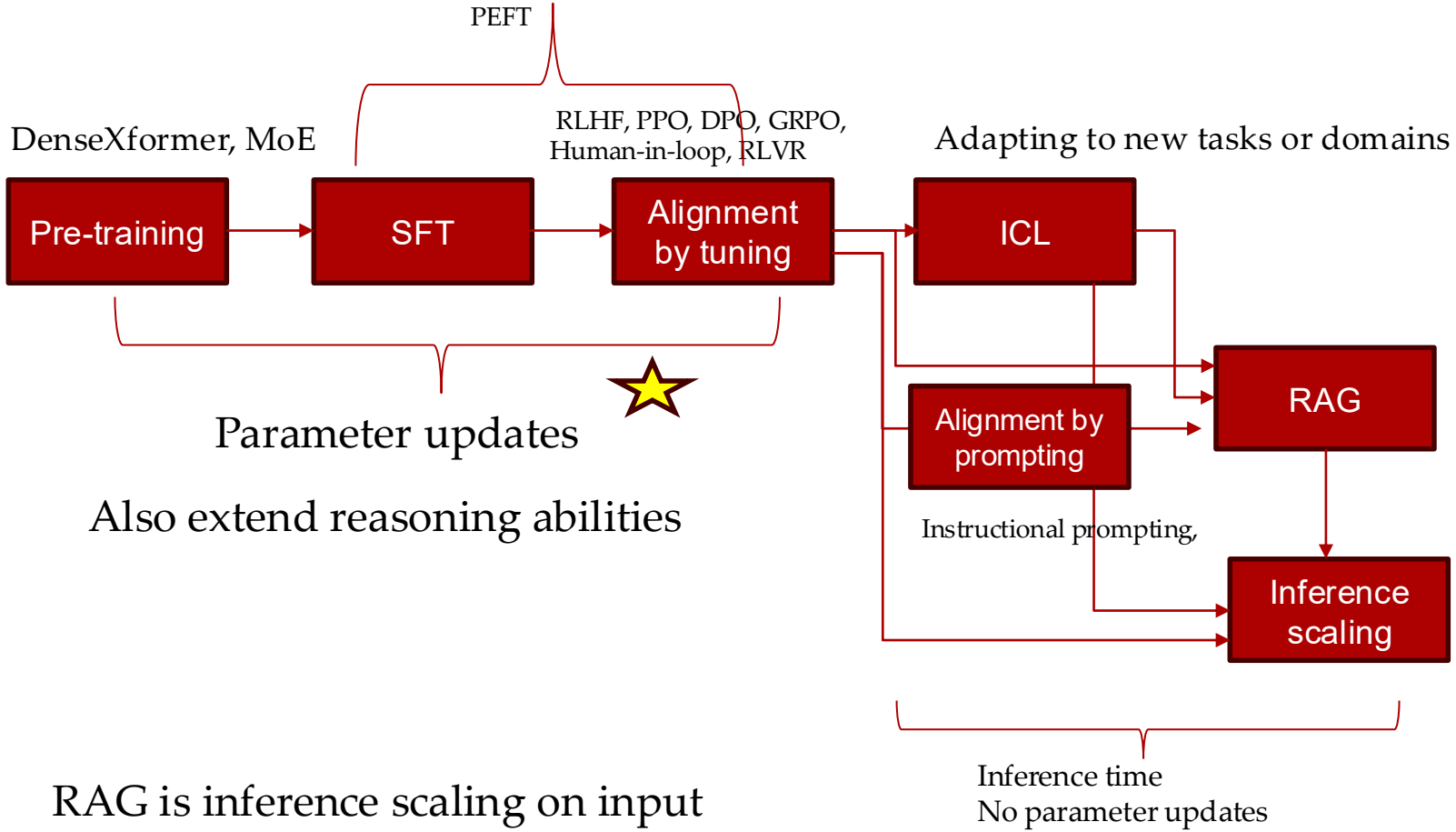
Improving LLMs: PEFT, Alignment, Inference Scaling and Reasoning

BIODS 277 / CS 271



Tanveer Syeda-Mahmood

LLM training and improvement



RAG is inference scaling on input

Today's topics

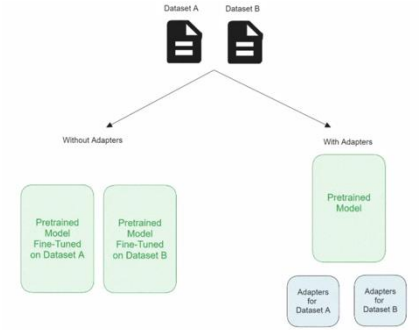
- LLM improvements that update parameters
  - During pre-training and SFT
  - During alignment
- Inference scaling and reasoning

Improving performance with parametric updates

- Methods differ mainly in **how much** of the model is updated and **how costly** the update is.
- PEFT
  - Adapters
  - LoRA
- MoE
- Alignment with parametric updates
  - RLHF, PPO, DPO, GRPO
- Synthetic data generation
- Instructlab

Parameter efficient fine tuning (PEFT)

- A class of methods that update only a fraction of the model's parameters instead of retraining the whole network?
  - we don't need to store several versions of fine-tuned models, but rather one pretrained model and several adapters.
  - Depending on the use case, we can then plug these adapters into the pretrained model during inference.
- Which parameters to select for updates?
  - Scaling vectors
  - Prompt tuning
  - Prefix tuning
  - Adapters
  - LoRA
- Reducing trainable parameters by **10×–10,000×**
- Lowering GPU memory and training time
- Enabling many task-specific versions of one base model



https://medium.com/@zilliz_learn/lora-explained-low-rank-adaptation-for-fine-tuning-llms-066c9bdd0b32

Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey

https://aiwithmike.substack.com/p/peft-parameter-efficient-fine-tuning?utm_campaign=post&utm_medium=web

Parameter selection for updates

- Is optimization based on loss function still happening? How does this affect backpropagation?

$$\nabla_\theta \mathcal{L} \longrightarrow \min_\theta \mathcal{L}(\theta) \longrightarrow \min_{\delta \in \mathcal{S}} \mathcal{L}(\theta_0 + \delta)$$

  - Backpropagation works as usual but on a different subspace
- Why does selective parameterization work?
  - LLMs are over-parameterized
  - Loss surfaces have wide flat valleys
  - Good solutions lie in low-dimensional manifolds
- Can the choice of parameters be arbitrary?
  - No, since gradients are **not isotropic**
  - Some parameters align with meaningful functional directions

Adapters

- Houlsby adapter
  - Places an auto-encoder-like structure inside the transformer layers
  - Similar performance to fine-tuning with two orders of magnitude fewer parameters
- How is this PEFT? Isn't this actually increasing parameters?
  - Yes, but you are only training a *small fraction* of parameters relative to the full model,
- Why add non-linearity?
  - Limited expressivity otherwise. Gating transformations, conditioning behavior is less
- Why place it there?
  - Gives maximal control over representations while preserving the transformer's stability
- Why place it twice in the architecture?
  - Attention routing and feature transformation can both be affected

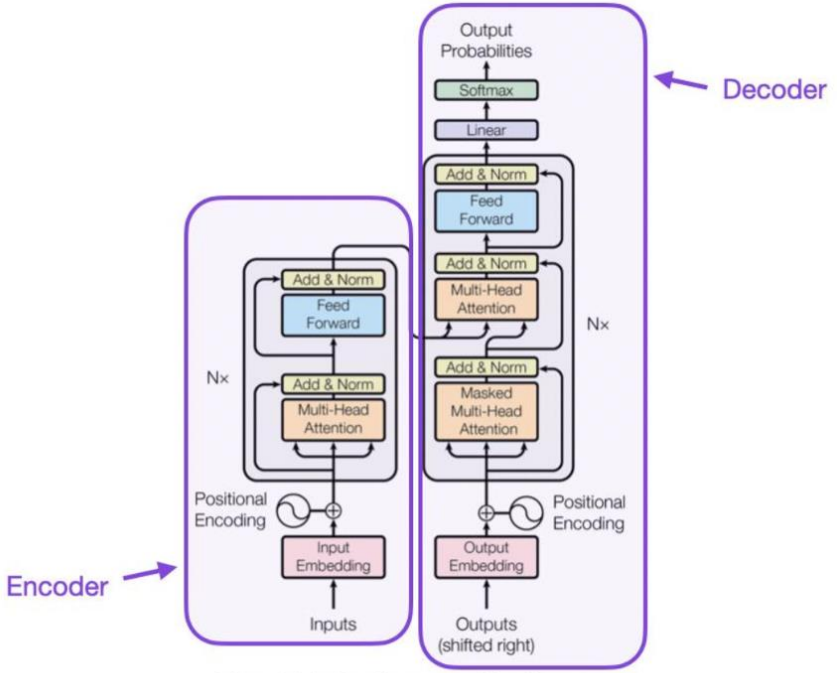
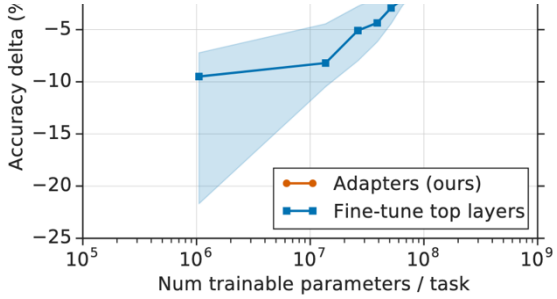


Figure 1: The Transformer - model architecture.



Parameter-efficient transfer learning for NLP (2019)

LoRA

- Task-specific adaptations live in a **low-dimensional subspace**
- Update matrix broken down into two low-ranked matrices
- Are small rank r updates enough when
  - Task is far from pre-training
  - Small models
  - Nonlinear transformations are needed

In normal fine-tuning, you update a weight matrix:

$$W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$$

LoRA says: don't update W.

Instead, learn:

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

- $A \in \mathbb{R}^{r \times d_{\text{in}}}$
- $B \in \mathbb{R}^{d_{\text{out}} \times r}$
- rank r ≤ rank(ΔW)

$$\Delta W = BA$$

$$W' = W + \alpha \frac{BA}{r}$$

$$\text{rank}(\Delta W) \leq \min(d_{\text{out}}, d_{\text{in}})$$

Even if the rank of ΔW is fixed, there are infinitely many choices of $B$and Athat produce the same ΔW.

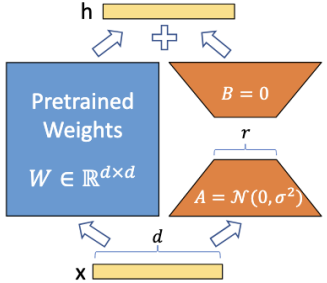
$$BA = (BP)(P^{-1}A)$$

So factorization is non-unique, but rank is unique

LoRA: Low-Rank Adaptation of Large Language Models (2021)

LoRA training and inference

- Training:
  - Initialize A, B
  - Forward pass:
  - Backpropagate:
  - Optionally scale:
- Inference:
  - B,A can be dropped after merging



$$y = W'x = (W + BA)x = Wx + B(Ax)$$

$$\frac{\partial \mathcal{L}}{\partial A} = B^\top \frac{\partial \mathcal{L}}{\partial (BAx)} x^\top \qquad \frac{\partial \mathcal{L}}{\partial B} = \frac{\partial \mathcal{L}}{\partial (BAx)} (Ax)^\top$$

$$W' = W + \alpha \frac{BA}{r}$$

$$W_{\text{merged}} = W + BA$$

Mixture of experts

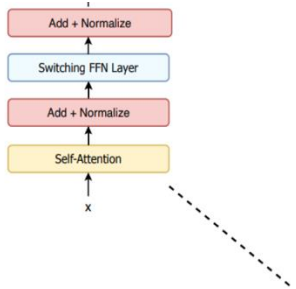
- Architectural and Training time design choice
- Many expert subnetworks
- A router/gating network selects a small subset of experts per token or per layer
- Only those experts are activated for that token
- Total parameters>>actual parameters
- Sparse activation
- Conditional computation
- Gives structural modality
  - Specialize for domain, modality, reasoning styles
  - Fine-tune for legal, healthcare, etc.

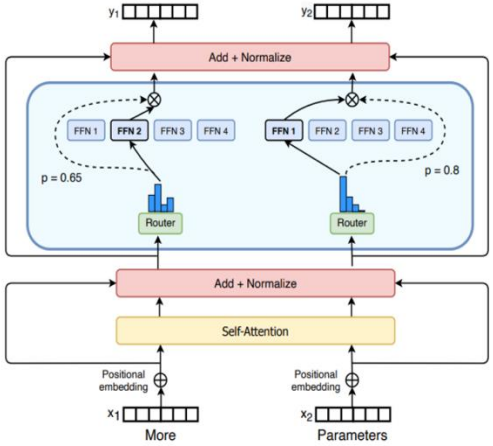
Mixture of Experts

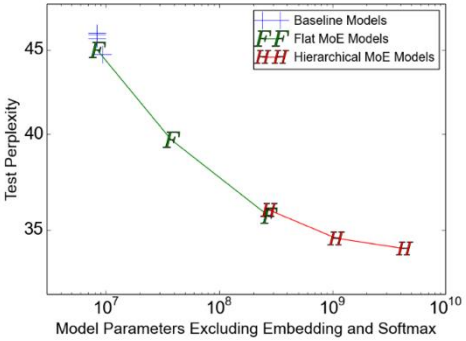
- A neural network with “expert” sub-networks
  - for each input, only a subset of experts is active using a gating function.
- Each expert network is a FFN or transformer layers
- Gating network is another neural network
  - Softmax + topK
- Router:
  - Maps inputs to experts dynamically by computing the raw scores
- Works well for multi-task and multi-domain learning

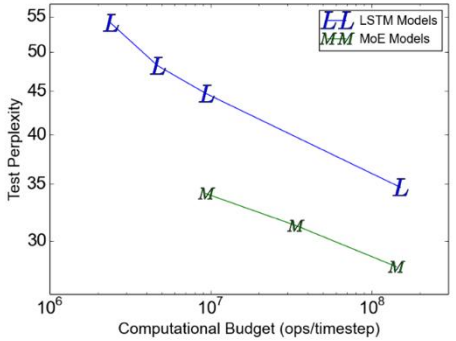
$$y = \sum_{i=1}^{N} g_i(x) E_i(x)$$

Switch transformer









A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory, and Applications

OUTRAGEOUSLY LARGE NEURAL NETWORKS: THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER(ICLR2017)

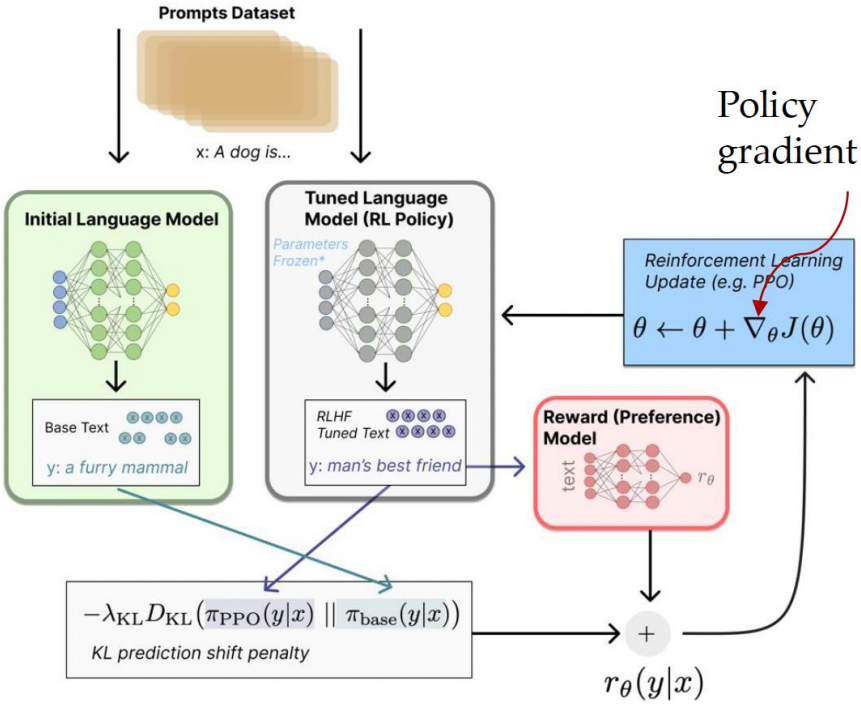
RLHF fine-tuning alignment

- Train an LLM using SFT
- Rate the output of the LLM on a preference dataset
- Train a reward or preference model using this supervision
- Use an RL algorithm to update the weights:
  - RL is the algorithm that takes the output of the reward model and uses it to update the LLM model weights so that the reward score increases over time.
    - Use PPO reinforcement learning to optimize the output of the LLM

RLHF Fine-tuning

KL-divergence between present and past responses

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}.$$

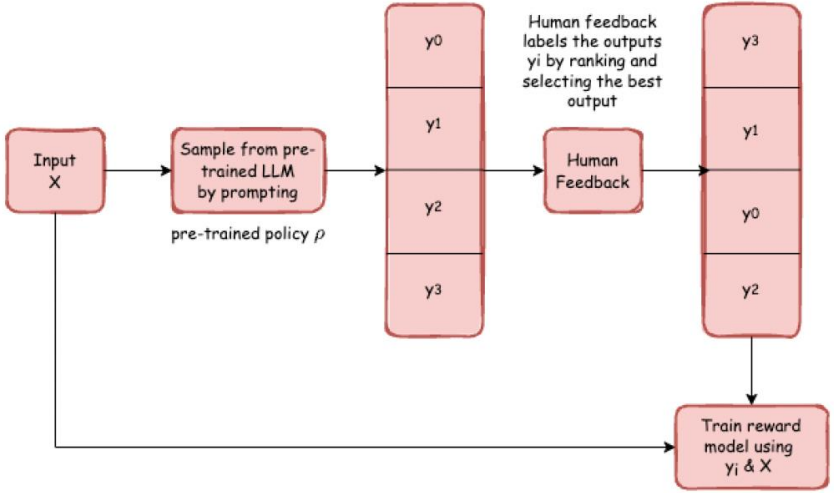


- SFT LLM learned using backprop
- Learn the reward model separately
- Update the LLM weights by the RL policy algorithm

https://www.ionio.ai/blog/a-comprehensive-guide-to-fine-tuning-llms-using-rlhf-part-1

Reward model

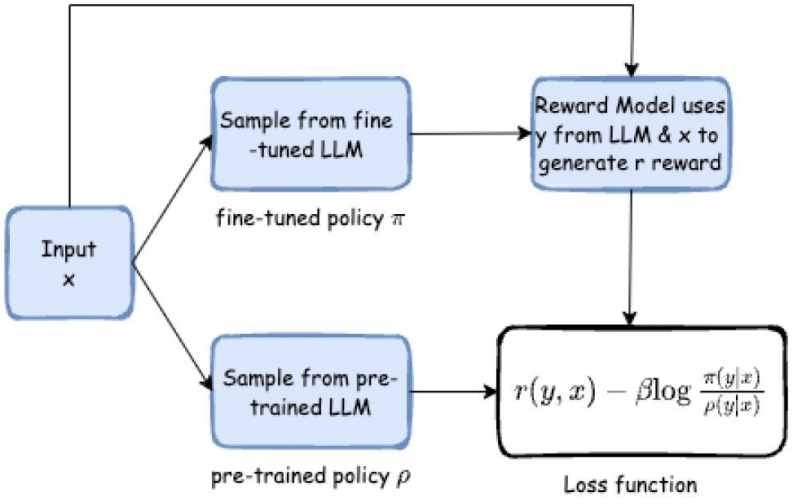
- Trained with a handful of examples and asking users for feedback.
- For each input xi, collect multiple outputs yi from LLM
- Humans rank the response as best to worst
- Train a rewards model:
  - Classifier model: takes xi,yi as input and produces a score as output that maximizes the probability of emitting yi given xi
  - Re-ranking model : maximizes the order of the ranking of the outputs
- Usually another transformer pre-trained with the SFT-trained LLM with a linear layer on top of final transformer layer.



$$\text{loss}(r) = \mathbb{E}_{(x,\{y_i\}_i,b)\sim S}\left[\log \frac{e^{r(x,y_b)}}{\sum_i e^{r(x,y_i)}}\right]$$

Logistic function

RL-fine-tuning : Training policy loss function



$$R(x,y) = r(x,y) - \beta \log \frac{\pi(y|x)}{\rho(y|x)}.$$

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \left[r_\phi(x,y)\right] - \beta \mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x)\right],$$

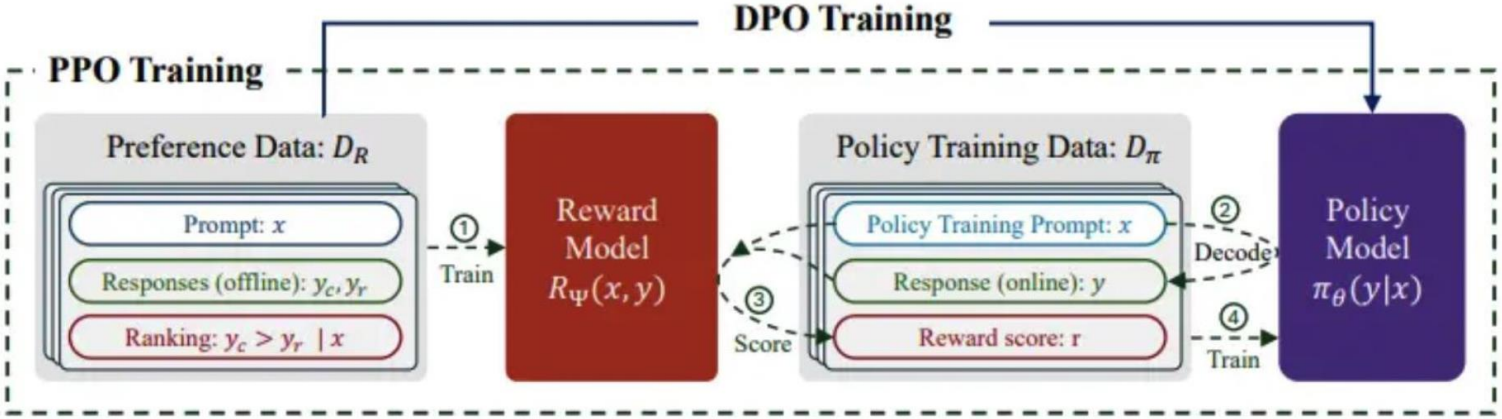
Maximize using PPO

$$r(x,y) = r_\phi(x,y) - \beta(\log \pi_\theta(y \mid x) - \log \pi_{\mathrm{ref}}(y \mid x)),$$

Due to the discrete nature of language generation, this objective is not differentiable and is typically optimized with reinforcement learning using PPO.

Direct policy optimization (DPO)

- No separate reward model needed
- Implicit reward included in the policy update rule.
  - Analytical mapping derived from reward functions to optimal policies, which transform a loss function over reward functions into a loss function over policies
- Human feedback still taken for preferences after SFT of the LLM
- The LLM model is optimized without using RL by minimizing the L_{DPO} loss.



Direct Preference Optimization: Your Language Model is Secretly a Reward Model, NeurIPS'23

Direct preference optimization

- Not an RL method

- a **frozen reference policy** π_{ref}
- a **trainable policy** π_θ

Human-curated preference training data

$$(x, y^+, y^-)$$

Maximize

$$\log \sigma\Big(\beta\big[\log \pi_\theta(y^+|x) - \log \pi_\theta(y^-|x) - (\log \pi_{\text{ref}}(y^+|x) - \log \pi_{\text{ref}}(y^-|x))\big]\Big)$$

- Increase probability of preferred answer
- Decrease probability of rejected answer
- Relative to the reference policy
- offline, supervised optimization on preference pairs.
- Still needs preference pairs to be assembled

GRPO

- *What if instead of relying on external feedback, we could find an automatic way to validate and rate model responses?*
- An RL method, doesn't require labeled data
- **making small, controlled updates using a group of observations**.
- Apply relative rewards within a group
- Add a reinforcement learning style policy gradient
- To ensure that the updated policy $\pi\theta$ does not deviate too much from the old policy, GRPO includes a **KL divergence constraint**:

$$\mathcal{L}(\theta) = -\mathbb{E}_x \left[\frac{1}{K} \sum_{i=1}^{K} A_i \log \pi_\theta(y_i \mid x) \right] + \beta \, \mathbb{E}_{x,y} \left[\log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} \right]$$

Multiple samples per prompt

$$\{y_1, \ldots, y_K\} \sim \pi_\theta$$

$$A_i = R_i - \frac{1}{K} \sum_{j=1}^{K} R_j$$

$$\nabla_\theta \mathbb{E}\left[r_i \cdot \log \pi_\theta(y_i|x) \right]$$

Reward functions:

- Automatically computable
- Task-grounded
- Comparable across samples
- Often relative, not absolute

e.g. Unit tests passed

Even weak heuristics seem to work

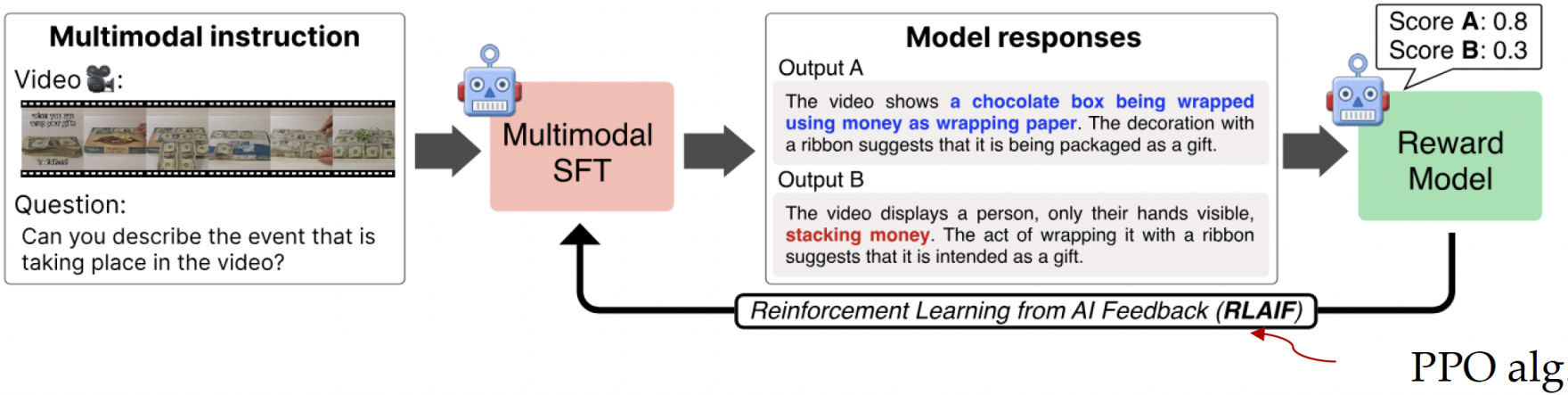
DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models.

Comparison of parametric approaches

Method	Parameters Updated	Cost	Flexibility
Full fine-tuning	All	Very high	Very high
Continued pretraining	All	Very high	Medium
SFT	All	High	High
RLHF	All	Very high	High
PEFT (LoRA, adapters)	Few	Low	Medium
Prompt tuning	Very few	Very low	Low–medium

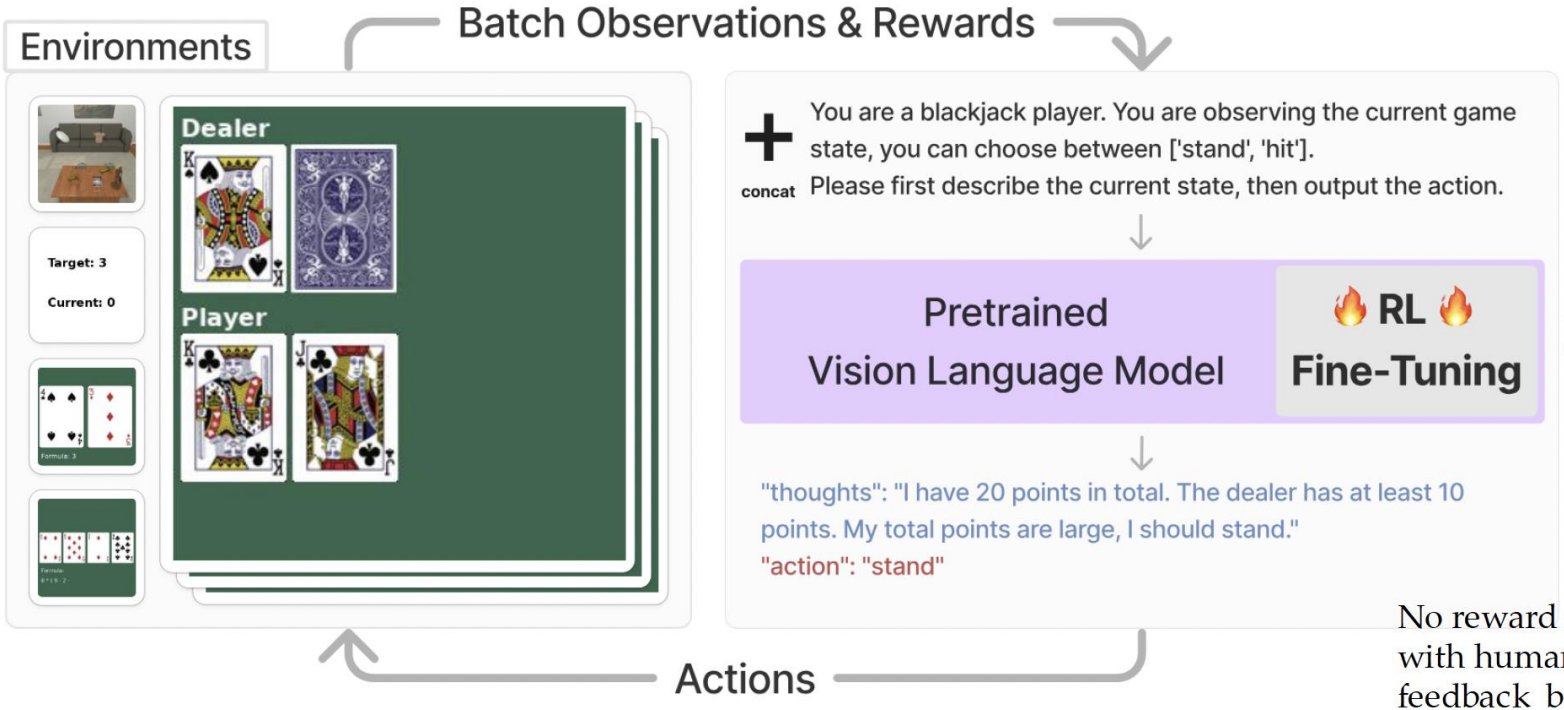
Improving VLM models using RLHF

Uses both reward models and PPO RL optimization



Tuning Large Multimodal Models for Videos using Reinforcement Learning from {AI} Feedback. ACL'2024

Improving VLM models using RL



PPO al

Fine-Tuning Large Vision-Language Models as Decision-Making Agents via Reinforcement Learning, NeurIPS'2024

No reward model with human feedback but custom reward function

LAB - Large-scale alignment

- Can we align chatbots using mostly models, tools, and rules instead of people?
- Teach the model *what aligned behavior looks like*
- Teach the model *why some behaviors are better*

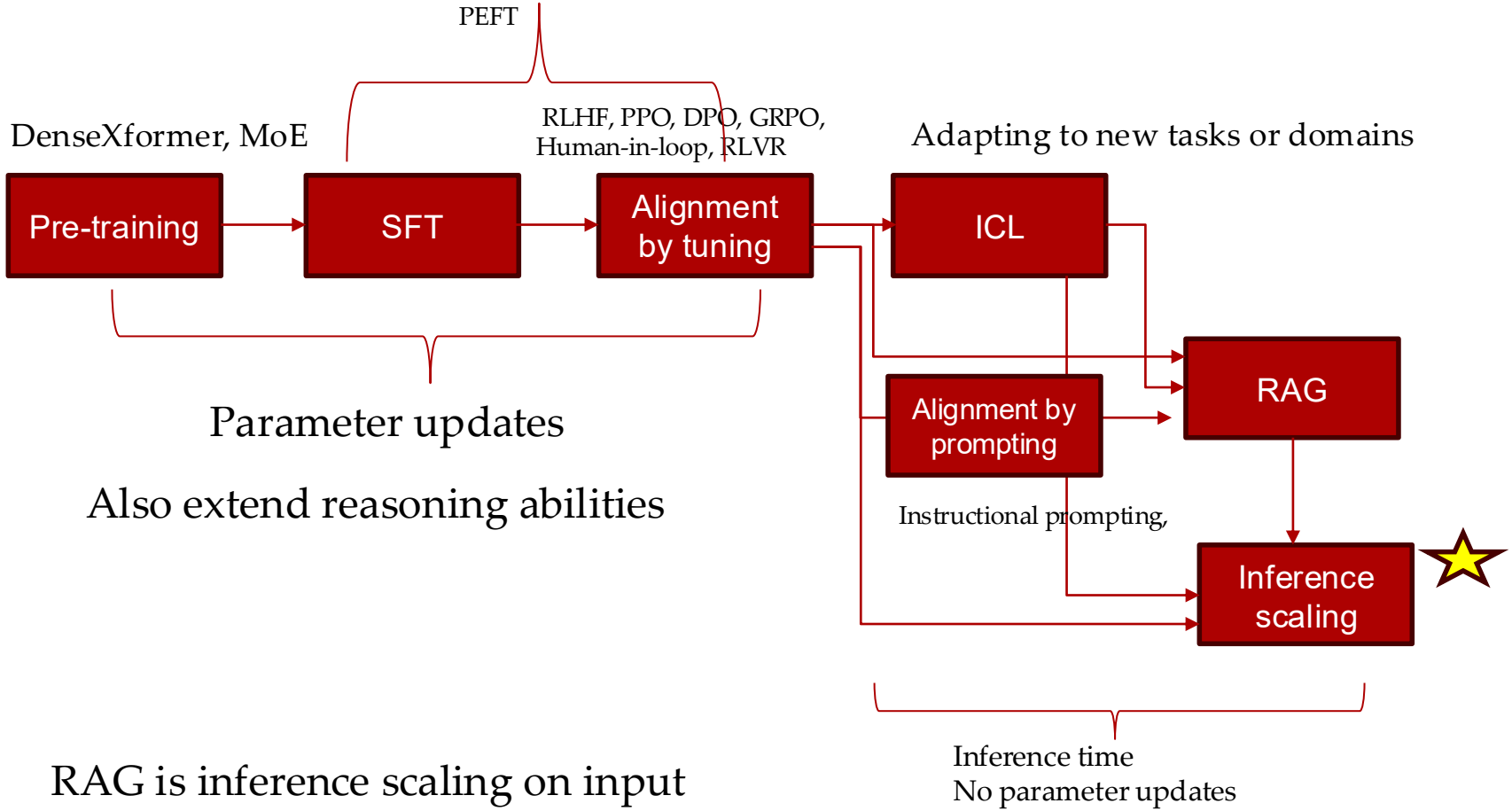
  → via AI-generated critiques and rules

LAB: Large-Scale Alignment for ChatBots

LAB method

- Use humans per subject category to generate examples
- Use a synthetic data generation methodology to produce many more samples
- Filter the samples using a critique model (LLM-as-a judge) and rank the responses
- Fine-tune the current model with the synthetically produced data
  - DPO, PPO, etc. can still be used.
- LAB is **alignment via automation**.
- LAB challenges:
  - Subtle human values
  - Ambiguous ethical tradeoffs
  - Cultural nuance
  - Long-term deception risks

LLM training and improvement



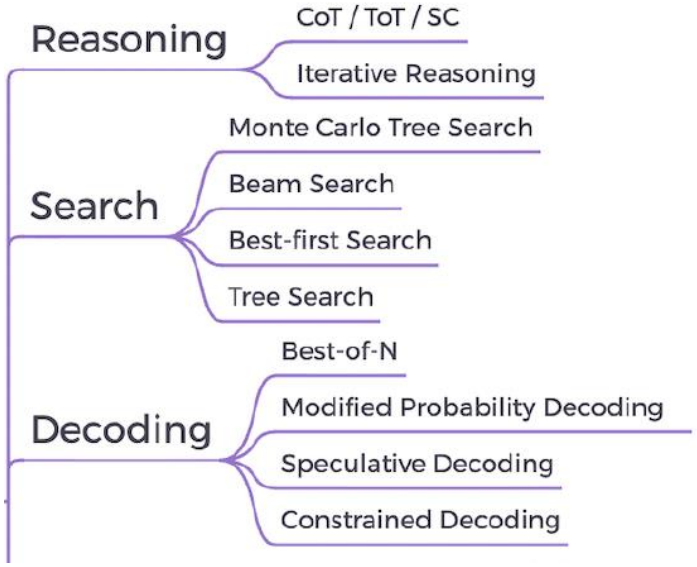
RAG is inference scaling on input

Inference scaling

- Purpose: improves accuracy, improves reasoning
- Inference scaling
  - Increase inference time compute without increasing model size
  - *Leads to to higher accuracy, improved reasoning, and greater adaptability without costly retraining*
- Inference scaling usually refers to spending **more compute at test time** *without changing the model weights (no training or model updates happening)*
- Because inference scaling introduces **iteration + selection**, it mimics some effects of learning.
- It is a temporary change of model performance:
- Involves generating multiple candidate solutions, evaluation, and selection (multiple rounds of such iteration)
- Can be done by changes at the input or output of LLM
  - Changes at the input end already studied under RAG

Inference time scaling methods

Inference scaling

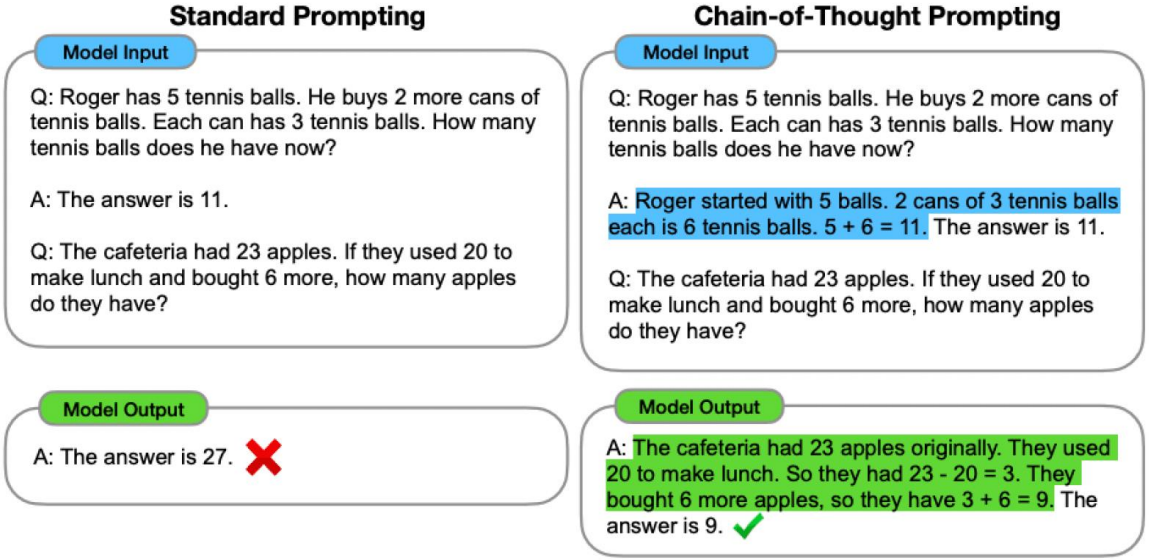


Reasoning methods

- COT prompting
  - COT, TOT, GOT
- Iterative reasoning
  - ReAct, *Self-Refine and Self-Debug*

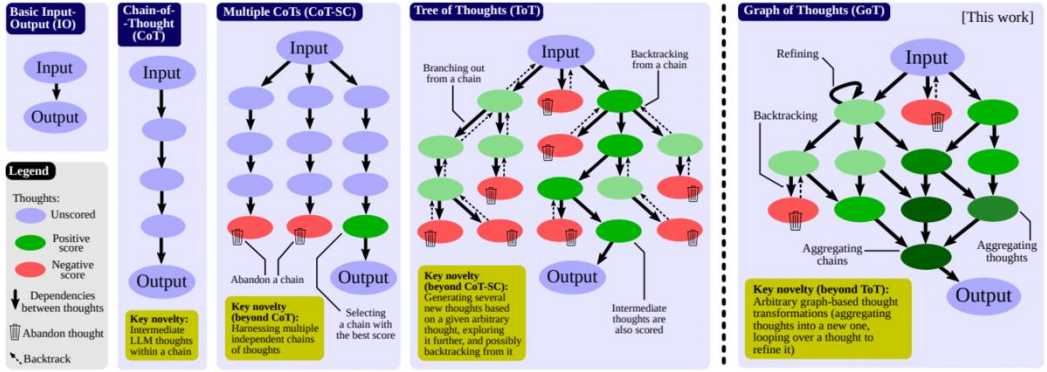
Chain-of-thought prompting

- Tackle address complex problems by breaking them into simpler, intermediate steps and encouraging the model to "think" before answering



Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

C.O.T. prompting variants



"Let's think step by step" works because it is:

1. **Common** in training data
2. **Consistently followed** by reasoning
3. **Rarely followed** by short answers

So phrases like:

- "Let's break down the reasoning"
- "Let's work through this carefully"
- "Step-by-step analysis"
- "Reason it out in detail"

all belong to the same **statistical equivalence class**.

In SC:

- *Paths Generation: produces multiple paths using (temperature) sampling.*
- *Majority Voting: selects the final answer by aggregating the most frequent outcome.*

Two factors dominate:

(a) Frequency

How often did the phrase appear in training?

(b) Purity of continuation

When it appeared, was it *consistently* followed by reasoning?

"Let's think step by step" scores high on both:

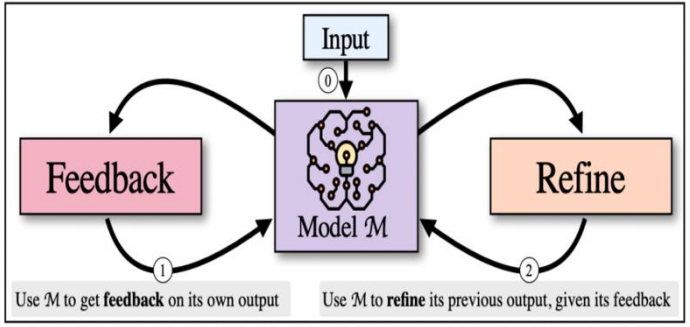
- Very common
- Almost always followed by explicit reasoning

Iterative methods of reasoning

- *Have repeated cycles of reasoning, action, and refinement*
- *Address reasoning, decision-making, and long-term planning*
  - *ReAct, Self-Refine and Self-Debug – critique their own output*
  - *Increase LLM's ability to generalize, reduce errors, and produce reliable, interpretable results across diverse applications.*

Reasoning methods

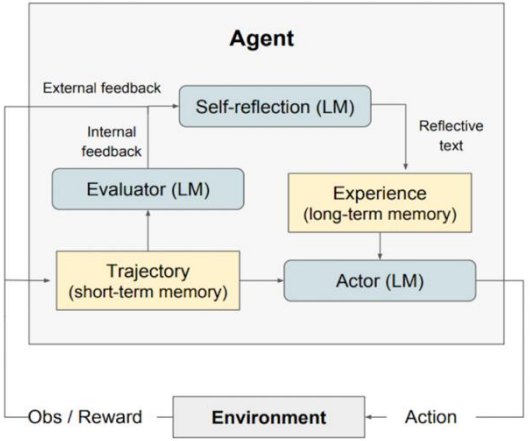
- *Reasoning with Task decomposition*
  - *decompose and solve: Decompose the query into sub-queries and solve subqueries*
- *Reasoning with feedback*
  - *Reasoning + action + observation (ReAct): Generate thoughts, consult sources (e.g. Wikipedia), feedback from environment dictates next actions*
  - *Better than COT producing grounded, fact-driven reasoning traces*
  - A new prompt that includes the previous history + the new observation
- *Iterative Reasoning with Memory*
  - *Use a memory in the loop of the refinement to supply the past episodes*
- *Iterative reasoning with planning*
  - *Uses a planner to decompose the initial task into a plan, convert to NL, and use it as a prompt with COT*
- *Iterative reasoning with selection*
  - *Use a selection mechanism to prune alternatives*
- *Iterative reasoning with a tool*
  - *Convert the interim COT into variables and express the operation as an executable code*



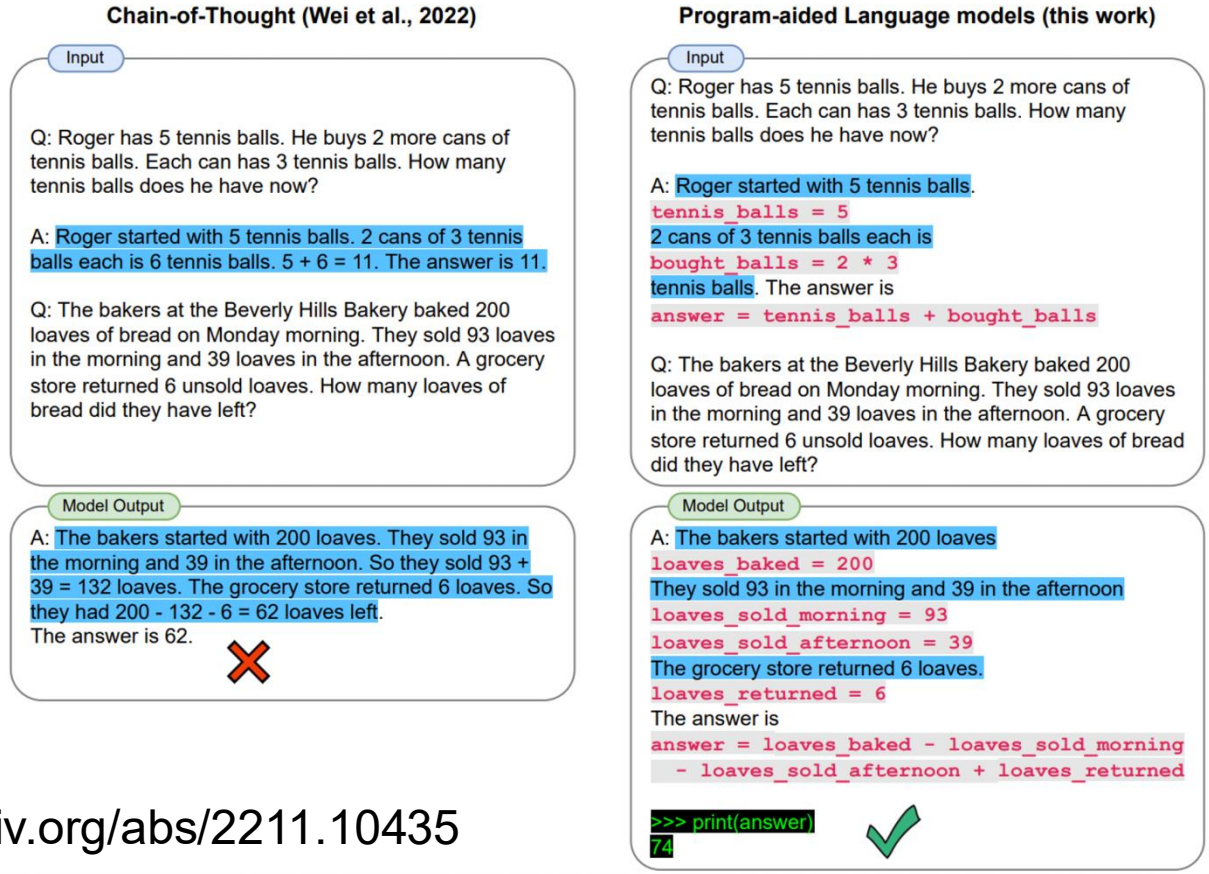
Prompt 1:
Question: Who is the CEO of Company X?
Thought: I should search for Company X.
Action: Search("Company X CEO")

Observation: The CEO is Alice Smith. (this is the feedback)

Prompt 2:
Thought: Now I know the CEO.
Answer: Alice Smith



PAL: program-aided LLM



https://arxiv.org/abs/2211.10435

Search methods of inference scaling

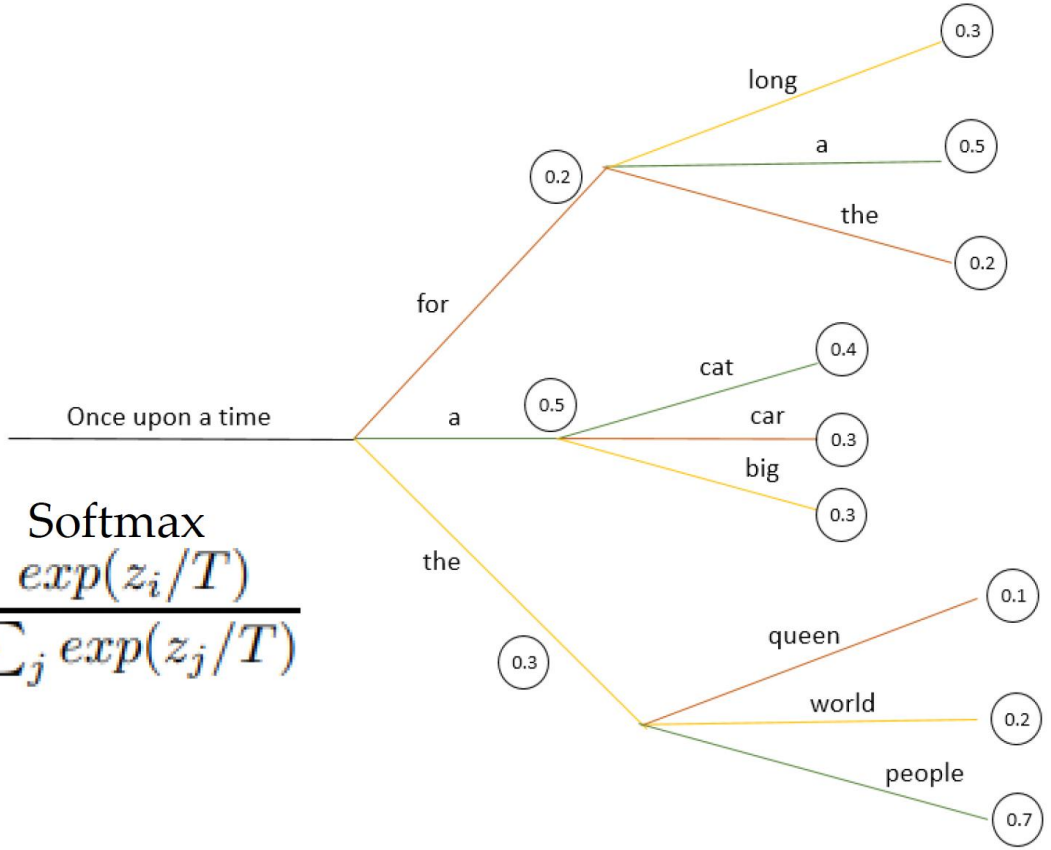
- *Search strategies used to guide LLMs in reasoning and sequence generation*
- *MCTS (Monte Carlo Tree Search)*
- *Beam search*
- *Stochastic beam search*

Understanding the LLM text generation process

- Transformer decoder
- Next token selector
  - Beam search
    - Deterministic
    - Stochastic
      - Random
      - Temperature-based
      - Top-k
      - Top-p

Softmax

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

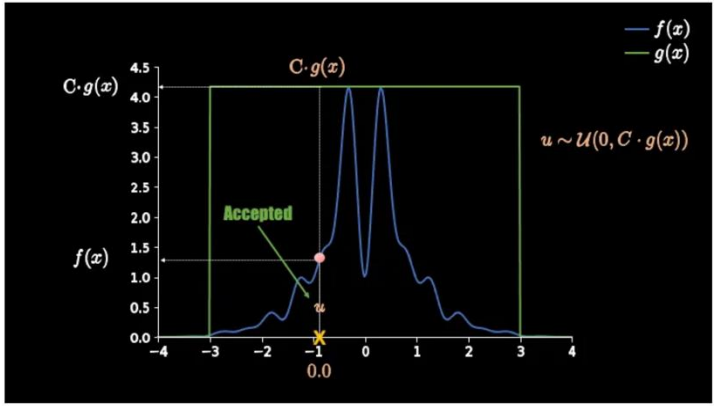


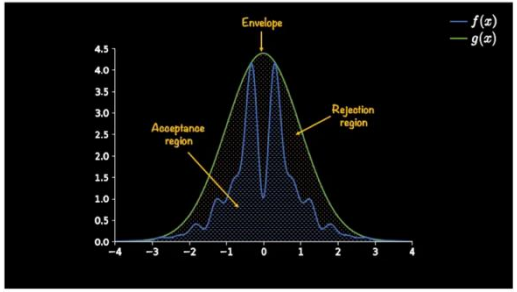
A Thorough Examination of Decoding Methods in the Era of LLMs

Search methods

- Monte-Carlo sampling
  - *Selection:*
    - *traverses the search tree by selecting child nodes using a policy that balances exploitation and exploration*
  - *Expand the node into child nodes*
  - *Rollout to estimate value*
  - *Backpropagate up the search tree*
- repeatedly sampling outputs from the model's probability distribution to explore the space of possible reasoning paths.

- get another sample from the proposal function
- get a random number (u) between 0 & Cg(x) from a uniform distribution function
- check if u ≤ f(x); if true accept it else reject it
- repeat





Monte Carlo sampling for search

- Sampling from the probability distribution
  - Good continuations are sampled more often
  - Bad ones are sampled rarely

Example: Best-of-N

1. Proposal: π_θ
   - Sample N sequences $y_1, \ldots, y_N \sim \pi_\theta(\cdot|x)$
2. Evaluate: $R(y_i)$
3. Select: $y^* = \arg\max_i R(y_i)$

Here:

- Proposal = π_θ
- Target distribution = implicitly $p^*(y) \propto \pi_\theta(y) e^{R(y)}$

Decoding methods

- Decoding: Given the distribution, what token comes next?”
- Standard decoding: p(t|context)
  - Greedy argmax, sampling, beam search
- Best of N, Majority voting
- Adjusted probability decoding: p′(t|c)=Adjust(p(t|c),constraints,objectives)

$$P_{\text{adjusted}}(y|x) = \alpha \times P(y|x) + (1 - \alpha) \times P_{\text{KNN}}(y|x)$$

$$p'(t) \propto p(t)^{1/T}$$

Temperature scaling

- Constrained decoding: adhering to syntactic, semantic logical rules (e..g JSON output)

How to adapt frontier model for a task?

- **If the task is *similar* to existing capabilities**
  - **ICL**
    - Task is small
    - Needs flexibility
  - **Prompt engineering**
    - You need consistent behavior
  - **Inference scaling**
    - Task requires multi-step reasoning
    - Errors come from shallow thinking
    - This improves reliability, not capability.
- If the task is *new but learnable from examples*
  - **SFT**
    - You can generate high-quality (x, y) pairs
    - You want persistent capability
- If the task requires *optimization, preferences, or tradeoffs*
  - **Best method: Preference learning (RLHF / DPO / RLAIF)**
- If the task is *fundamentally novel or out-of-distribution*
  - **Pre-training**
- If the task must adapt online or per-user
  - **In-context learning**
  - External memory
  - Retrieval
  - Adapters / LoRA (sandboxed)
  - Avoid full weight updates unless you can isolate them.

Practical decision table

Task type	Best method
Small / temporary	In-context learning
New skill, clear labels	SFT
Judgment / preferences	RLHF / DPO
Hard reasoning	Inference scaling
Fundamentally new domain	Pretraining + tools
Per-user adaptation	ICL + memory

LLM for enterprise data

- Enterprise tasks typically have these properties:
  - Highly proprietary (not in pretraining)
  - Small but critical datasets
  - Strict correctness requirements
  - Long-tail schemas and conventions
  - Evolving rules
  - Auditing and traceability needs
- LLMs, even frontier ones, are optimized for:
  - Broad generalization
  - Average-case performance
  - Natural language fluency
  - Probabilistic correctness
- Advanced architectures are coming

New emerging architectural patterns

1 LLM + structured memory

- Databases
- Knowledge graphs
- Vector stores
- Versioned facts

The model *queries*, not memorizes.

2 LLM + symbolic systems

- Rules engines
- Constraint solvers
- SMT / SAT
- Domain-specific languages

The LLM:

- Translates intent → formal representation
- Delegates correctness to symbolic components

3 LLM + tool orchestration

- APIs
- Internal services
- Code execution
- Workflows

The LLM is the **planner**, not the executor.

4 LLM + verification layers

- Static checkers
- Type systems
- Schema validation
- Post-hoc validators

Failures are caught *before* output is accepted.