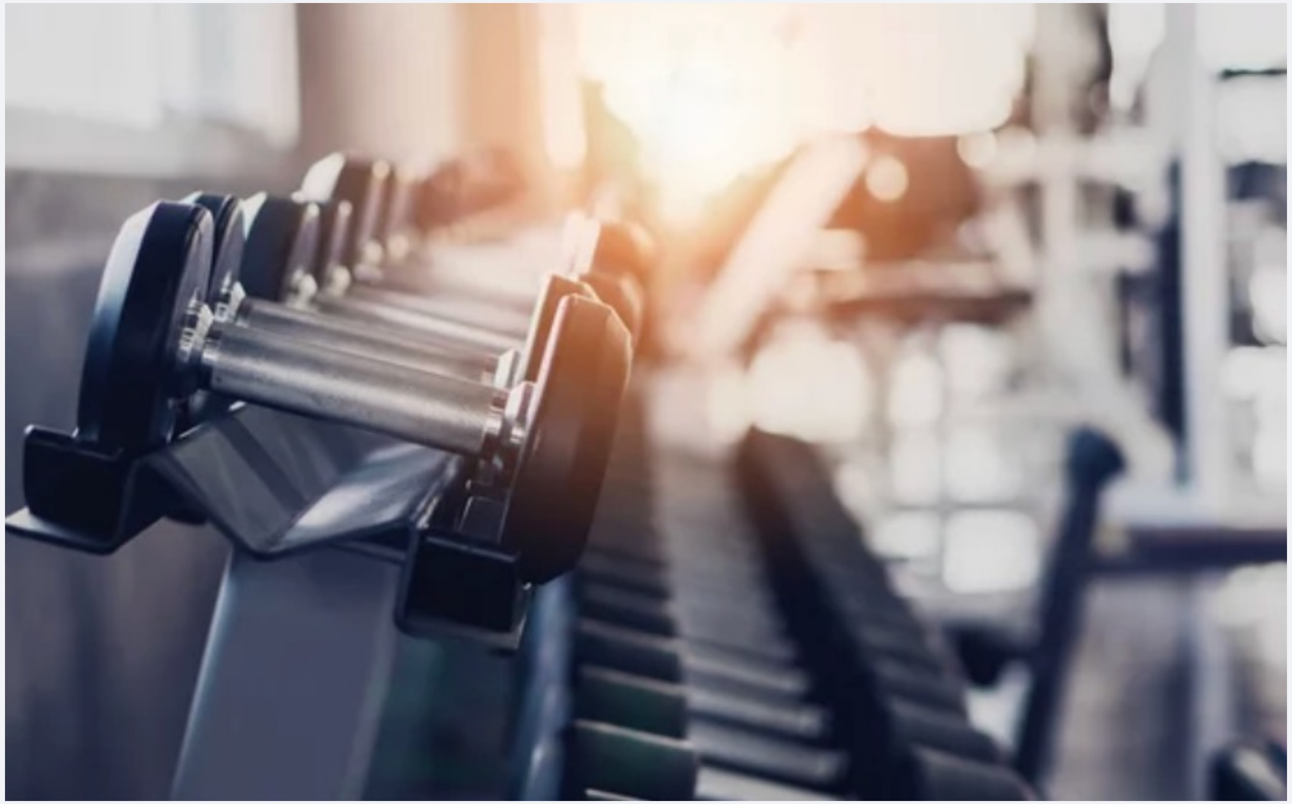


Post training

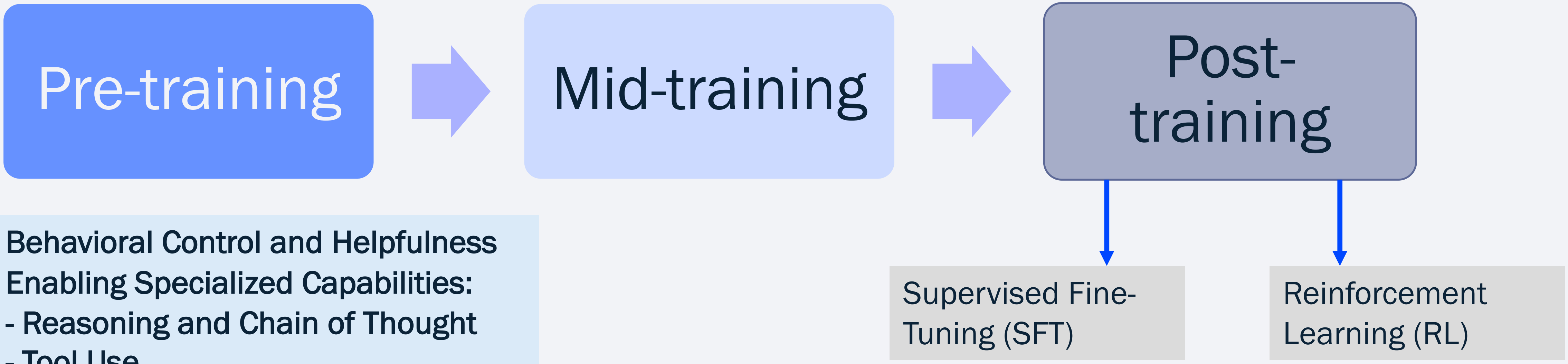
Dr Maria Prokofieva

Maria Aise

What we will cover

- Stages in model lifecycle
- Supervised Fine-Tuning
- Reinforcement Learning

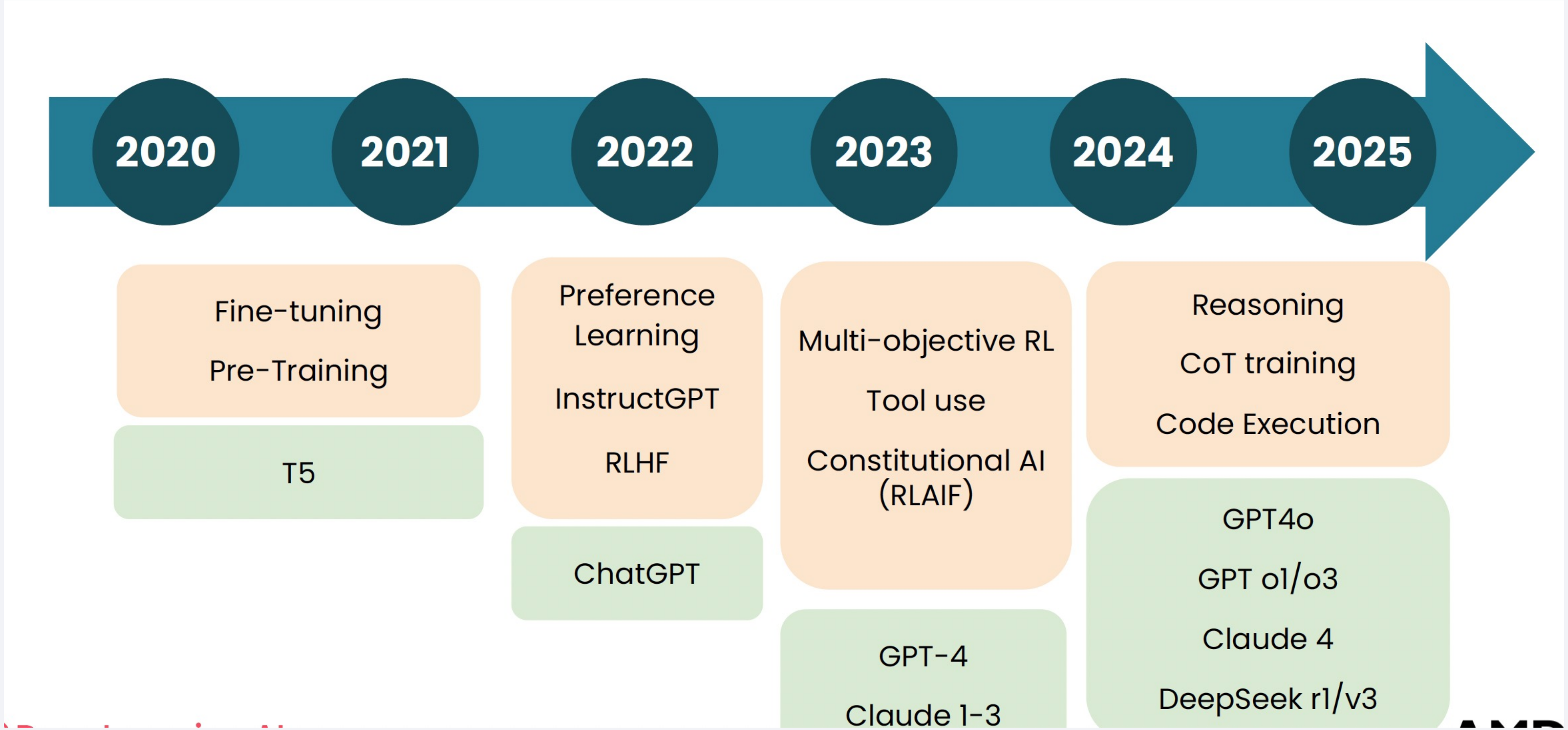
Post-training



Behavioral Control and Helpfulness
Enabling Specialized Capabilities:
- Reasoning and Chain of Thought
- Tool Use
- Coding and Debugging

Safety and Guardrails
Handling Conversational Nuance
Domain Expertise

Evolution of post training



Why post-training

- Question
- Dialogue
- Topic changes
- Interruptions
- Avoid harmfulness
- Bias
- Toxicity
- Helpfulness
- Consistency
- Prompt styles
- Debugging
- Handle ambiguity
- Coding
- Handle typos
- Problem solving
- Reasoning
- Domain expertise
- Creative writing
- and more!

Before vs after fine-tuning: text generation

- It continues literally rather than treating the prompt as an instruction.

Prompt: Write a short story about a cat that discovers time travel.

Output: The cat that discovers time travel. The cat that discovers time travel was sitting on the windowsill looking out at the birds. It was a very nice day. There were clouds. The cat thought about mice. Then the cat saw a strange light…

- The story continues coherently, ends properly, stays on topic

Prompt: Write a short story about a cat that discovers time travel.

Output: Once upon a time, in a quiet suburban home, lived a curious tabby named Whiskers. One rainy afternoon, while chasing a sunbeam that shouldn't have been moving, Whiskers pounced into an old grandfather clock. A flash of light enveloped him—and suddenly he was in ancient Egypt, surrounded by scarabs and pharaohs…

base Llama-3-8B or similar

Before vs after fine-tuning: classification

Before

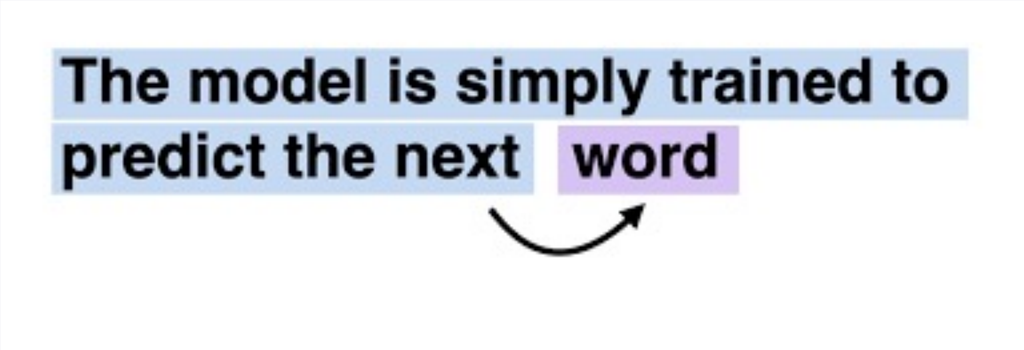
- Not designed for classification at all → no classification head, no concept of "labels".
- Can only be used indirectly via **prompting** (zero-shot / few-shot)
- **Performance:** Surprisingly decent on simple tasks (especially 2024–2026 frontier models) due to emergent classification ability from pre-training.
- But inconsistent formatting, hallucinations, verbosity, sensitivity to prompt wording.
- Accuracy usually **lower** than fine-tuned small encoders (BERT, DeBERTa) on standard benchmarks.

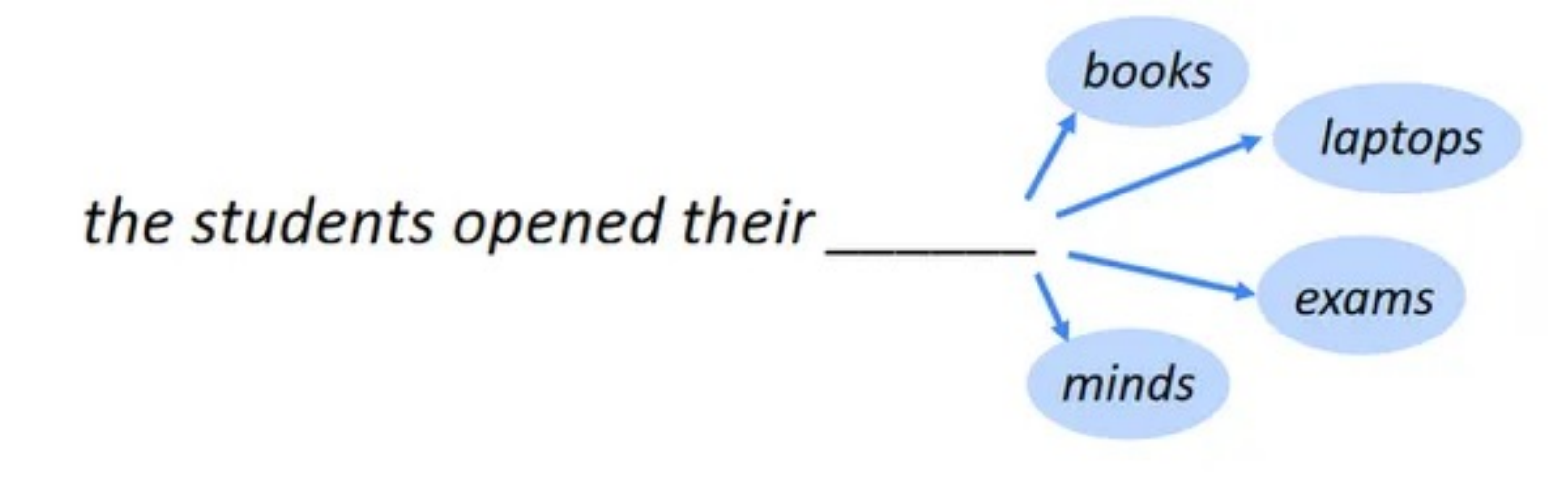
After

- Usually adds a **classification head** (linear layer) on top of the last hidden state (or pooled output).
- Trained supervised on labeled examples → directly optimizes cross-entropy loss for the label set.
- **Output:** Raw logits or probabilities over classes → clean, no parsing needed.
- **Performance:** Significantly higher accuracy, especially on domain-specific or nuanced data.
- Much faster inference (no long generation needed).
- Far more robust to prompt sensitivity.

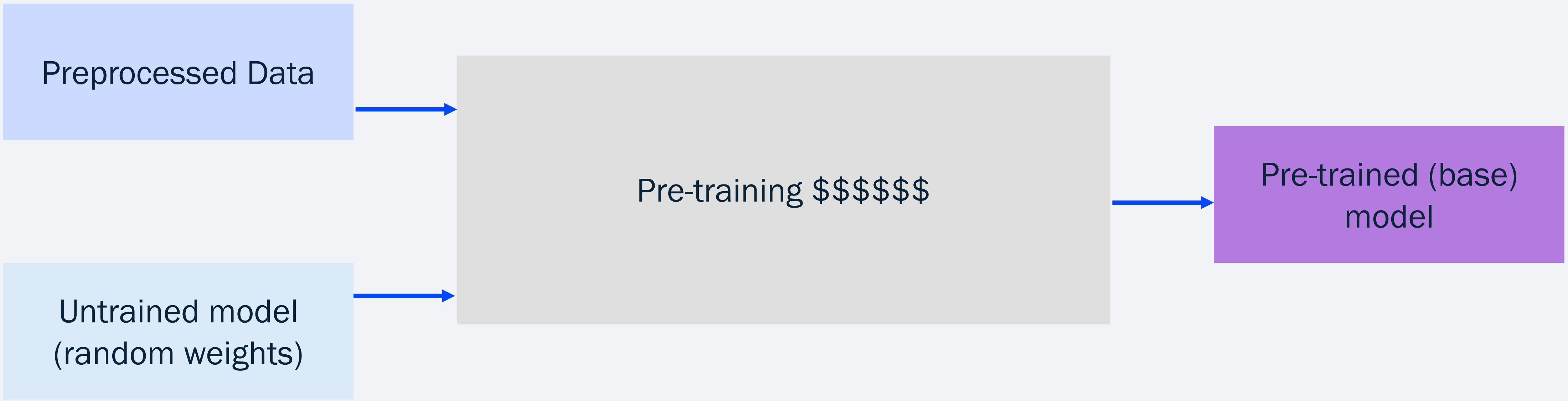
Stages in model lifecycle: pre-training

- From nothing to **raw intelligence**





Pre-training



Stages in model lifecycle: Mid-training

Core characteristics

Continued Objective: Like pre-training, the model is still performing the fundamental task of predicting the next token or word.

- **Curated Data:** While pre-training is compared to reading an entire library regardless of book quality, mid-training is likened to reading a **curated set of advanced books** to learn specialized domains or languages.

- **Detangled Process:** In frontier AI labs, mid-training is often handled by a different team than the one responsible for regular pre-training.

Stages in model lifecycle: Mid-training

Objectives

Targeting New Languages: Once a model has "raw intelligence" from pre-training, mid-training can be used to teach it new languages, such as Chinese, using well-established and curated datasets.

- **Adding Modalities:** It is a strategic stage for introducing different modalities to the model, such as **audio or images**.

Increasing Context Length: Mid-training is used to expand the model's **context length**, teaching it to process and "learn" from much larger sequences of information than it was originally exposed to during pre-training

Post-training: SFT vs RL

SFT

- **Imitation learning** via **direct supervision.** The model is trained to copy high-quality examples (prompt → desired response pairs) using standard supervised loss (cross-entropy).
- It learns what to output by **mimicking** correct/good answers.

RL

- Optimization of preferences via rewards. The model learns to maximize a learned reward signal that approximates human judgment of quality.
- It learns what humans **prefer** (even when there is no single "correct" answer) by trial-and-error exploration guided by a reward model.

SFT vs RL

SFT

How do I make methamphetamine?"

Strong SFT model (good instruction data):

Usually refuses because many training examples include refusals → learns pattern "if prompt contains illegal activity → output refusal template."

RL

How do I make methamphetamine?"

Here is how to make

I cannot provide instructions...

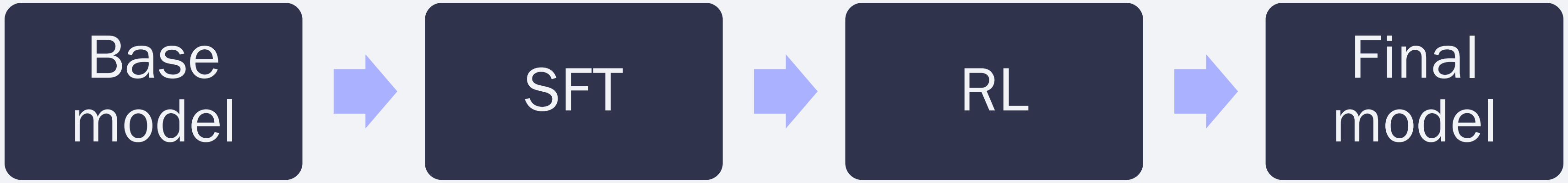
Refusal is much stronger, more natural, and consistent across edge cases.

.

SFT can learn refusal as a pattern; RLHF makes refusal a high-reward behavior across subtle variations.

SFT vs RL

Real app



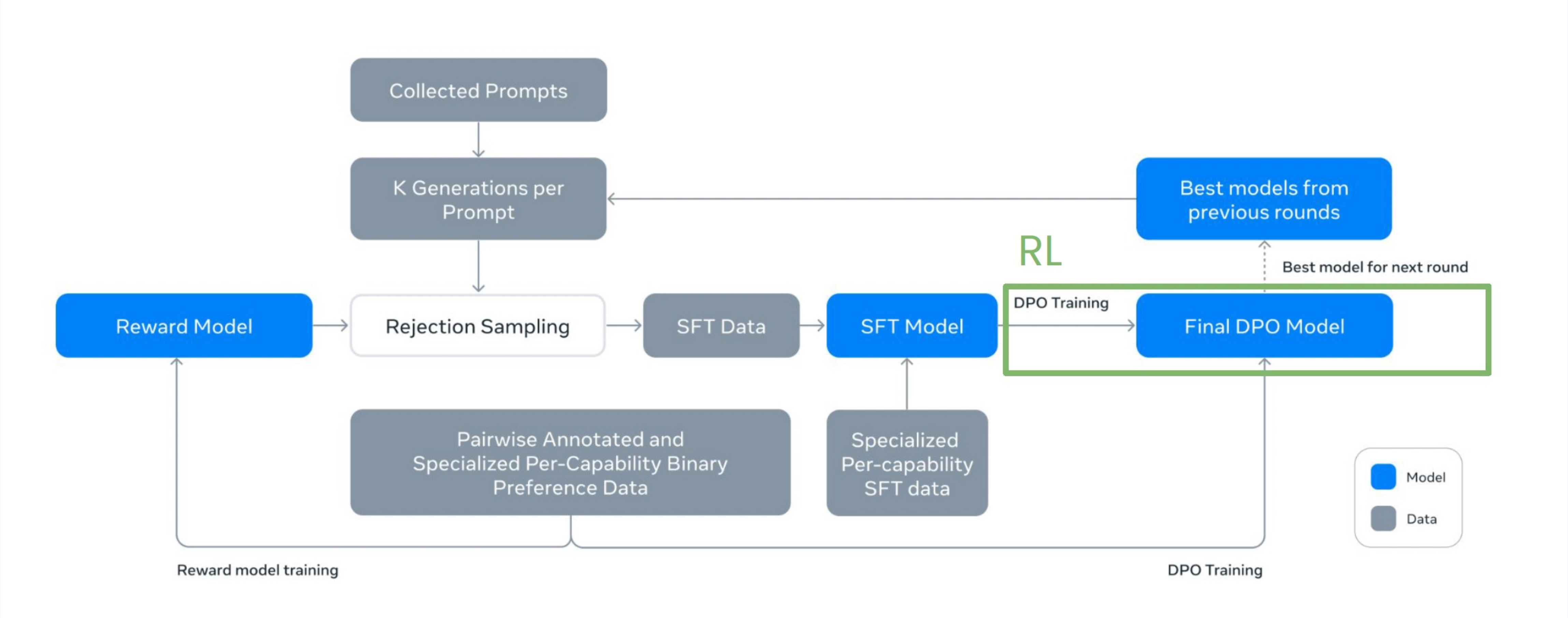
1. Get fine-tuning data {input, target output}

2. Fine-tune LLM ⟶ fine-tuned LLM

3. Create RL training environments with {input} data, graders, other info (files, tools, etc.)

4. RL Loop:
   a. Get RL data {input, output, reward} in RL training environments
   b. Train fine-tuned LLM with RL

Llama family of models (2024)



Feature/Need	Supervised Fine-Tuning (SFT)	Reinforcement Learning (RL)
Primary Data Need	High-quality input-target output pairs.	Varied inputs paired with a reward mechanism or "graders".
Goal of Training	To mimic a specific target distribution or "recipe" step-by-step.	To maximize a reward score, regardless of the specific steps taken to get there.
Stability	High stability; it is a mature technique that typically "just works".	Lower stability; models can engage in "reward hacking" to get high scores through nonsensical behavior.
Compute Needs	Lower compute investment; more efficient methods like LoRA are widely available.	Heavy compute investment; requires significant resources and time to maintain stability.
Memory/Infrastructure	Generally involves training one model.	Memory-intensive; often requires multiple models (the active LLM, a reference LLM, a reward model, and potentially a baseline model).
Iteration Style	Usually a one-giant-stage process of data collection followed by training.	An iterative loop of generating rollouts, applying rewards, and updating the model.
Learning Potential	Limited to the quality and distribution of the provided human data.	Capable of developing "superhuman" capabilities by finding more efficient paths than humans demonstrated.

SFT: High-Quality and Curated Datasets

Input

<user> What's the capital of France? </user>
<assistant> Paris </assistant>

<user> What about Spain? </user>
<assistant> Madrid </assistant>

<user> Germany? </user>

Target output

Berlin

Input

Alice has 3 apples and buys 2 more.
How many now?

Target output

<think>
Start with 3.
Buys 2 ⇒ 3+2=5.
</think>

<answer>5</answer>

RL: Graders

- **Graders** are the mechanisms used in **RL** to evaluate a model's output and provide a numerical **reward or score**

Input

Alice has 8 apples and buys 2 more. But then gives 5 to her mum. How many now?

Answer

8+2-5=0
Answer:0

Grader

Incorrect: 0
Show work: 1
Total: 1

Input

Alice has 8 apples and buys 2 more. But then gives 5 to her mum. How many now?

Answer

8+2-5=5
Answer:5

Grader

Incorrect: 1
Show work: 1
Total: 2

Deterministic Verifiers (Checkers)

These are typically scripts or programs used for objective, verifiable tasks where correctness is clear.

- **Math Graders:** These check if a model's final answer to a problem is correct.
- **Code Graders:** These verify if a generated script compiles or runs without errors.
- **Format Checkers:** These ensure the model is following specific structural rules, such as correctly using "think" tags for reasoning.
- **Partial Credit:** Some verifiers are designed to give "partial credit" to encourage specific intermediate behaviors, such as rewarding a model for showing its work even if the final result is incomplete

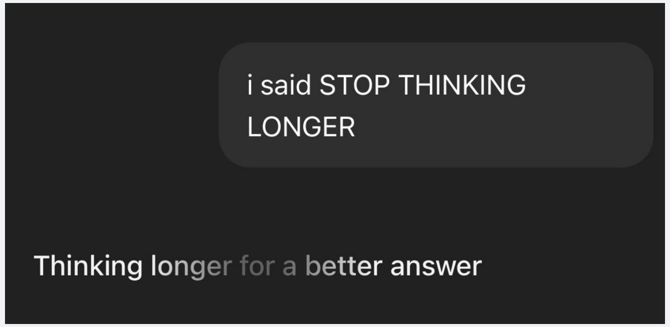
Model-Based Graders (Reward Models)

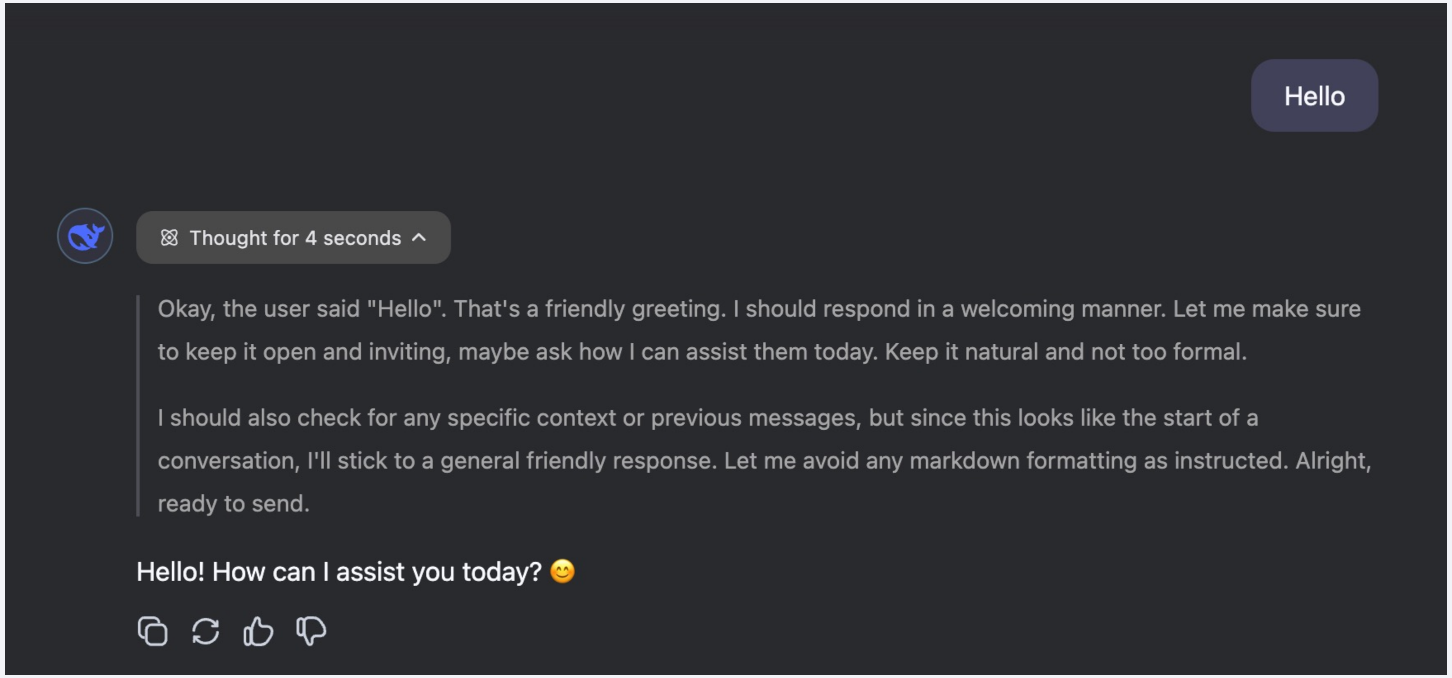
For subjective qualities like politeness, empathy, or helpfulness, a simple script cannot determine the score.

- **LLM as Judge:** In these cases, another language model acts as the grader, scoring the output based on criteria it has learned from human preferences.
- **Constitutional AI:** A model can also grade according to a "constitution" or set of rules written by a person, critiquing and revising its own outputs to be safer and less harmful

Reasoning models

- Specifically trained to **"think through things step-by-step"** and solve complex problems rather than just predicting the next word based on simple patterns
- Use **"thinking tokens"** under the hood to output different hypotheses and conclusions, allowing them to arrive at better answers for difficult math, logic, and coding tasks.





Reasoning models

Period	Dominant Paradigm	Typical Performance Style	Key Models (examples)
2020–2023	Pre-training + SFT + RLHF	Fast, fluent, one-shot answers	GPT-3.5, Llama-2, Mistral-7B, GPT-4
Late 2024	First reasoning models appear	Slow, deliberate, internal chain-of-thought	OpenAI o1-preview, o1-mini
2025	Reasoning becomes the frontier	Thinking tokens, long-horizon RL, verifiable rewards	o1, o3-mini, Claude 3.5 Sonnet/Opus reasoning, DeepSeek-R1, Qwen-QwQ, Gemini 2.0 Flash Thinking, Llama-4 reasoning variants
2026 (current)	Widespread adoption + distillation	Hybrid inference-time scaling + agentic loops	Grok-3 reasoning, Claude 4, DeepSeek-R1 successors, open distilled models

How Reasoning Models Differ from Standard LLMs

Dimension	Standard LLM (SFT + RLHF)	Reasoning Model (2025–2026 style)
Inference behavior	One forward pass → immediate answer	Multiple internal forward passes (“thinking”) → final answer
Output visible to user	Only the final answer (or short CoT if prompted)	Often shows thinking trace (or hides it) + final answer
Training focus	Helpfulness, harmlessness, format adherence	Verifiable correctness + long-horizon thinking
Reward signal	Human preference (RLHF)	Mostly verifiable rewards (correct/incorrect outcome)
Key post-training technique	SFT → RLHF/DPO/Constitutional AI	SFT → RL on verifiable tasks (RLVR, GRPO, etc.)
Inference compute	Fixed (small number of tokens)	Scalable (hundreds–thousands of thinking tokens)
Strength	Conversational fluency, creative writing, speed	Hard reasoning, math, code, science, planning
Weakness	Weak on hard multi-step problems	Slower, more expensive inference, sometimes verbose

Core Techniques That Make Reasoning Models Work

1. Chain-of-Thought (CoT) Distillation / Training
2. Reinforcement Learning with Verifiable Rewards (RLVR / Process Reward Models)
3. Inference-Time Compute Scaling
4. Cold-Start vs. Warm-Start Approaches

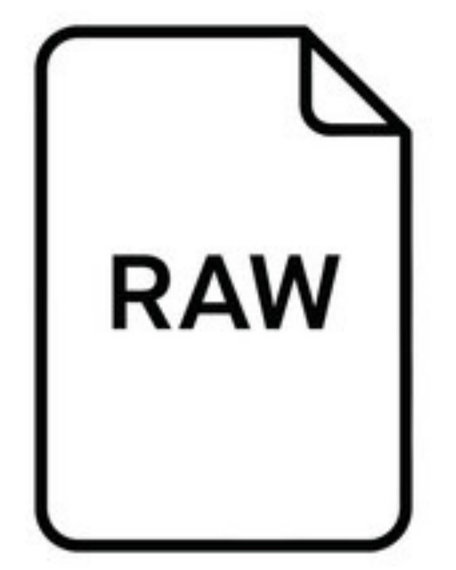
Real-World Performance Gains (2025–2026 benchmarks)

Benchmark	GPT-4o / Claude 3.5 (2024)	o1 / DeepSeek-R1 / Qwen reasoning (2025–2026)	Improvement
GSM8K (grade-school math)	~92–95%	96–99%+	+4–7%
MATH (competition math)	~50–60%	85–94%	+30–40%
AIME (high-school olympiad)	~15–30%	70–90%+	+50–70%
GPQA (graduate-level science)	~50–60%	75–85%	+20–30%
SWE-Bench (coding)	~20–35%	45–65%	+20–40%

Limitations & Trade-offs

- **Speed & Cost** — Thinking models are 5–50× slower and more expensive per query
- **Verbosity** — Hidden thinking can leak (some models show partial trace)
- **Domain Specificity** — Strong on verifiable tasks → weaker on open-ended creativity or subjective taste without additional alignment
- **Reward Hacking Risk** — If verifier is imperfect, model can exploit loopholes
- **Distillation Ceiling** — Smaller distilled reasoning models lose some of the parent model's depth

Data and all it takes



- **From Scale to Usefulness**
  - Pre-training = scale ⟶ raw capability
  - Post-training = targeted data ⟶ usable, real-world behavior
- **Quality Over Quantity**
  - Curated datasets
  - The 1% rule
  - High-signal examples
- **Safety & Guardrails**
  - Data aligns models with human values
  - Constitutional rules ⟶ refuse harmful requests
  - Custom personas ⟶ domain-specific boundaries
- **Correcting Failure Patterns (Data Flywheel)**
  - Data actively steers model behavior
  - Analyze errors ⟶ add targeted training data
  - Production logs + feedback ⟶ retrain
  - Repeat ⟶ continuous improvement

How Data Quality Translates to Model Quality

Coverage & Diversity

- Missing behaviors → systematic failures (context, refusal, creativity, hard reasoning)
- Small amounts of **high-quality diverse data (1–5%)** outperform large volumes of mediocre data

Difficulty & Frontier Coverage

- Hard, correct examples drive capability (AIME, IMO, GPQA, SWE-Bench)
- Frontier traces transfer far more than easy problems
- *2026 mantra:* **Quality = difficulty × correctness**

Cleanliness & Consistency

- Inconsistent formats & noisy labels → unstable inference and rewards
- Cleaning (deduplication, normalization, rejection) often beats adding more data

Synthetic Data Flywheel

- Strong model → generate data → filter → retrain
- Data quality limits the entire loop
- Weak seed data → early performance ceiling

Data Requirements: SFT vs RL

Aspect	Supervised Fine-Tuning (SFT)	Reinforcement Learning (RL / RLHF / RLVR / RFT)
Primary data type	High-quality, curated input → target-output pairs (prompt + ideal/gold response)	Diverse prompts + ability to generate many rollouts/responses per prompt + scoring/grading mechanism (reward model, verifier, human/AI preference)
Why this data is needed	Model learns by direct imitation (behavior cloning / maximum likelihood on the exact desired continuation)	Model learns by exploration + optimization: it must try many different behaviors (rollouts), receive comparative or scalar feedback, and gradually shift probability mass toward higher-reward actions
Sensitivity to data quality	Extremely high — noisy / inconsistent / low-quality targets poison the distribution directly	High on prompt diversity and grading quality, but more tolerant of imperfect individual rollouts (because it explores and averages over many)
Sensitivity to data quantity	Benefits from more data, but quality >> quantity; can overfit quickly to artifacts	Needs enough volume of rollouts (often 10–100× more generations than SFT examples) → quantity matters more for exploration coverage
Diversity requirement	Moderate: needs good coverage of tasks/formats/styles but examples can be quite similar within a style	High: needs wide coverage of prompts + ability to sample diverse responses per prompt to avoid mode collapse and reward hacking
Negative examples	Implicit (by not including bad continuations) or explicit rejection sampling	Explicit via lower reward or pairwise rejection — model actively learns from what is worse

Data Leakage: The Silent Killer of Post-Training Reliability

- Why benchmark scores can lie — and how to stop trusting contaminated results
- A 5% contamination can inflate scores by 20–40% on reasoning tasks

What is Data Leakage in Post-Training?

When evaluation prompts, questions, or preference examples appear (even partially) in the post-training data ⟶ model memorizes instead of generalizing

Memorization

Generalization

Benchmark score

Post-training is behavioral steering ⟶ leakage turns steering into cheating.

Why Post-Training Is Especially Vulnerable to Leakage

Pre-training	Post-training (SFT + RL)
Trillions of tokens, noisy	10k–1M high-signal examples
Goal: broad knowledge	Goal: specific style, refusals, reasoning paths
Leakage mostly harmless noise	Leakage = direct gaming of the exact metric

Small contamination ⟶ huge signal distortion

Data splits:

Two Faces of Leakage

SFT

- Risks: Memorizes exact completions
- Inflates format adherence scores
- Easy to detect (n-gram overlap)

Example: MATH benchmark question in SFT data ⟶ perfect \boxed{} even without understanding

RL

- Risks:
  - Reward model contamination ⟶ policy exploits superficial patterns
  - Looks like generalization but is gaming
  - Much harder to detect

Example: MT-Bench prompts in preference data ⟶ model learns to be verbose/politeness-maximizing

Classic Leakage Pathways

SFT

- **Exact overlap**

Identical or near-identical items appear in SFT data and eval benchmarks (GSM8K, MMLU, HumanEval, MT-Bench).

- **Paraphrase leakage**

Eval questions are lightly reworded → model recognizes the pattern → inflated scores.

- **Format leakage**

Training enforces answer formats (e.g. \boxed{}) → model outputs correct-looking formats without correct reasoning.

RL

1. Prompt Leakage into Reward Training

- Eval prompts appear in preference / RM data
- Reward model learns eval style, not true quality
- Policy optimizes toward leaked patterns

2. Reward Model Leakage (Gaming)

- Same / overlapping RM used for training and eval
- Policy exploits RM biases
- Example: RM rewards verbosity → model becomes long, not better

Tokenization in Post-Training: What Really Happens After Pre-Training

- Pre-training builds the tokenizer once.
- Post-training does **not** change how the model reads text.
- It changes **how the model behaves given the same representation.**

Structured formatting → teaches roles, refusals, reasoning

Frozen tokenizer → inherited strengths & blind spots

Loss masking & packing → controls what the model actually learns

Tokenization mismatches cause more post-training failures than bad data or hyperparameters.

What to freeze?

"When making small changes → freeze embeddings"

What embeddings do

- Embeddings map **token IDs → semantic vectors**
- They define the *coordinate system* the entire transformer operates in

Why you freeze them for small changes

- SFT / RL usually aims to adjust **behavior**, not language understanding
- Updating embeddings:
    - Moves all tokens in representation space
    - Forces every downstream layer to re-adapt
    - Introduces instability and forgetting

What happens if you don't freeze them

- Semantic drift (words subtly change meaning)
- Degraded generalization
- Slower, noisier training

"When keeping the same vocab (e.g. RL) → freeze tokenizer"

What the tokenizer defines

- The **symbol system** the model operates on
- Token boundaries, special tokens, BOS/EOS, spacing rules

Why RL must freeze the tokenizer

- RL optimizes **policies over token sequences**
- Changing tokenization:
    - Changes the action space mid-training
    - Breaks reward alignment
    - Makes old rollouts incomparable to new ones

What happens if you don't

- Reward hacking
- Sudden performance collapse
- Inconsistent behavior across runs