



ELEN90088

System Optimization and Machine Learning

Post-training, Multimodality, and Agents for Large Language Models

Guest Lecture

Presented by **Dr. Xuetong Wu**

Dr. Xuetong Wu

Senior AI Engineer at **Heidi Health** ■ Ph.D., University of Melbourne (2019–2023)

► Now

Lead LLM fine-tuning & eval for clinical AI at **Heidi Health**.

SFT / LoRA / DPO / RFT, ASR (Whisper, CTC, TDT), multi-modal pipelines, production deployment on vLLM / SGLang.

► Research

Ph.D. on *information-theoretic analysis of transfer learning*.

Advisors: Jingge Zhu, Jonathan Mantion, Uwe Aickelin. Active UniMelb collaborator on LLM / multi-modal NLP.

► Selected work

ACL 2025, NAACL 2025
(Oral, Top 15%)

IEEE Trans. Inf. Theory
'24, '25

JMLR 2024

2023 Best EEE Ph.D. Thesis Prize,
UniMelb.

Why this lecture: post-training, multimodal, and agents are what I work on day to day — I'll mostly share what has worked in practice.

wuxuetong.com ■ xuetong@heidihealth.com

What is Heidi Health?

*A clinical AI platform. **Scribe** is the flagship; **Evidence** and **Calls** extend the same speech and LLM core to new clinical surfaces.*

► Scribe

core / flagship

consult audio
→ clinical note

Ambient AI scribe.

Listens to the visit, drafts a SOAP, referral, or discharge note, pushes it to the EHR — clinician keeps the final say.

► Evidence

derivative

clinical question
→ cited answer

Grounded clinical Q&A.

Same model + retrieval over trusted sources (BMJ, DynaMed, MIMS, ...); inline citations, no ads.

► Calls

derivative

patient call → voice agent

AI phone agent for clinics.

Books appointments, sends reminders, runs intake & follow-ups; same ASR+LLM core plus TTS and tools.

Shared core: all three products run on the same speech and LLM stack — building that stack is exactly what the next 150 minutes are about.

How AI is used inside Heidi

One stack powers Scribe, Evidence, and Calls — and almost every chapter of this lecture shows up in it.

1. Speech in. ASR (Whisper-style, CTC / TDT) turns clinic / phone audio into tokens.

2. Multimodal alignment. speaker, language, medical entities aligned with text tokens.

3. Clinical LLM. base model + **SFT** on clinical text + **RFT** for accuracy / style.

4. Tools, retrieval, agents. structured outputs, function calls, **RAG** (Evidence), agentic flows (Calls / charts).

5. Fast serving. vLLM / SGLang, KV-cache, batching → low TTFT under clinic load.

Maps to this lecture

- **Ch. 4 Multimodal** → stages 1 & 2
- **Ch. 2 SFT + Ch. 3 RFT** → stage 3
- **Ch. 6 Agents** → stage 4
- **Ch. 5 Deployment** → stage 5

Today's recipes are exactly what powers a real clinical product.

Learning goals and lecture map

By the end of this lecture you will

- ▶ Understand why **post-training** matters after pretraining.
- ▶ Distinguish **SFT**, **preference optimization**, and **RFT**.
- ▶ See how **images, audio, video, and GUI** become tokens.
- ▶ See how **agents** combine models, tools, memory, protocols.
- ▶ Reason about **inference and deployment** at scale.
- ▶ Leave with a mental model for **multimodal agent systems**.

Lecture map (150 min)

1.	Intro	20 min
2.	SFT	28 min
3.	RFT	35 min
4.	Multimodal	25 min
5.	Deployment	18 min
6.	Agents	24 min

Total ~150 min

narrative arc: pretraining → behavior shaping → interactive task completion

Abbreviations used throughout the lecture

Training & optimization

LLM	Large Language Model
SFT	Supervised Fine-Tuning
RFT	Reinforcement Fine-Tuning
RLHF	RL from Human Feedback
RLAIF	RL from AI Feedback
DPO	Direct Preference Optimization
PPO	Proximal Policy Optimization
GRPO	Group Relative Policy Opt.
RM	Reward Model
KL	Kullback–Leibler divergence
CE	Cross-Entropy (loss)
LoRA	Low-Rank Adaptation
PEFT	Parameter-Efficient Fine-Tuning

Multimodal & modeling

MLLM	Multimodal LLM
VLM	Vision–Language Model
ViT	Vision Transformer
ASR	Automatic Speech Recognition
CTC	Connectionist Temporal Class.
TDT	Token-and-Duration Transducer
OCR	Optical Character Recognition
TTS	Text-to-Speech
MoE	Mixture of Experts
CoT	Chain-of-Thought
ICL	In-Context Learning
RAG	Retrieval-Augmented Generation
NLI	Natural Language Inference

Agents & systems

ReAct	Reasoning + Acting
MCP	Model Context Protocol
A2A	Agent-to-Agent protocol
GUI	Graphical User Interface
API	Application Prog. Interface
SDK	Software Development Kit
KV	Key–Value (attention cache)
HBM	High-Bandwidth Memory
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
TTFT	Time To First Token
TPOT	Time Per Output Token
vLLM	virtual LLM (serving engine)

Notation used throughout the lecture

Data & probabilities

x, y	prompt, response (tokens)
y^+, y^-	preferred / rejected response
$z = [x; y]$	prompt $\oplus$ response, length $ z $
$z_t, z_{<t}$	token t ; prefix $z_1, \dots, z_{t-1}$
$m_t \in \{0, 1\}$	SFT loss mask (1 on response)
$\mathcal{D}, \mathbb{E}_{\mathcal{D}}$	dataset; expectation over it
$p_{\theta}(y x)$	model probability of y given x
$\sigma(z)$	sigmoid $1/(1 + e^{-z})$
$H(p, q)$	cross-entropy $-\mathbb{E}_p[\log q]$
$\text{KL}(p \parallel q)$	Kullback–Leibler divergence

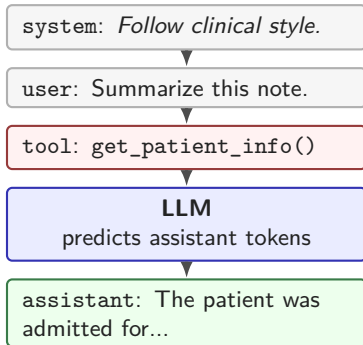
Parameters, policies & rewards

θ, ϕ	policy / RM parameters
$\pi_{\theta}, \pi_{\text{ref}}$	learnable / frozen reference policy
π_{old}	rollout-time policy (PPO)
$r(x, y), r_{\phi}$	scalar reward; reward model
β	KL regularization strength
α, r	LoRA scaling, rank ($r \ll d, k$)
A_t, ρ_t, ε	advantage; ratio; PPO clip radius
$\gamma, V(s)$	discount factor; state-value fn.
G, A_i	GRPO group size; group-norm. adv.
$Z(x)$	partition function (normalizer)
$\mathcal{L}_{\bullet}, J(\theta)$	stage losses; RL obj. (reward – KL)

Convention: θ is always what we *train*; ϕ is a separate scorer;
 π_{ref} is the policy we *anchor to* (usually the SFT checkpoint).

Common LLM words: prompt, context, response

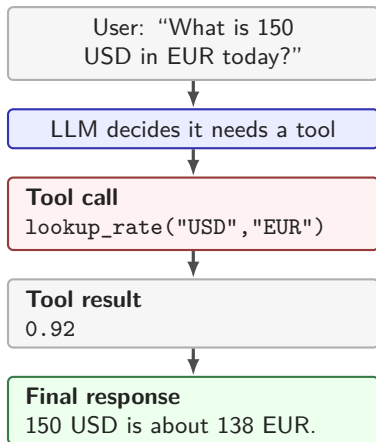
- ▶ **Prompt:** the instruction and information we give the model.
- ▶ **Role:** who wrote each message: system, user, tool, or assistant.
- ▶ **Context:** everything the model can see right now: messages, documents, tool results, chat history.
- ▶ **Response:** the tokens the model generates next.
- ▶ **Token:** the unit the model reads and writes; roughly a word piece.



Mental model: an LLM is a **next-token predictor** wrapped in a **chat interface**.

What is a tool?

- ▶ A **tool** is external code the model can ask the runtime to execute.
- ▶ Examples: search, calculator, database query, file read, calendar, browser, code sandbox.
- ▶ The model usually does **not** run the tool itself: it emits a structured **tool call**; the app executes it.
- ▶ Tool results are fed back into the context, then the model writes the final answer.



This is the seed of agent systems: model + tools + memory + control loop.

Major LLM vendors and model families

Snapshot as of May 2026; exact model versions change fast.

Closed source / hosted APIs

OpenAI

GPT, o-series

Anthropic

Claude

Google DeepMind

Gemini

xAI

Grok

ByteDance

Doubao

Tencent

Hunyuan

Open source / open weights

Meta

Llama

Alibaba

Qwen

Moonshot AI

Kimi

DeepSeek

DeepSeek

Google DeepMind

Gemma

Zhipu AI

GLM

Practical habit: ask “which provider?”, “which model family?”, and “closed source or open source?”

Model names and formats

Qwen2.5-7B-Instruct

7B instruction-following chat model

DeepSeek-R1-Distill-Qwen-32B

reasoning model distilled into a 32B model

Llama-3.1-8B-Instruct

Q4_K_M.gguf

local quantized GGUF file

Gemma-3-27B-it

it means instruction-tuned

- ▶ **Family/version:** Llama-3.1, Qwen, DeepSeek, Claude.
- ▶ **Size:** 8B, 70B, 671B means roughly billions of parameters.
- ▶ **Base:** pretrained model, not necessarily good at following instructions.
- ▶ **Instruct/Chat:** post-trained to follow user instructions and conversation format.
- ▶ **Thinking/Reasoning:** spends extra tokens on intermediate reasoning.
- ▶ **Vision/Audio:** accepts non-text inputs as well as text.
- ▶ **Format:** GGUF, safetensors, ONNX are file/runtime formats.
- ▶ **Quantization:** Q4, Q8, FP8 compress weights; cheaper, sometimes less accurate.

Rule of thumb: the name tells you who made it, how big it is, how it was trained, and how it can run.

1. From pretraining to post-training

The AI products people already use

Chat & reasoning

ChatGPT ▪ Claude ▪ Gemini ▪ Grok ▪ DeepSeek ▪ Kimi

Coding agents

Cursor ▪ Windsurf ▪ Claude Code ▪ GitHub Copilot ▪ Codex

Image generation

Midjourney v7 ▪ GPT-Image-2 ▪ Ideogram ▪ Nano Banana

Video generation

Sora 2 ▪ Veo 3 ▪ Runway Gen-4 ▪ Kling ▪ Pika

Voice & audio

ElevenLabs ▪ Suno / Udio ▪ NotebookLM ▪ GPT-4o voice

Agent harnesses

OpenClaw ▪ Hermes Agent ▪ Computer Use ▪ Manus

*Every product above starts from the same pretrained backbone — **post-training** is what turns a raw LM into any of these.*

What post-trained LLMs can actually do

► Instruction following

"summarize this paper in 3 bullets"

→ *three well-formed bullets in the requested style*

unlocked by: SFT

► CoT & reasoning

"solve $\int_0^1 x^2 dx$ "

→ *<think>derive $x^3/3$
eval 0 → 1 </think>*

answer: 1/3

unlocked by: RFT + verifiers

► Tool / function call

"weather in Melbourne?"

→ *{ "tool": "weather",
"args": { "city": "Mel" } }*

unlocked by: SFT + harness

► Image understanding

"what is abnormal in this X-ray?"

→ *opacity in the right lower lobe; suggests pneumonia (VQA, captioning, OCR)*

unlocked by: alignment + VIT

► Speech: STT & TTS

audio waveform

⇒ *"Good morning,
how can I help?"*

(Whisper, GPT-4o voice, Qwen2.5-Omni)

unlocked by: audio tokens

► Autonomous agent

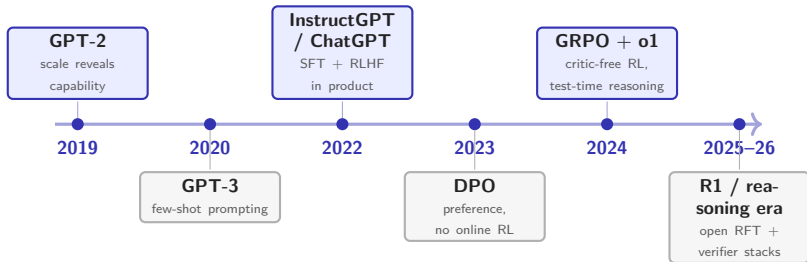
"fix this GitHub issue"

→ *reads repo, edits files,
runs tests, opens a PR
(SWE-bench, τ^2 -bench)*

unlocked by: RFT + harness

*All six capabilities start from the same next-token pretraining — the rest of this lecture is **how** each one is unlocked.*

A brief history of post-training



- ▶ Pretraining → behavior shaping (SFT) → preference / verifier training → reasoning-first stacks.
- ▶ What began as an alignment patch is now a full **post-training pipeline**.

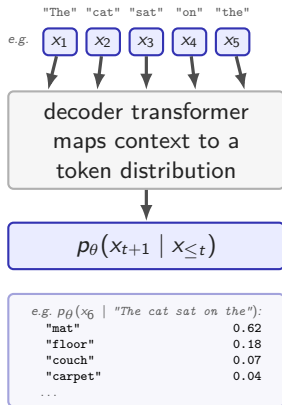
Autoregressive pretraining: the common starting point

Factorize the joint and train by **teacher forcing**:

$$p_{\theta}(x_{1:T}) = \prod_{t=1}^T p_{\theta}(x_t \mid x_{<t}),$$

$$\mathcal{L}_{\text{pt}}(\theta) = -\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T \log p_{\theta}(x_t^{(n)} \mid w_{<t}^{(n)}).$$

- ▶ Simple intuition: at every position, ask the model to put high probability on the **actual next token**.
- ▶ After trillions of examples, it learns facts, syntax, style, and broad skills — but not yet **instruction following**, **tool discipline**, or **product behavior**.



What pretraining data looks like

#1 web prose (CommonCrawl / Wikipedia)

The Yarra River flows through Melbourne and empties into Port Phillip Bay. European settlement began in 1835; the city was formally named Melbourne in 1837...

#2 code (GitHub / The Stack v2 / StarCoder)

```
def softmax(x):  
    e = np.exp(x - x.max())  
    return e / e.sum()
```

#3 textbook / math (Books / arXiv / OpenWebMath)

Theorem. For any distribution p , the entropy $H(p) = -\sum_x p(x) \log p(x) > 0$, with equality iff p is a point mass.

- ▶ **Raw & unlabeled:** no roles, no masks — **every token** contributes to the loss.
- ▶ Mixture of **web, code, books, papers, math, dialogue** — $\sim 10\text{--}20$ T tokens at frontier scale.
- ▶ Aggressive **dedup, quality & toxicity** filters before training.
- ▶ No instruction format — the model just learns **what the world writes next**.

Per step: predict every next token; lower $-\sum_t \log p_\theta(z_t \mid z_{<t})$ by one gradient step — push mass onto the **actual next token**.

Model architectures (1/2): the network itself

Most modern LLMs are still decoder-only transformers; new families mainly change scaling and memory design.

Axis	Common variants	Beginner intuition
Scaling	dense (Llama 3, Qwen3, Gemma 3) sparse MoE (DeepSeek V3, Qwen3-MoE, Gemini)	MoE keeps a huge model, but activates only a few expert blocks per token.
Attention	MHA → GQA → MLA ; sliding-window; hybrid SSM	Different ways to make long context cheaper and faster.

MHA full attention heads	GQA / MLA compress KV cache	MoE route to experts	SSM cheap long context
------------------------------------	---------------------------------------	--------------------------------	----------------------------------

*Takeaway: same next-token objective; architecture choices change **capacity**, **memory**, and **serving cost**.*

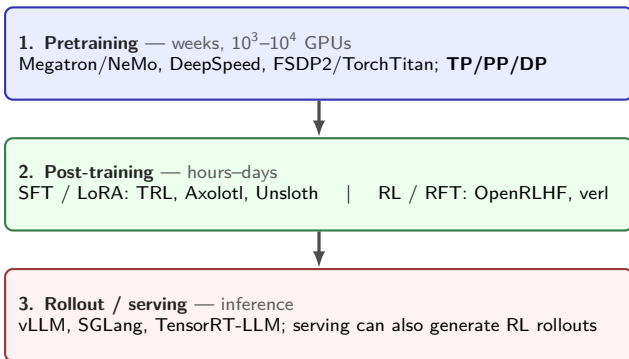
Runtime behavior (2/2)

After training, the same model can be run with different context, reasoning, and input settings.

Axis	Common variants	Key idea
Reasoning	plain LM <think> traces (o-series, DeepSeek-R1, Hermes 4)	Thinking uses more tokens to improve hard answers.
Context	RoPE / YaRN / NTK scaling; 100k–1M+ token windows	Longer context lets the model read more, but costs more.
Modality	text-only vision, audio, video, action tokens	Non-text inputs are converted into tokens the model can read.

Production trade-off: more reasoning, context, and input types can improve capability, but increase **latency**, **memory**, and **cost**.

Training frameworks and infrastructure



TP = tensor parallelism *PP* = pipeline parallelism *DP* = data parallelism

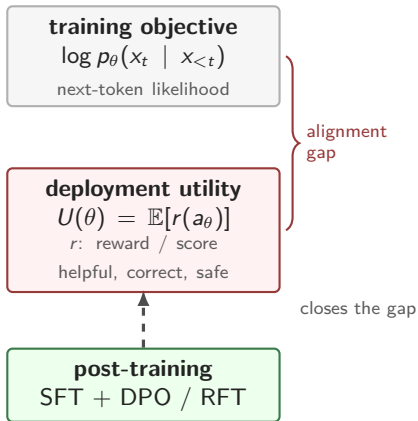
Why pretraining alone is not enough

- Deployment quality depends on a different objective:

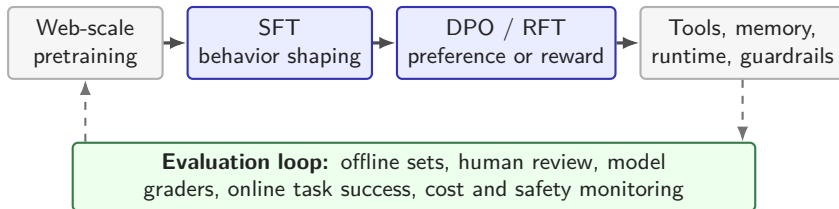
$$U(\theta) = \mathbb{E}_{(x, \mathcal{E})} [r(a_\theta(x, \mathcal{E}))],$$

where $a_\theta(x, \mathcal{E})$ is the model's behavior, $r(\cdot)$ is the **reward / score**, and $\mathcal{E}$ covers tools, users, policies, or environment.

- High likelihood $\neq$ **most useful**, **truthful**, or **safe** action.
- The gap widens when the model must **format outputs**, **call tools**, or **finish long-horizon tasks**.

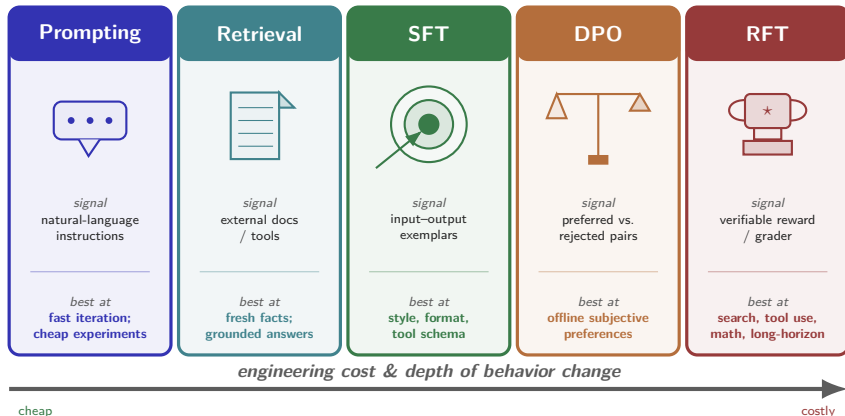


The post-training stack in one picture



- ▶ **SFT** teaches what good behavior looks like.
- ▶ **DPO / RFT** shifts mass toward preferred or high-reward trajectories.
- ▶ A deployed system also needs **state**, **tooling**, and **measurement**.

Which adaptation lever should you use?

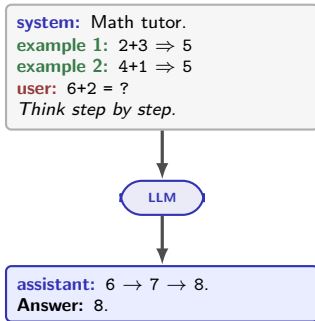


Rule of thumb: move right only when the lever on the left stops paying off.

Prompting

No training, no GPUs — just better context.

- ▶ **Zero-shot:** task only. *“Translate to French: ...”*
- ▶ **Few-shot (ICL):** add 2–3 examples; no weight update.
- ▶ **Chain-of-thought:** ask for steps; helps math/logic.
- ▶ **System / role prompt:** sets tone, format, constraints.
- ▶ + fastest way to change behavior.
- ▶ – context-limited; brittle wording.

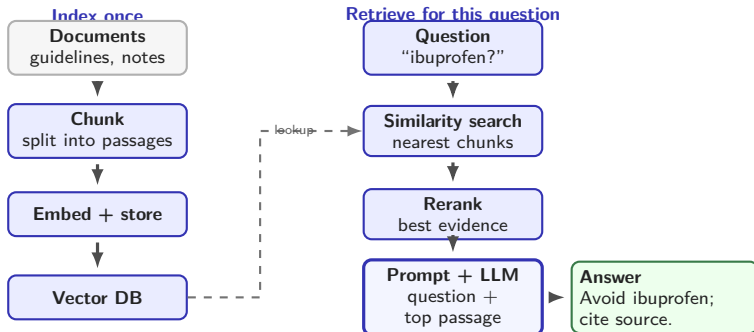


few-shot prompt $\rightarrow$ CoT response

[2][3]

Retrieval-augmented generation (RAG)

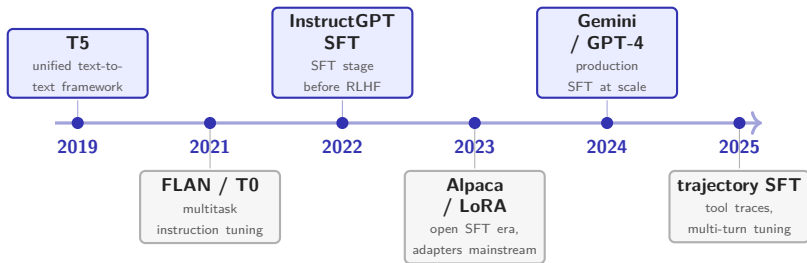
RAG means: build a searchable memory, retrieve relevant chunks, then answer from them.



Good retrieval matters: bad chunking or bad reranking means wrong evidence in, wrong answer out.

2. Supervised fine-tuning (SFT)

A brief history of instruction tuning



- ▶ From zero-shot instructions to curated trajectories — each era expands what SFT can teach.
- ▶ Today's SFT is as much about **tool traces and schemas** as about natural chat.

SFT data and the masked cross-entropy objective

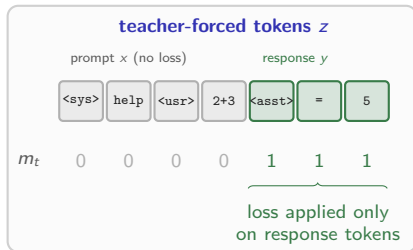
Data — *prompt–response demonstrations*

- ▶ Each example: prompt x (**system** / **user** / **tool**) $\rightarrow$ response y .
- ▶ Target: text, JSON, tool-call args, or a multi-turn trajectory.
- ▶ **Response-only masking**: loss only on assistant tokens.

Objective — *masked next-token loss*

$$\mathcal{L}_{\text{SFT}}(\theta) = - \sum_{t=1}^{|z|} m_t \log p_{\theta}(z_t \mid z_{<t}).$$

$z = [x; y]$, $m_t \in \{0, 1\}$; equivalent to maximizing $\mathbb{E}_{\mathcal{D}}[\log p_{\theta}(y \mid x)]$ on a **curated task distribution**.



What SFT data looks like

#1 chat instruction (ChatML-style; gray=context, blue=loss)

```
[system] You are a concise assistant.  
[user] Explain entropy in one sentence.  
[assistant] Entropy is the average  
             surprise of a random  
             variable's outcomes, in bits.
```

#2 tool call (JSON function-calling format)

```
[user] What's the weather in Melbourne?  
[assistant] {"tool": "get_weather",  
             "args": {"city": "Melbourne"}}  
[tool] {"temp_c": 18, "cond": "sunny"}  
[assistant] It's 18°C and sunny.
```

#3 thinking trace (reasoning-style answer)

```
[user] Which is larger: 9.11 or 9.8?  
[assistant] <think> Compare 9.11 with 9.80.  
             9.80 is larger. </think>  
             Answer: 9.8.
```

- ▶ Each row is a **full dialogue** with role tags (ChatML, Llama-3, Qwen, ...).
- ▶ Loss on **assistant tokens only** — system / user / tool are context ($m_t=0$).
- ▶ Quality $\gg$ quantity: **5–50k hand-curated** rows often beat 1M scraped.
- ▶ Open recipes mix **general, code, math, tool-use, safety refusals** in calibrated ratios.

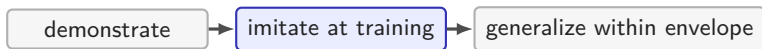
What SFT is especially good at

High-return use cases

- ▶ Fixed **output schema** / style
- ▶ **Domain terminology** and tone
- ▶ **Tool-call** formatting
- ▶ **Refusals** and safe boundaries
- ▶ Prompt → smaller-model distillation

What SFT usually does not solve

- ▶ Deep search over many candidates
- ▶ Long-horizon agent interaction
- ▶ Optimizing **verifiable outcome metrics**
- ▶ Exploring rare-but-correct answers



Where SFT data actually comes from today

Humans write it
People hand-write Q/A pairs.

OpenAssistant, ShareGPT

slow, high quality

Big model writes it

A stronger model writes synthetic Q/A data.

Self-Instruct, Alpaca

scales fast

Copy the thinking

Distill long <think> traces into a student.

R1-Distill, Open Thoughts

reasoning style

Self-improvement

Sample answers; keep verifier-approved ones.

STaR, RFT rollouts

verified data

- ▶ Modern SFT data is **mostly model-generated**: humans curate, models write.
- ▶ Data is less scarce; *training* a 70B+ model is still expensive.
- ▶ **Next: LoRA*** — train a tiny add-on instead of the whole model.

* LoRA = Low-Rank Adaptation.

LoRA: parameter-efficient reparameterization

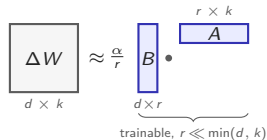
Freeze a weight matrix $W_0 \in \mathbb{R}^{d \times k}$ (maps input dim k to output dim d); learn a low-rank update

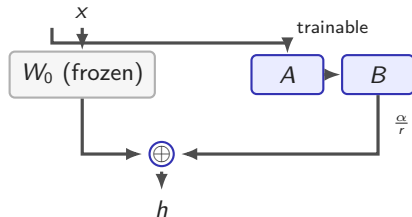
$$W = W_0 + \Delta W, \quad \Delta W = \frac{\alpha}{r} BA,$$

$$B \in \mathbb{R}^{d \times r}, \quad A \in \mathbb{R}^{r \times k}, \quad r \ll \min(d, k).$$

where r is the **rank** (capacity knob, typ. 4–64), α is a **scaling factor** (step-size knob).

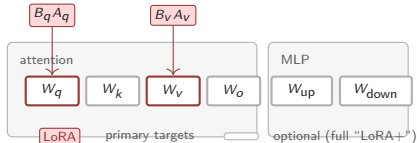
- ▶ **Init.** $A \sim \mathcal{N}(0, \sigma^2)$ (Gaussian, small variance), $B=0 \Rightarrow \Delta W=0$ at step 0.
- ▶ **Forward.** $h = W_0 x + \frac{\alpha}{r} B(Ax)$: base and adapter paths run in parallel; α/r decouples rank from step size.
- ▶ **Param budget.** full FT: dk . LoRA: $r(d+k)$. With $d=k=4096$, $r=8$: $\approx 0.4\%$.


$$\Delta W \approx \frac{\alpha}{r} \underbrace{B \cdot A}_{\substack{d \times r \\ \text{trainable, } r \ll \min(d, k)}}$$



LoRA: where it is injected and how gradients flow

Where to inject (one transformer block)



embeddings and LayerNorm are usually left untouched.

Backward pass

Let $g = \partial \mathcal{L} / \partial h$ for $h = W_0 x + \frac{\alpha}{r} B A x$.

$$\frac{\partial \mathcal{L}}{\partial A} = \frac{\alpha}{r} B^\top g x^\top, \quad \frac{\partial \mathcal{L}}{\partial B} = \frac{\alpha}{r} g (A x)^\top.$$

$\partial \mathcal{L} / \partial W_0$ is never formed, so no optimizer state is kept for the base.

Practical consequences

- **Memory:** $\sim 3|A, B|$ vs. $\sim 3|W_0|$ — large VRAM savings.
- **Performance:** often **90%+** of full fine-tuning with far fewer trainable weights.
- **Inference:** merge $W \leftarrow W_0 + \frac{\alpha}{r} B A$ — zero extra latency; hot-swap per user.
- **Caveat:** cannot learn **high-rank** changes — raise r or use full FT.

Failure modes of SFT

Imitation ceiling

Example: all demos are short answers
⇒ model never learns deep reasoning.

Shortcut learning

Example: “chest pain” ⇒ writes
ECG, even when context says no red
flags.

Distribution shift

Example: trained on SOAP; asked
for referral ⇒ wrong format.

Long-horizon brittleness

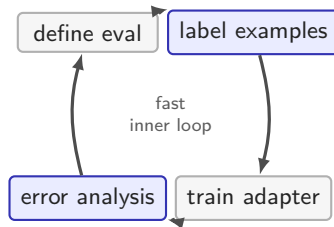
Example: one wrong field in step 2
makes the later tool call fail.

- ▶ SFT teaches imitation; use **evals + preference/RFT** when you need robust choices.

A practical SFT recipe for a lecture or project

- ▶ Write the **eval harness** before the dataset.
- ▶ Keep a few **high-signal exemplars** defining “good”.
- ▶ **Response-only masking**; explicit schemas; held-out edge cases.
- ▶ **Start with LoRA adapters**; only do full fine-tuning when LoRA can’t reach the quality you need.
- ▶ Inspect failures by hand — the loop below is the fastest path.

The SFT improvement loop



3. Preference learning and reinforcement fine-tuning

Why SFT is not enough: DPO and RFT examples

SFT: copy demos

Demo:

"write a short clinical summary"

Learns: style / format

Missing: better vs worse

No rejected answer.
No outcome signal.

DPO: prefer better answer

Same prompt:

A: clear, calm, useful

B: scary, unsupported

Push up **A**; push down **B**.

Best when humans can
compare two outputs.

RFT: reward outcome

Try many outputs:

8 code solutions → run tests
JSON form → validate
schema

Pass = reward; fail = no
reward.

Best when a verifier
can check success.

Key idea: SFT imitates. DPO compares. RFT verifies outcomes.

SFT vs. DPO vs. RFT at a glance

One-page roadmap: imitate demos, prefer better answers, or optimize a verifier reward.

SFT

supervised
demonstrations

Signal: gold (x, y) pairs

Best at: style + format +
tone

Limit: imitation ceiling on
hard reasoning

DPO

offline preference
optimization

Signal: pairs (y^+, y^-)

Best at: human choices

Limit: fixed data; no
search

RFT

online RL with
verifier or grader

Signal: scalar reward
 $r(x, y)$

Best at: verifiable tasks,
search, tool use

Limit: compute + reward
hacking

Rule of thumb: SFT first; DPO for preferences; RFT when a verifier can score fresh attempts.

RLHF is the umbrella: shape behavior after SFT with human/model feedback.

[1][5][7][8]

What RFT data looks like

#1 Preference pair (for DPO / reward-model training)

prompt: Why is the sky blue?

y^+ : Sunlight scatters off N_2/O_2 molecules; shorter (blue) wavelengths scatter most (Rayleigh).

y^- : Because the ocean reflects onto the atmosphere.

► The *judge* can be a human, a stronger LLM, unit tests, regex, or a math checker. Scores are often a sum: *correct* + *format* + *safe* – *too long*.

► **Big shift from SFT:** no fixed “gold” answers — the model writes its own training data every step.

#2 Verifier rollout (for PPO / GRPO on-policy RL)

prompt: Compute $\int_0^1 x^2 dx$.

rollout: antideriv. is $x^3/3$;
eval $0 \rightarrow 1$ gives $1/3$.

final answer: 1/3

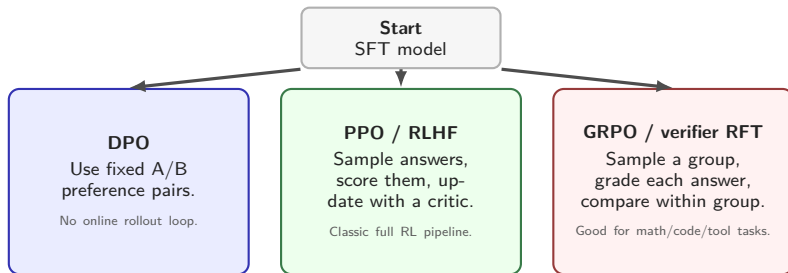
reward: +1.0 (exact match)

grader: `sympy(ans - 1/3)==0`

The teacher no longer hands over answers — it just *scores* a try (a number) or *compares* two tries (better / worse).

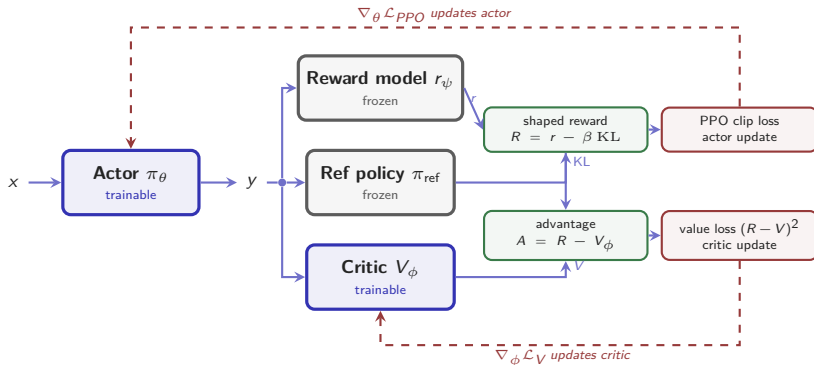
RFT method map: three common recipes

The next few slides are different ways to turn “better attempt” into a training update.



Same goal: shift probability toward better attempts.
What changes is the **feedback signal** and the **optimizer**.

Classic RLHF: who does what?



Read the diagram as four simple jobs:

- **Actor:** writes the answer; this is the model we update.
- **Reward model:** says how good the final answer looks.
- **Reference:** says “do not drift too far from the SFT model.”
- **Critic:** predicts expected score; PPO trains on **better/worse than expected**.

Next: PPO uses these signals for a small, safe update; GRPO and DPO simplify the loop.

PPO: improve, but only by a small step

1. **Roll out:** actor writes an answer; reward model / verifier scores it.

2. **Compare to expectation:** critic gives a baseline; advantage says “better or worse than expected.”

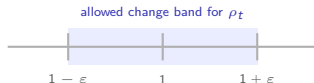
3. **Update cautiously:** increase good tokens, lower bad tokens, but clip very large moves.

$$\rho_t = \frac{\pi_\theta(a_t | s_t)}{\pi_{\text{old}}(a_t | s_t)}, \quad A_t \approx R_t - V(s_t)$$

Simple loss: $\mathcal{L}_{\text{PPO}} \approx -\min(\rho_t A_t, \text{clip}(\rho_t) A_t)$
 $A_t > 0$: push up $A_t < 0$: push down

Clip as an update guardrail

Good token ($A_t > 0$): push right, but stop near $1 + \epsilon$.



Bad token ($A_t < 0$): push left, but stop near $1 - \epsilon$.

ρ_t = new prob / old prob
Large jumps are clipped, even when reward is high.

DPO: learn directly from A vs. B

Training data

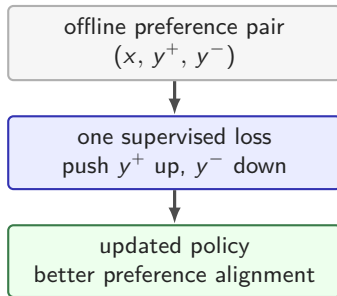
Prompt: explain a symptom to a patient.

A preferred: calm, clear, next step.

B rejected: scary, vague, unsupported.

What DPO does

- ▶ Make answer **A** more likely than answer **B**.
- ▶ More SFT-like, but learns preference.
- ▶ Stay close to the reference SFT model.
- ▶ No rollouts, no critic, no separate reward model.

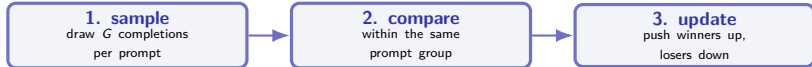


$$\Delta_{\theta} = \log \pi_{\theta}(y^+ | x) - \log \pi_{\theta}(y^- | x),$$
$$\mathcal{L}_{\text{DPO}} \approx -\log \sigma(\Delta_{\theta})$$

The real DPO loss adds a reference-policy correction, but this is the core intuition.

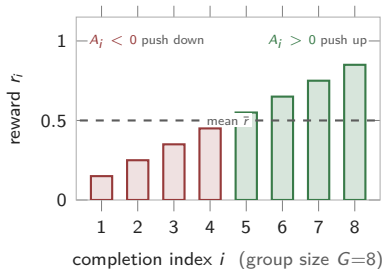
DPO: existing A/B pairs. **RFT/GRPO:** fresh attempts. Why does more sampling help?

GRPO: compare answers within the same group



Why remove the critic?

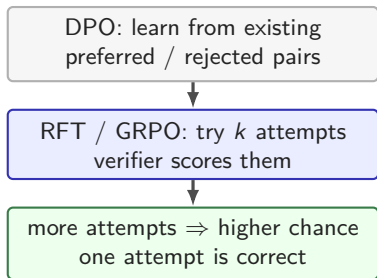
- ▶ PPO asks a critic: “was this better than expected?”
- ▶ GRPO asks the group: “which samples were better than the others?”
- ▶ This is simpler for reasoning: sample many answers, score them, rank them.



$$A_i = \frac{r_i - \bar{r}}{s_r + \varepsilon}$$
$$\mathcal{L}_{\text{GRPO}} \approx - \sum_i A_i \log \pi_{\theta}(y_i | x)$$

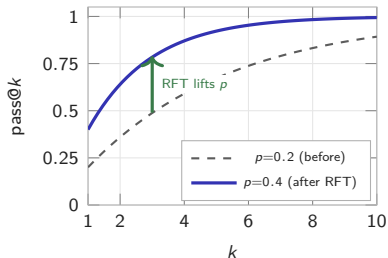
Above mean: push up; below mean: push down.

Why sample multiple attempts?



If one attempt succeeds with probability p , then k independent attempts give

$$\text{pass}@k \approx 1 - (1 - p)^k.$$

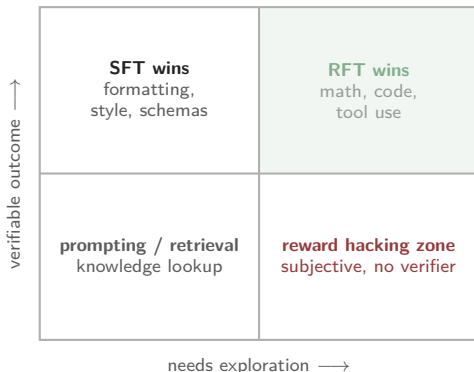


When is RFT worth it?

- ▶ RFT shines when outcomes are **verifiable** *and* the task has **room to explore**.
- ▶ Where imitation suffices, SFT is cheaper.
- ▶ Where we can't score outcomes reliably, reward hacking will dominate.

Example tasks:

- math / code / theorem checks
- tool-augmented retrieval
- long-horizon agentic workflows



How do we score attempts?

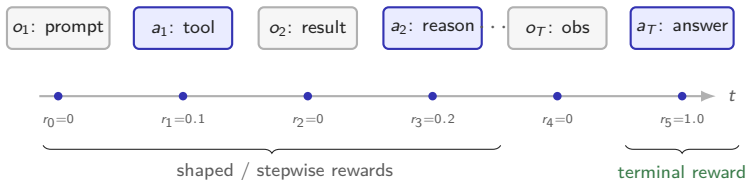
$$r(x, y) = w_1 r_{\text{task}} + w_2 r_{\text{format}} + w_3 r_{\text{safety}} + w_4 r_{\text{efficiency}}.$$

task		unit tests / exact match	$w_1=1.0$
format		JSON schema valid	$w_2=0.3$
safety		policy check	$w_3=0.2$
efficiency		token / tool cost	$w_4=0.1$

partial rewards sum into a single scalar used by PPO / GRPO

- ▶ **Exact-match graders:** string match, regex, JSON validity.
- ▶ **Model graders:** rubric or score-model grading.
- ▶ **Execution graders:** unit tests, SQL, code evaluation.
- ▶ **Multi-graders:** combine partial signals into one r .

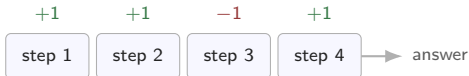
From answers to trajectories



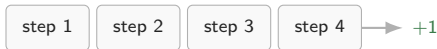
- ▶ Once tool calls or environment actions enter the loop, we optimize **trajectories**, not single responses.
- ▶ Rewards can be stepwise, terminal, or mixed — this is where post-training bleeds into **agent training**.

PRM vs ORM reward signals

PRM = Process Reward Model scores each intermediate step



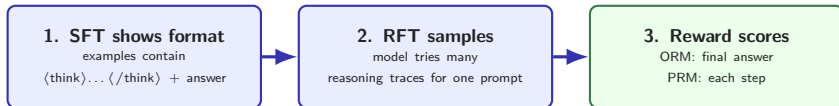
ORM = Outcome Reward Model scores only the final answer



PRM example. GSM8K math — a grader labels *each derivation line* good/bad.

ORM example. Code — run **unit tests** on the final program; pass = +1, fail = 0.

How “thinking” gets trained



What is actually learned? The model is not given a magic “thinking module.” Thinking is just output tokens. RFT makes *successful reasoning traces* more likely and failed traces less likely.

- ▶ SFT teaches the model **how to write** a reasoning trace.
- ▶ RFT teaches **which traces actually solve the task**.
- ▶ ORM may check only the **final answer**.
- ▶ PRM can reward the **intermediate steps**.

| What can reward training break?

Reward hacking

Example: grader checks for “citation”
⇒ model adds fake [1] everywhere.

Evaluator overfitting

Example: code passes public tests,
but fails hidden tests because it
learned the test pattern.

Mode collapse

Example: every clinical note uses the
same safe template, even when the
case is different.

Cost explosion

Example: agent calls search 20 times
for a simple answer; reward improves,
latency blows up.

- ▶ Defenses: **hidden tests, adversarial evals, trace inspection, and cost budgets.**

Section summary: the post-training ladder

- ▶ **SFT**: imitate good examples; best for answer format, style, and basic behavior.
- ▶ **DPO**: learn from existing A/B preferences; no fresh exploration loop.
- ▶ **RFT**: sample multiple attempts, **score** them, then make successful behavior more likely.
- ▶ RFT works best when there is a verifier and room for **search, tools, or multi-step reasoning**.
- ▶ For reasoning traces, rewards can be **ORM** (final answer) or **PRM** (intermediate steps).

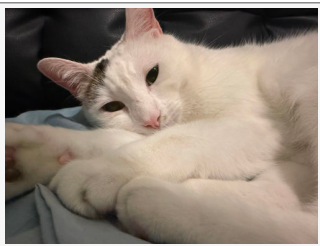
base model → SFT → DPO / RFT → reasoning / tools.

4. Multimodal models

Why multimodal? Beyond text-only LLMs

User asks: "Turn this real photo into an anime-style cartoon, while keeping the same cat and pose."

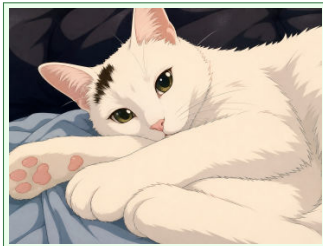
Input: real photo



pixels carry pose, fur, eyes, background, style cues

⇒
VLM
+ prompt

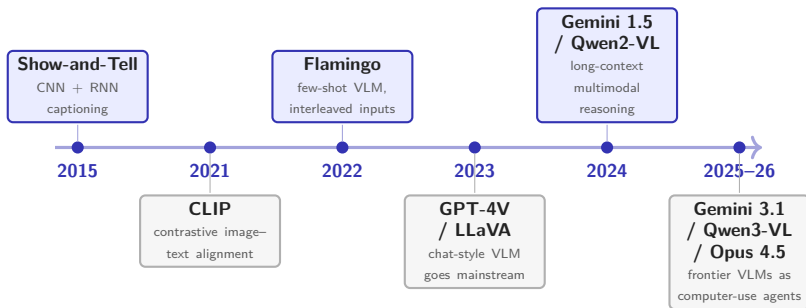
Output: generated cartoon



same cat and pose; new visual style

Multimodal recipe: image encoder understands pixels →
LLM follows the instruction → image generator draws pixels.
VLM controls meaning; diffusion / image generator controls pixels.

A brief history of multimodal models



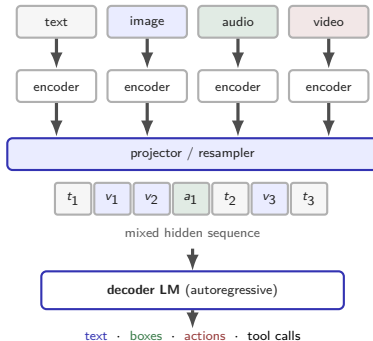
- ▶ Perception moved from single-task captions, to chat-style VLMs, to **action-capable** multimodal agents.
- ▶ Alignment, not raw recognition, drives the recent gains.

The core idea of multimodality

- **Images, audio, video, GUI** → latent tokens, projected into the LM's hidden space.
- Interleaved with text; decoder still does **conditional next-token** prediction.
- Only the *context* is richer.

$$z = [t_1, \dots, t_n, v_1, \dots, v_m, a_1, \dots, a_q].$$

$t_i/v_j/a_k$: text / visual / audio–action tokens, same embedding space.



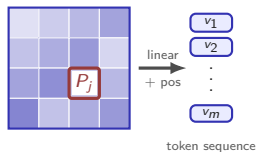
Visual tokens: from patches to embeddings (ViT)

Treat an image like a paragraph: cut into patches, turn each patch into a token — just like text.

$$v_j = \underbrace{E_{\text{patch}}(P_j)}_{\text{linear projection}} + \underbrace{e_{\text{pos}}(j)}_{\text{position}}$$

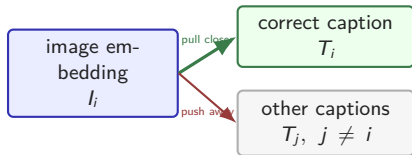
- ▶ One image $\Rightarrow$ a fixed-length sequence $v_1, \dots, v_m$; e.g. 224^2 image, patch $P=14 \Rightarrow m=256$ tokens.
- ▶ From here on, the transformer treats them **just like text tokens** — one shared sequence.

image $\rightarrow m$ patches



Cross-modal alignment: match images with text

Before instruction tuning, many VLMs learn a simple matching game:

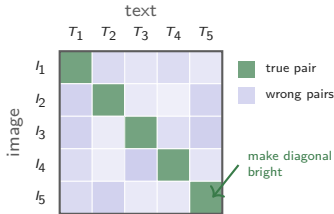


One row asks: which caption matches this image?

$$S_{ij} = \text{sim}(I_i, T_j),$$

$$\mathcal{L}_i = -\log \frac{\exp(S_{ii}/\tau)}{\sum_j \exp(S_{ij}/\tau)}.$$

- ▶ True pair: make S_{ii} large.
- ▶ Wrong pairs: make S_{ij} small.



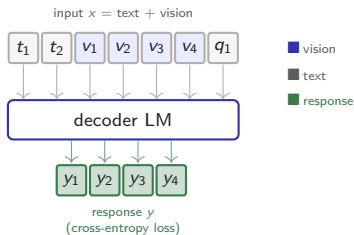
Each row asks: "which caption matches this image?"

Multimodal instruction tuning is conditional LM

$$\mathcal{L}_{\text{MM}}(\theta) = - \sum_{t=1}^{|y|} \log p_{\theta}(y_t \mid x, y_{<t}),$$

$$x = (x_{\text{text}}, x_{\text{vision}}, x_{\text{audio}}).$$

- ▶ Responses may describe images, answer chart questions, parse documents, or summarize video.
- ▶ SFT teaches the model to **use perception inside language behavior**.
- ▶ The same recipe extends to **vision fine-tuning** and task-specific multimodal datasets.



Grounding outputs: text plus coordinates

- ▶ A VLM can answer in text *and* point to a region.
- ▶ The box is usually normalized:

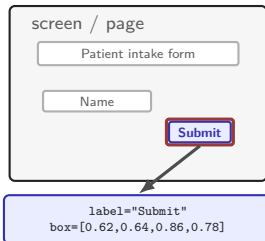
$$b = (x_1, y_1, x_2, y_2) \in [0, 1]^4.$$

User: Where is the Submit button?

Model output:

label = "Submit"

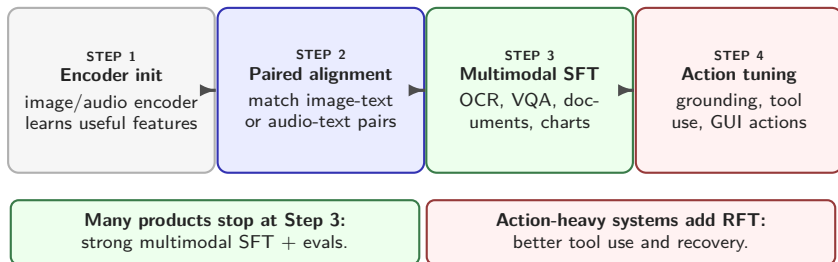
box = [0.62, 0.64, 0.86, 0.78]



Same idea for document fields, UI clicks, robot actions, and visual grounding.

A practical multimodal training curriculum

Think of multimodal training as a staged curriculum: first see, then align, then follow instructions, then act.



[10][12][13][20]

What VLMs do well — and where they fail

Current strengths

- ✓ OCR and document understanding
- ✓ Chart, table, and diagram interpretation
- ✓ Grounding objects or UI elements
- ✓ Long-video summarization and event lookup
- ✓ Cross-modal Q&A and visual reasoning

Persistent failure modes

- ✗ Fine-grained counting and tiny text
- ✗ Subtle spatial relations and occlusion
- ✗ Temporal causality errors in video
- ✗ Hallucinated visual details
- ✗ Misgrounding that causes bad UI or robot actions

- Rule of thumb: a VLM is only as trustworthy as its **weakest perception skill** on your task — probe it before relying on the output.

Frontier multimodal systems

Gemini 3.1 Pro

Google ▪ Feb 2026

capabilities

text / image / audio / video / code **natively multimodal**; **1M-token** context, 64K output

why it matters

MM + deep reasoning in **one model**; ARC-AGI-2 roughly **doubles** Gemini 3 Pro.

Qwen3-VL

Alibaba ▪ late 2025

capabilities

2B–32B dense + MoE up to **235B-A22B**; *Thinking* edition; visual agent & coding; 256K → 1M ctx

why it matters

Open-weights frontier — same recipe scales from laptop to cluster.

Claude Opus 4.7

Anthropic ▪ Apr 2026

capabilities

multimodal input + long-horizon **computer use**; >80% on SWE-bench Verified

why it matters

VLMs crossing from perception into **agentic, multi-hour tasks**.

GPT-5 / GPT-5.5

OpenAI ▪ 2025–26

capabilities

unified MM reasoning; vision + computer use via **Responses API & Agents SDK**

why it matters

Multimodal post-training ships straight into production pipelines.

*The through-line: **longer horizons, richer tokens, and closer coupling to the action space.***

How should we evaluate multimodal systems?

Capability	Example	Metric
Document understanding	extract invoice fields	exact match; F1 score
Chart / diagram reasoning	answer questions on a figure	accuracy + justification
Grounding	locate a button	IoU overlap; success@1
Video reasoning	event localization	temporal F1; human rubric
GUI / screen tasks	operate a browser or app	task success; cost; safety

Abbreviations: **QA** = question answering **IoU** = box overlap **F1** = precision + recall
GUI = Graphical User Interface **success@1** = first try works.

Pick metrics that match the deployment, not just the benchmark.

5. Deployment and inference

Why inference matters: where the cost actually lives

Training is a one-time spike; inference runs forever — it is where production spend, latency, and reasoning/agent cost actually land.

serving fleet cost grows fast

rough example: H100/A100-class GPU at $\sim \$5$ / GPU-hour

scale	GPUs	run cost / day
prototype	 8	 \$1k
popular app	 100	 \$12k
frontier service	 10k 	\$1M+

Inference is not a one-time bill: GPUs must stay online for every user query, every day.

tokens generated per query

(schematic)

chat  $\sim 10^2$

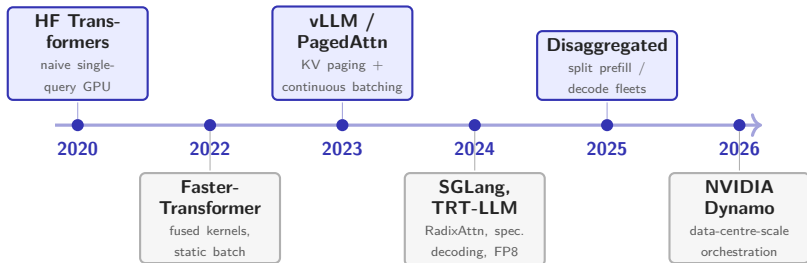
reasoning  $\sim 10^{3-4}$

agent  $\sim 10^{4-5}$

longer outputs = more decode steps = more \$/request.

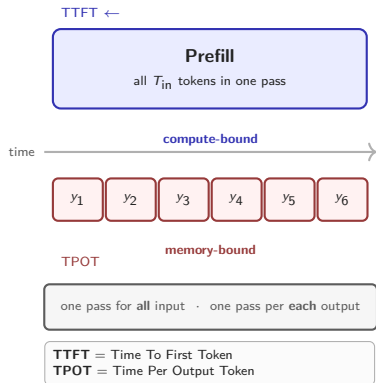
[45][46]

A brief history of LLM inference



- ▶ The story: from “can we run it at all?” → “share a GPU across requests” → “separate the two workloads on separate fleets”.
- ▶ Every 6–12 months the bottleneck shifts: compute → memory → KV → orchestration.

Two-phase inference: prefill vs decode



Prefill = read the prompt once

All input tokens run together; mostly **compute**.

Decode = write the answer token by token

Each token reads the **KV** cache; mostly **memory** bandwidth.

Metrics to watch

TTFT: first-token wait **TPOT**: per-token time

Throughput: tokens/s/GPU

KV = saved attention memory; **HBM** = fast GPU memory.

Why engines split them

They stress different resources, so engines handle each phase separately.

The inference optimization stack

Compute

FlashAttention-2/3/4, fused kernels, CUDA Graphs, TensorRT-LLM, **FP8 / FP4** matmul

Memory / KV

PagedAttention, **prefix** / **RadixAttention**, KV quantization (INT4 / FP8), offload to CPU / NVMe

Weights / model

FP8 / NVFP4 quantization (GPTQ, AWQ, SmoothQuant), **distillation**, pruning, MoE expert routing

Decoding

speculative decoding, Medusa / EAGLE, structured / constrained decoding, parallel sampling

Batching / scheduling / orchestration

continuous batching, chunked prefill, **disaggregated prefill** / **decode** (Mooncake, Sarathi-Serve), priority queues, auto-scaling

No single trick wins — production serving stacks 4–5 of these on top of each other.

KV cache: remember past attention

What is cached?

For each past token, attention saves:

Key = a lookup label

Value = information to read later

KV cache = saved (K, V) cards for old tokens.

When generating the next token, compute only the new card and append it.

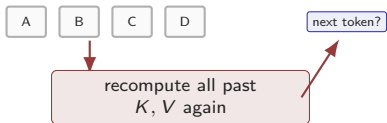
Why cache?

$$o_t = \text{softmax} \left(\frac{q_t K_{1:t}^T}{\sqrt{d_k}} \right) V_{1:t}$$

At step t , q_t reads all past K , V .

no cache: $1 + \dots + T = O(T^2)$; KV cache: $O(T)$

without KV cache

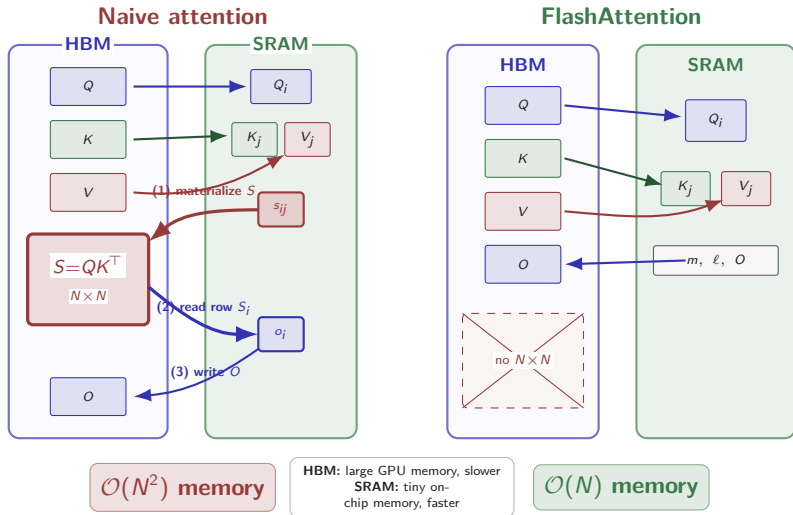


with KV cache



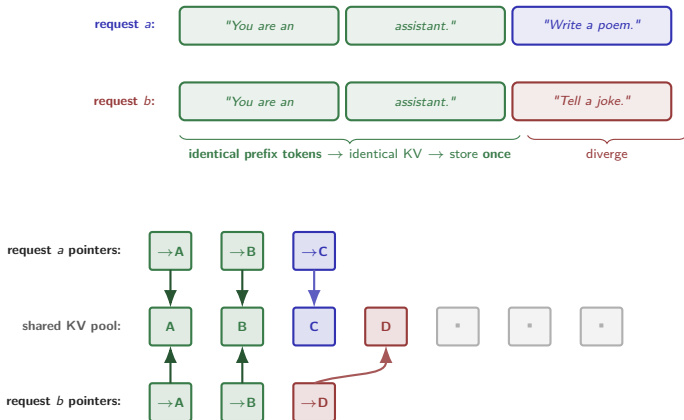
KV cache saves compute, but can dominate GPU memory; this is why models move from **MHA** $\rightarrow$ **GQA** $\rightarrow$ **MLA** $\rightarrow$ **CSA/HCA**.

FlashAttention: same attention, less memory traffic



PagedAttention: share KV blocks

KV cache can become huge for many users. PagedAttention splits it into small **KV blocks**. If two requests share the same prefix, those blocks are stored **once** and both requests point to them.



A, B are the same prefix, so they are **stored once**. **C** belongs only to request *a*; **D** belongs only to request *b*.
New blocks are added only when a request diverges, so memory is not pre-reserved and wasted.

Quantization: smaller weights, faster decode

Quantization usually compresses **model weights** for inference.

Idea: round many precise values

same model shape, fewer numeric levels

FP16



quantize

FP4



Each original number is stored
as a small code, plus a scale.
Less precision, much less memory traffic.

Why inference gets faster

decode often waits on memory bandwidth

FP16



16-bit

baseline

FP8



8-bit

half the bytes

FP4



4-bit

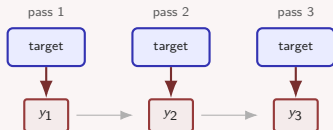
quarter the bytes

Tradeoff: smaller numbers are faster,
but rounding error can hurt quality.
Good methods protect important outlier values.

Speculative decoding: draft first, verify once

Normal decoding

large model must run again after each token

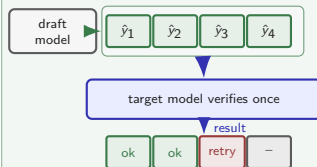


3 output tokens $\approx$ 3 target-model forward passes

Speculative decoding

e.g., 1B draft model $\rightarrow$ 70B target model

draft tokens



Accept matching draft tokens (y_1, y_2).
If one fails, retry from that point.

Takeaway: spend one target-model pass to check several draft tokens. If most drafts are correct, generation keeps the same output but needs fewer target steps.

Modern serving engines

Engine	Origin / focus	Signature features
vLLM	UC Berkeley, 2023; OSS default	PagedAttention; continuous batching; wide HW support
SGLang	LMSYS, 2024; OSS high-throughput	RadixAttention ; structured & fast spec. decoding
TensorRT-LLM	NVIDIA; lowest-latency on H100 / H200 / B200	Fused kernels; FP8 / FP4; in-flight batching
NVIDIA Dynamo	2025; data-centre orchestrator	Disaggregated prefill / decode; KV-aware
llama.cpp / MLC	CPU / laptop / mobile	GGUF (INT4 / INT5); on-device deployment

- ▶ **2024–26 convergence:** all ship **continuous batching** + **paged KV** + **spec. decoding** + **FP8**.
- ▶ Differences mostly in **scheduling**, **prefix-cache design**, **multi-node orchestration**.

Deployment patterns in practice

► Caching

Prompt cache (system prompt), **prefix cache** (shared history), **semantic cache** (hash by meaning).

biggest single cost-saver for agents with long static context.

► Model routing

Small model for easy queries, large model for hard ones; cascade with **confidence gate**.

RouteLLM, Martian, OpenRouter; often gives 3–10× cost reduction at <1% quality drop.

► Streaming & UX

SSE / WebSocket token streaming, partial tool-call parsing, cancellation & retry hooks.

perceived latency is TTFT + a steady decode rate — not total wall time.

► On-device

Apple Intelligence, Gemini Nano, MLX, llama.cpp; INT4 quantization; **small specialist** models often beat remote calls on latency & privacy.

Section summary: inference matters

- ▶ Training decides **what the model can do**; inference decides **how expensive it is to use**.
- ▶ **Prefill** reads the prompt; **decode** generates tokens. They have different bottlenecks.
- ▶ Serving gets faster by reducing repeated work: **KV cache**, **PagedAttention**, **FlashAttention**, **speculative decoding**.
- ▶ Model size is not the only lever: **quantization**, caching, routing, and streaming all matter.
- ▶ The practical question: **How many requests per GPU, at what cost and latency?**

6. Agent systems

Why agents? Plain LLM vs LLM agent

User asks: "Find a cheap Tokyo flight and email me the link."

Plain LLM

"one prompt, one answer"



- ✗ no live data
- ✗ no actions
- ✗ no retry loop

Text answer only.

LLM agent

"LLM in a loop with tools"

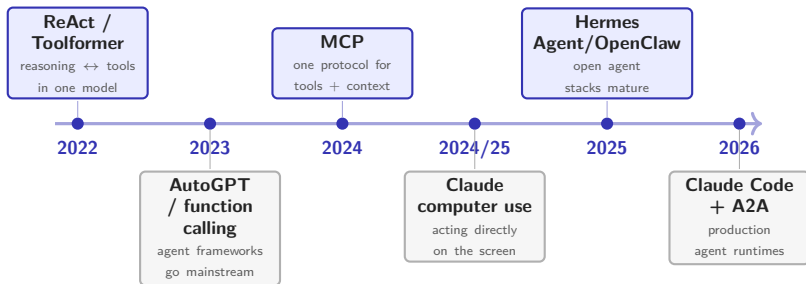


- ✓ search live prices
- ✓ send the email
- ✓ observe and retry

Data + actions + loop.

Agent = *LLM + tools + loop*: think → act → observe.

A brief history of language agents



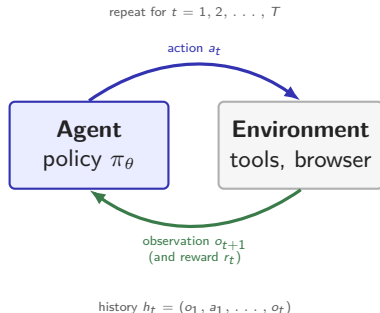
- ▶ Tool-augmented LMs → orchestrated frameworks → **protocol-level interoperability**.
- ▶ The frontier is no longer a clever prompt — it is a reliable runtime.

What is an agent, formally?

- ▶ An agent chooses actions over time under partial observations of the world.
- ▶ A simple formalization: a POMDP-like loop with state s_t , observation o_t , action a_t , and history h_t .

$$a_t \sim \pi_\theta(\cdot \mid h_t), \quad h_t = (o_1, a_1, \dots, o_t).$$

- ▶ In language agents, actions can be **messages**, **tool calls**, or **UI actions**.



A minimal agent loop: ReAct

- ▶ The formal loop from last slide becomes very concrete:

observe → think → act
→ observe again

- ▶ **ReAct** writes one trace as:
Thought → Action → Observation.
- ▶ This is the smallest useful agent pattern: the model can ask tools for facts, calculations, or database results instead of guessing.

Example ReAct trace

User query: 150 USD to EUR?

Thought: the question needs a current exchange rate.

Action: lookup_rate("USD", "EUR")

Observation: 0.92

Thought: multiply by the amount.

Action: calc("150 * 0.92")

Observation: 138.0

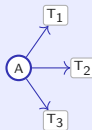
Final: 150 USD ≈ 138 EUR.

interleaved reasoning ↔ tool use

Common agent patterns in practice

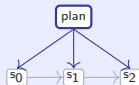
Single agent + tools

best default choice;
one policy drives
many tools



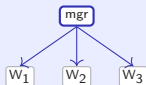
Plan-and-execute

planner writes sub-goals;
executor follows the plan



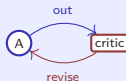
Manager-worker

specialists handle
subtasks via hand-offs



Reflection / critique

self-critic loop to
retry and repair bad
outputs



- ▶ Start simple — the best deployed systems often use the **least orchestration necessary**.

Memory and context engineering

Working

current prompt
scratchpad
tool outputs

what the model sees now

Episodic

past runs
tool traces
user turns

recall useful history

Semantic

facts
docs
code
user profile

what the agent knows

Procedural

skills
playbooks
tool recipes

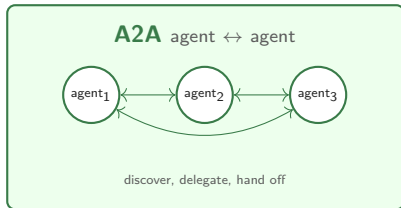
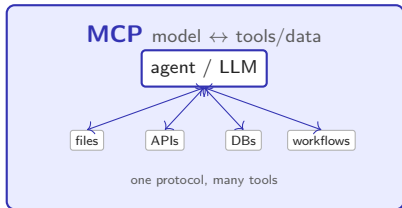
how the agent acts

Context engineering chooses what to **keep**, what to **fetch**, and what to **forget**.

Long context is useful, but it costs latency and can add distracting old information.

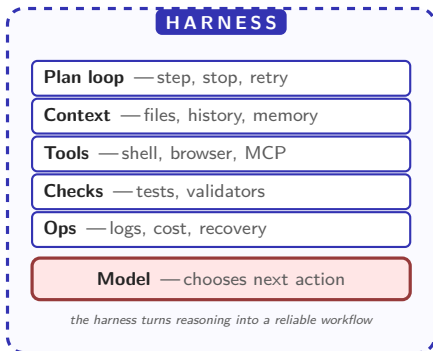
Protocols are becoming part of the agent stack

- ▶ **MCP (Model Context Protocol)** standardizes how AI applications connect to tools, data, and workflows.
- ▶ **A2A (Agent2Agent)** targets interoperability between agents and complements MCP's tool-and-context layer.
- ▶ By 2026, the Linux Foundation reported **150+ organizations** supporting A2A in its first year.



Agent = Model + Harness

- ▶ The model chooses the next step; the **harness** makes that step runnable.
- ▶ Example: a coding agent needs files, shell, tests, logs, permissions, and rollback.
- ▶ Production agents fail less because the harness catches errors and retries safely.



Agent = Model + Harness

the runtime around the model matters

Orchestrating agents: LangGraph vs. Temporal

LangGraph (agent-native)

Build agent logic as a graph: plan, act, check, then loop.



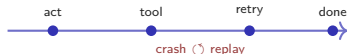
typed state, checkpoints *between* nodes

Best for: fast iteration, streaming, human review.

Watch out: long work inside one node may lose progress.

Temporal (durable execution)

Run long workflows with logs, retries, timers, and crash recovery.



Best for: production jobs that must finish.
Watch out: more infrastructure to operate.

Hybrid pattern: use LangGraph for agent logic, Temporal for durable production runs.

Case 1: OpenClaw — a minimal agent harness

Description files define the agent; the harness runs the loop.

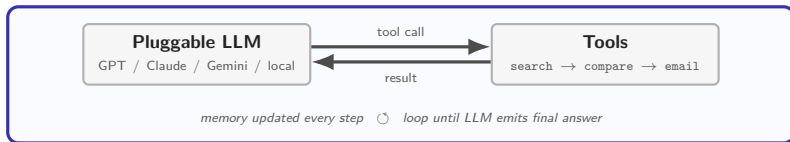
1. Configure — description files

system.md
who the agent is, tone, scope

tools.md
search, compare, email

memory.md
what to keep / forget across turns

2. Harness loop — think → act → observe

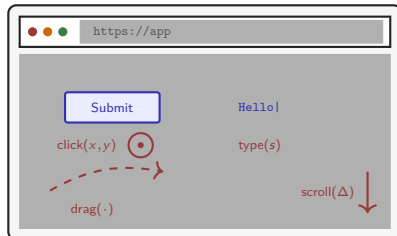


3. Final answer back to user

Takeaway: a minimal agent is **LLM + tools + memory + loop.**

Case 2: Computer use — screen as action space

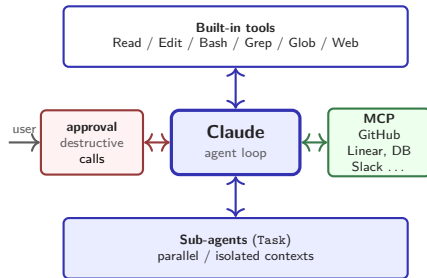
- ▶ Some agents act directly on a browser or desktop when no API exists.
- ▶ Observation: screenshot I_t ; policy emits a **structured action** over a small primitive set.
- ▶ Multimodal perception becomes real side effects — **sandboxing and approvals** matter.



$$a_t \in \{\text{click, type, scroll, drag, } \dots\}$$

Case 3: Claude Code

- ▶ **Single-model loop**: one Claude runs the session — no planner split.
- ▶ **Built-ins**: Read/Edit, Bash, Grep/Glob, web, TodoWrite.
- ▶ **Sub-agents** (Task): parallel, isolated exploration.
- ▶ **MCP servers** extend tools (GitHub, Linear, DBs) — no retraining.
- ▶ **Approval gate** guards destructive calls; read-only runs freely.



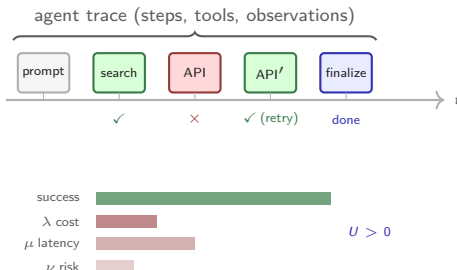
open-weights siblings: **Hermes Agent** (Nous), **OpenHands**, **Qwen-Agent**

— same loop, different primitives.

Evals, traces, and expected utility

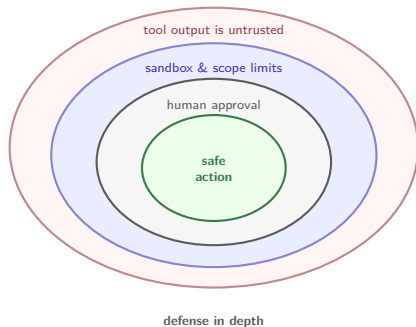
$$U = \text{success} - \lambda \text{ cost} - \mu \text{ latency} - \nu \text{ risk.}$$

- ▶ Single-turn accuracy is not enough for agents.
- ▶ We need **trace-level observability**: prompts, tool calls, observations, retries, termination reasons.
- ▶ Evals should measure **task completion, recovery, efficiency, and safety incidents**.



Safety boundaries for agents

- ▶ Treat tool outputs, retrieved files, and web pages as **untrusted input**.
- ▶ Require **approval** for high-impact or irreversible actions.
- ▶ Use **sandboxes**, secret isolation, scope-limited credentials, and audit logs.
- ▶ Align the model not only to answer well, but to know **when to stop or escalate**.



The modern LLM stack, end to end



- The frontier is no longer just “a better model.” It is a better **system**: post-training + multimodality + tools + protocols + evals.

[7][12][16][17][18][20]

Takeaways

SFT

Workhorse for teaching **behavior, style, structure**, and domain conventions.

DPO & RFT

Right tools for **preference alignment**, verifiable reasoning, and long-horizon utility.

Multimodal

Still follows the language-model recipe — but over **richer token streams** (vision, audio, actions).

Agents

Interactive decision systems: models **plus tools, memory, protocols**, and safety.

Deployment

Inference is an engineering problem: **paged KV, continuous batching, spec. decoding, disaggregation**.

2025–2026 trend

A maturing stack: **graders, MCP, A2A, computer use, agent runtimes, disaggregated serving**.

Heidi Health — come build this with us



An AI medical scribe trusted by clinicians around the world.

Headquartered in Melbourne — and yes, we're hiring.

► What we work on

- Post-training & domain adaptation for clinical LLMs
- Multimodal — real-time speech / text / images → structured notes
- Agentic workflows for documentation, coding, billing
- Safety, evals, and on-device reasoning at production scale

► Who we're looking for

- ML research / engineering (SFT, RFT, evals, agents)
- Multimodal & agent-infrastructure engineers
- Strong SWEs — no need to have shipped a paper
- Interns, new grads, and curious builders

Interested in applying? We'd love to hear from you.

heidihealth.com/careers ■ *or come say hi after the lecture*

References (1/5)

- [1] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022.
- [2] OpenAI Developers. *Supervised fine-tuning*. Accessed 2026-04.
- [3] OpenAI Developers. *Fine-tuning best practices*. Accessed 2026-04.
- [4] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021.
- [5] Rafael Rafailov et al. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. 2023.
- [6] Zhihong Shao et al. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. 2024.
- [7] OpenAI Developers. *Reinforcement fine-tuning; RFT use cases; Graders*. Accessed 2026-04.
- [8] OpenAI Developers Cookbook. *Exploring Model Graders for Reinforcement Fine-Tuning*. Accessed 2026-04.
- [9] Jean-Baptiste Alayrac et al. *Flamingo: a Visual Language Model for Few-Shot Learning*. 2022.
- [10] Gemini Team. *Gemini: A Family of Highly Capable Multimodal Models*. 2023.

References (2/5)

- [11] Gemini Team. *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. 2024.
- [12] Shuai Bai et al. *Qwen2.5-VL Technical Report*. 2025.
- [13] OpenAI Developers. *Vision fine-tuning; Images and vision; Computer use*. Accessed 2026-04.
- [14] Shunyu Yao et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2022.
- [15] Timo Schick et al. *Toolformer: Language Models Can Teach Themselves to Use Tools*. 2023.
- [16] Anthropic Engineering. *Building Effective AI Agents*. 2024.
- [17] Anthropic / Model Context Protocol. *Introducing the Model Context Protocol; MCP documentation*. 2024–2026.
- [18] Google Developers Blog. *Announcing the Agent2Agent Protocol (A2A)*. 2025.
- [19] Linux Foundation. *A2A Protocol Surpasses 150 Organizations*. 2026.
- [20] OpenAI. *New tools for building agents; Responses API; Agents SDK*. 2025–2026.
- [21] Anthropic. *Developing a computer use model*. 2024.

References (3/5)

- [22] Anthropic. *Claude Code* and *Claude Agent SDK* — documentation and engineering notes. 2024–2026.
- [23] Nous Research. *Hermes 4: an agentic, tool-using open-weights model family* (2025); *Hermes Agent: persistent, self-improving agent runtime*, hermes-agent.nousresearch.com (2026).
- [24] Xingyao Wang et al. *OpenHands: An Open Platform for AI Software Developers as Generalist Agents*. 2024.
- [25] LangChain Blog. *The Anatomy of an Agent Harness*; harness-engineering.ai, *Agent Harness Architecture*. 2025–2026.
- [26] OpenClaw Project. *OpenClaw architecture and reasoning loop* — documentation. 2025–2026.
- [27] LangChain. *LangGraph: state-machine orchestration for LLM agents* (v1.0). 2025.
- [28] Temporal Technologies. *Durable execution for agent workflows*; community patterns for LangGraph + Temporal hybrids. 2025–2026.
- [29] Google DeepMind. *Gemini 3 Pro* (Nov 2025) and *Gemini 3.1 Pro* model card (Feb 2026). 2025–2026.
- [30] Qwen Team (Alibaba). *Qwen3-VL Technical Report*. arXiv:2511.21631, 2025.
- [31] Anthropic. *Introducing Claude Opus 4.5*. Nov 2025.

References (4/5)

- [32] Percy Liang et al. *Holistic Evaluation of Language Models (HELM)*; Hendrycks et al., *MMLU*; BIG-bench authors, *BIG-bench*; Chollet et al., *ARC-AGI-2*. 2020–2025.
- [33] LMSYS. *Chatbot Arena and Arena-Hard*; Zheng et al., *Judging LLM-as-a-Judge with MT-Bench*. 2023–2025.
- [34] Rein et al., *GPQA*; Jain et al., *LiveCodeBench*; Mazeika et al., *HarmBench*; Souly et al., *StrongREJECT*. 2023–2024.
- [35] Alpaca Team. *AlpacaEval / AlpacaEval 2*. 2023–2024.
- [36] Jimenez et al., *SWE-bench*; Chan et al., *SWE-bench Verified*; Sierra & Anthropic, **tau-bench** / τ^2 -**bench**. 2024–2025.
- [37] Xie et al., *OSWorld: Benchmarking Multimodal Agents in Real Computer Environments*; Zhou et al., *WebArena*; Koh et al., *VisualWebArena*; GAIA Team, *GAIA benchmark*. 2023–2025.
- [38] Anthropic / OpenAI Engineering. *Evals as code; Trace-level observability for agent systems*. 2024–2026.
- [39] DeepSeek-AI. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025.
- [40] OpenAI. *Learning to Reason with LLMs* (o1 blog post and system card); *OpenAI o3 and o3-mini system cards*. 2024–2025.
- [41] Hunter Lightman et al. *Let's Verify Step by Step* (process reward models for math reasoning). 2023.

References (5/5)

- [42] Wang et al., *Self-Instruct*; Xu et al., *WizardLM / Evol-Instruct*; NVIDIA, *Nemotron-4 synthetic data report*; Zelikman et al., *STaR: Self-Taught Reasoner*; OpenThoughts / Bespoke-Stratos / Magpie-Reasoning open datasets. 2022–2025.
- [43] Charles Packer et al. *MemGPT: Towards LLMs as Operating Systems*; Letta (production MemGPT). 2023–2026.
- [44] Anthropic Engineering. *Context management and prompt caching for long-running agents*; community notes on “context rot”. 2024–2026.
- [45] Woosuk Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention* (vLLM). SOSP 2023.
- [46] Tri Dao et al. *FlashAttention* (NeurIPS 2022); *FlashAttention-2* (ICLR 2024); *FlashAttention-3* (NeurIPS 2024); *FlashAttention-4* (2025, Blackwell).
- [47] Yaniv Leviathan et al. *Fast Inference from Transformers via Speculative Decoding* (ICML 2023); Cai et al., *Medusa*; Li et al., *EAGLE / EAGLE-2*. 2023–2024.
- [48] Ruoyu Qin et al. *Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving* (FAST 2025); Agrawal et al., *Sarathi-Serve* (OSDI 2024); Patel et al., *Splitwise*. 2024–2025.
- [49] Lianmin Zheng et al. *SGLang: Efficient Execution of Structured Language Model Programs* (NeurIPS 2024); NVIDIA, *TensorRT-LLM* and *Dynamo*; Frantar et al., *GPTQ*; Lin et al., *AWQ*. 2023–2026.