BUSI70575 · OPTIONAL LECTURE 4

# Introducing Reasoning Models and Agentic AI

One decoder, four chapters:

**Transformers** • **Vanilla LLMs** • **Reasoning Models** • **Agentic AI**

Systematic Trading Strategies with Machine Learning Algorithms
Hachem Madmoun • Imperial College Business School

# Where we are going

1. The Transformer engine
2. Vanilla LLMs: building an assistant
3. Reasoning: teaching the model to think
4. Agentic AI: orchestrating models
   - The product: Sakana Fugu
   - TRINITY: evolve a tiny coordinator
   - The Conductor: write the whole workflow
   - The report card
5. Conclusion

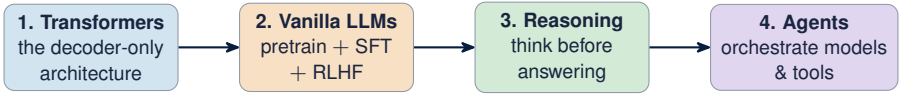
## One thread through the lecture

Every system in this talk is the **same decoder-only Transformer**. What changes is the **training signal** we apply and what we **wrap around it**.

## The arc

Understand the machine (**Transformers**) $\rightarrow$ build an assistant (**Vanilla LLMs**) $\rightarrow$ teach it to think, and **reward the thinking** (**Reasoning**) $\rightarrow$ let it command a team of models (**Agentic AI**).

# The story in one picture




Each chapter ends where the next begins. An architecture is **not yet an assistant**; an aligned assistant still **cannot reason** reliably, and **nobody rewards its thinking**; and even the best reasoner is **one model acting alone**, with one style of thinking. Agents close the loop.

SECTION 01

# The Transformer engine

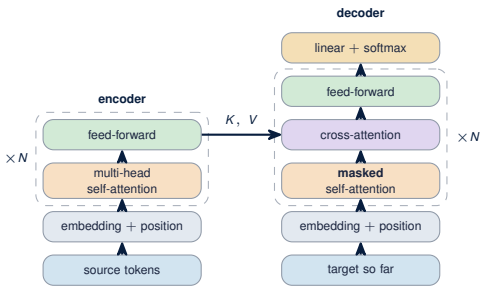
01

# The Transformer, in one breath

*Attention Is All You Need* [28] replaced recurrence with **self-attention**: every token looks at every other token in one parallel operation.

- No RNNs, no left-to-right bottleneck: the whole sequence is processed **in parallel**.
- Stacks of **multi-head attention** + **feed-forward** + **normalisation**, repeated $N$ times.
- Originally an **encoder–decoder** for translation; modern LLMs keep only the **decoder**.

Everything in this lecture is built on this one block.




The original encoder–decoder, simplified [28]. Decoder-only LLMs keep the right tower and drop cross-attention.

# Self-attention: a contextual embedding

Project every token $X^t$ into a **query** $q^t$, a **key** $k^t$ and a **value** $v^t$, then re-average the values by query–key similarity:

$$A(Q,K,V) \;=\; \mathrm{Softmax}\!\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \qquad A(q^t,K,V) \;=\; \sum_{t'=1}^{T} \alpha^{\langle t,t'\rangle}\, v^{t'}.$$

- $\dfrac{QK^\top}{\sqrt{d_k}}$: pairwise similarities (scaled).
- Softmax: similarities $\rightarrow$ attention weights $\alpha$.
- Multiply by $V$: a context-aware vector per token.

## Multi-head

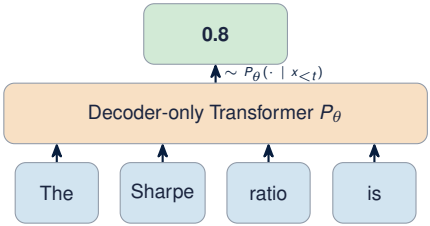
Do this $h$ times in parallel with different projections, then concatenate: each head captures a different notion of similarity.

# Decoder-only: generation as next-token prediction

Modern LLMs (GPT, Claude, Gemini, DeepSeek) keep only the **masked decoder** [21]: token $t$ attends only to positions $t' \leq t$. This makes the model **autoregressive**. A language model factorises a sequence by the chain rule and is trained with the **cross-entropy** of the next token:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log P_\theta(x_t \mid x_{<t}) \,.$$

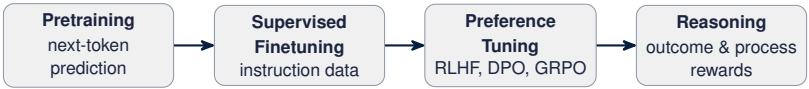
**Everything that follows is this same object with a different training signal.**




Greedy vs. sampling, temperature, top-$k$/top-$p$ [8] all shape this one distribution.

**Quiz Time:** `link`

# An engine is not an assistant




- The decoder gives us a machine that **continues text**. Nothing more, nothing less.
- Whether it becomes a helpful assistant, a careful mathematician, or an autonomous agent is decided entirely by the **training signals** we now apply, in stages.

## Bridge to Section 2

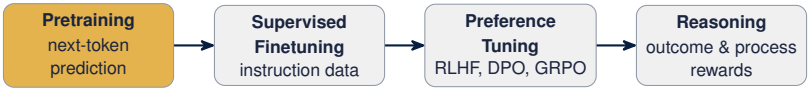
From here on, the architecture never changes. Every upgrade in this lecture (assistant, reasoner, orchestrator) is a change of **data and reward**, not of wiring. We start with the recipe that built the assistants you already use.

SECTION 02

# Vanilla LLMs: building an assistant

02

# The training pipeline of a modern LLM




- Modern LLMs are built in **stages**; each exists **because the previous one falls short**.
- **Pretraining** gives broad knowledge of language, code and the world.
- **Supervised finetuning** teaches it to follow instructions.
- **Preference tuning** aligns it with what humans actually prefer.
- **Reasoning training** (Section 3) teaches it to think before answering, and asks who **rewards** that thinking.

# Pretraining: a powerful text completer

- **Objective:** predict the next token, on trillions of tokens of web text, code, books and science [2].
- **Output:** a **base model** with broad knowledge, but **not an assistant**.
- At scale, abilities **emerge**: in-context (few-shot) learning appears with no gradient updates.
- **Chinchilla** [7]: for a compute budget, scale parameters and data **together** ($D \approx 20N$).

## A base model is not an assistant

Ask "what is the Sharpe ratio of . . . ?" and a base model may **continue** the text with more quiz questions, because online, questions are followed by questions.

## Two facts to carry forward

Knowledge is **frozen** at the cutoff and **hard to edit**. Fresh or private facts must reach the model through its **context** at inference time; in Section 4, agents will fetch them with **tools**.

# Supervised finetuning: learn the assistant format




Continue training on curated **(instruction, response)** pairs, with the **same next-token loss** [32].

- **Loss masking:** compute the loss on the **response** tokens only; the instruction is context.
- Same architecture, same objective, **different data**.
- **LoRA** [9] / **QLoRA** [6]: train a low-rank update $W = W_0 + \frac{\alpha}{r} BA$, $< 1\%$ of parameters, on one GPU.




## What SFT cannot do

It only **imitates** good answers; it adds **no negative signal** against bad behaviour the model already has.

# Preference tuning, step 1: learn a reward model




**Key insight:** for a human, *ranking* two answers is far cheaper and more consistent than *writing* the perfect one [20].

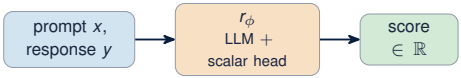
Collect preference triplets $(x, y_w, y_l)$, with $y_w$ preferred. Fit a **reward model** $r_\phi$ (an LLM with a **scalar head**) using the **Bradley–Terry** model [1]:

$$\mathbb{P}(y_w \succ y_l \mid x) = \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big).$$

Train it by maximum likelihood, i.e. minimise

$$\mathcal{L}_{\mathrm{RM}}(\phi) = -\,\mathbb{E}_{(x,y_w,y_l)}\Big[\log \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big)\Big].$$

**Remember this loss:** DPO will reuse it verbatim.




Trained on *pairs*: nudge $r_\phi(x, y_w)$ above $r_\phi(x, y_l)$. One scalar per answer, an LLM-sized judge of quality.

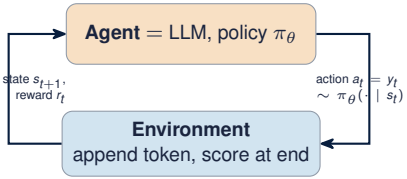
## Now what?

The reward $r_\phi$ is a **proxy** for human taste. Step 2: put it to work. Three ways, coming up.

# Step 2: reinforcement learning against $r_\phi$

With $r_\phi$ frozen, train the policy $\pi_\theta$ (our LLM) to **raise reward without drifting**:

$$\max_\theta \underbrace{\mathbb{E}_{x,\,y\sim\pi_\theta}[r_\phi(x,y)]}_{\text{maximise reward}} - \underbrace{\beta\, D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})}_{\text{stay near the SFT model}}.$$




Reward is **terminal**: 0 until the answer ends, then $r_\phi$.

Optimised by **policy gradient** (the log-prob trick), never by differentiating $r_\phi$.

## PPO [24]

Advantage = reward − a **critic** baseline $V_\psi$; clip each update, stay near $\pi_{\mathrm{ref}}$. **Four models in memory**: policy, reference, reward, critic. Powerful, delicate.

## GRPO [25]

**Drop the critic.** Sample a **group** of $G$ answers; the group is the baseline:

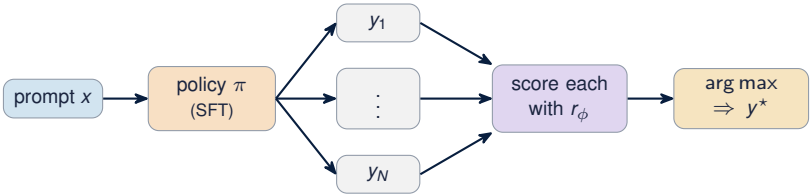
$$\hat{A}_i = \frac{r_i - \mathrm{mean}(r_1,\ldots,r_G)}{\mathrm{std}(r_1,\ldots,r_G)}.$$

**No critic**; and a rule-based scorer can even replace $r_\phi$. **Keep this: it returns in Sections 3 and 4.**

Objective & PPO: [20, 24]; group baseline: [25].

# Skip the RL: best-of-$N$ with the reward model

RL bakes the reward into the weights. But once we have $r_\phi$, we can also use it **at inference only**, with **no gradient step at all**: sample $N$ answers, score them, keep the best. This is **best-of-$N$** (a.k.a. reranking / rejection sampling) [4].




## Why it is attractive

No policy training, no PPO/GRPO machinery: the **same SFT model** plus a scorer. Quality climbs with $N$, and you can turn it on per query.

## The cost

$N\times$ inference on every query, and large $N$ **over-optimises the proxy**: you find the answer $r_\phi$ loves, not the one humans do. Reward hacking, now at test time.

# Skip the reward model too: DPO

**DPO** [22] asks: do we need a *separate* reward model at all? Solve the KL-regularised objective in closed form; its optimum is

$$\pi^\star(y \mid x) = \tfrac{1}{Z(x)}\, \pi_{\mathrm{ref}}(y \mid x)\, e^{r(x,y)/\beta} \;\Longrightarrow\; r(x,y) = \beta \log \frac{\pi^\star(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \beta \log Z(x).$$

Read backwards: every policy defines an **implicit reward**, so parametrise the reward *by the policy*

$$r_\theta(x,y) \;=\; \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)}$$

and drop $r_\theta$ into the **same Bradley–Terry loss** from step 1. The $\beta \log Z(x)$ term is identical for $y_w, y_l$ and **cancels**:

**Reward-model loss (step 1)**

$$-\mathbb{E}\big[\log \sigma\big(\, r_\phi(x,y_w) - r_\phi(x,y_l)\,\big)\big]$$

**DPO loss (same form)**

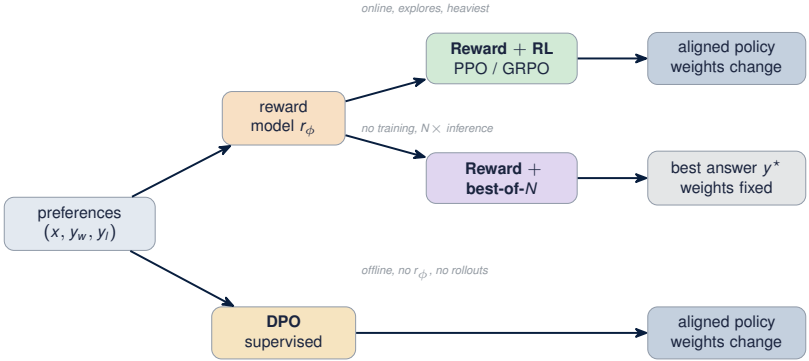
$$-\mathbb{E}\big[\log \sigma\big(\, r_\theta(x,y_w) - r_\theta(x,y_l)\,\big)\big]$$

**DPO is just the reward-model objective**, with the reward reparametrised as the policy's own log-ratio. Fitting the "reward" *is* training the policy: **one supervised step, no reward model, no rollouts**.

DPO and the reparametrisation: [22]; Bradley–Terry: [1].

# Three ways to use preferences

All start from the same preference data. They differ in **what they train** and **when they pay**.




## Bridge to Section 3

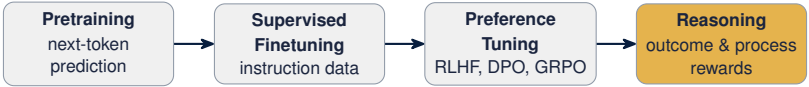
We now have an aligned assistant and, in GRPO, a cheap RL engine. Whether it learns to **reason** comes down to one question: **what reward do we feed it?** That question opens Section 3, and GRPO comes with us.

SECTION 03

# Reasoning: teaching the model to think

03

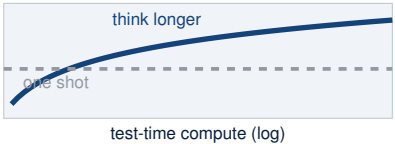
# Reasoning: a new scaling axis




Section 2's pipeline ends in a **helpful assistant**, not yet a **reasoner**; and until 2024, more capability meant one thing: **bigger models, more data**.

**OpenAI o1** (2024): spend compute *thinking* at inference, in a long internal chain-of-thought; large gains on **math and code**.

**DeepSeek-R1** (2025) [5]: an **open** reasoner at o1 level on math. **How you train one is this section's story.**




Accuracy keeps climbing as the model thinks longer [17].

Inference-time reasoning is a **new scaling axis**. Section 3 in three moves: **elicit** the thinking, **own** it, **judge** it. Finance will force the third.

Test-time scaling: OpenAI o1 (2024); [17]. DeepSeek-R1: [5].

# Chain-of-thought: just ask it to think

**Standard prompting**

Q: A baker made 36 muffins. He sold half in the morning, then a third of the rest in the afternoon. How many are left?

A: The answer is **6**. ×

**Chain-of-thought prompting**

Q: (same question)

A: Morning $36/2 = 18$ sold, 18 left. Afternoon $18/3 = 6$ sold. $18 - 6 = 12$. Answer: **12**. ✓

*Same model, same question; only the prompt changed.*

**The breakthrough** [31]: writing out **intermediate steps** unlocks multi-step problems, with **no architecture change**.

**Zero-shot CoT** [10]: just append *"Let's think step by step"*, no few-shot examples needed.

**Self-consistency** [30]: sample $N$ chains, take the **majority vote**; independent errors cancel.

## Result

CoT prompting alone lifts a small **0.6B** base model (Qwen3-0.6B) from 15.2% to 40.6% on **MATH-500**.

# Beyond CoT: a family of test-time tricks

Once CoT existed, the field branched into two themes: spend compute on **more paths** through the same model, or pull in **signals from outside** it.

## More paths through the model

**Self-consistency** [30]: sample $N$ chains, majority vote (last slide).

**Tree of Thoughts** [35]: search a **tree** of partial chains, score the nodes, **backtrack** from weak branches. Game of 24: CoT $\approx$ **4%**, ToT $\approx$ **74%**; a bad first move becomes recoverable.

## Signals from outside the model

**ReAct** [36]: interleave **Thought** $\rightarrow$ **Action (tool)** $\rightarrow$ **Observation**; every tool call is independently checkable. **Hold this loop: it is Section 4's agent.**

**Reflexion** [26] / **self-refinement** [14]: write a verbal **critique** of your own failure, retry with it in context: 91% pass@1 on HumanEval vs $\approx$ 80% raw.

Every trick above buys accuracy with **inference compute**, and none touches a weight. Time to look at the bill.

Taxonomy after Lahlou, *A Gentle Introduction to LLM Reasoning*, NETYS 2026 [11].

# Renting reasoning: the test-time bill

Stack **self-consistency** on top of CoT: sample $N$ chains, take the majority vote. Same 0.6B model, same benchmark, one scoreboard:

| Qwen3-0.6B on MATH-500 | accuracy | inference cost |
|---|---|---|
| base model, greedy | 15.2% | $1\times$ |
| + chain-of-thought | 40.6% | $\approx 8\times$ |
| + self-consistency, $N = 10$ | **52.0%** | $\approx 85\times$ |

At $N = 10$ the prompted **base** model even beats a reasoning-tuned 0.6B (48.2%).

## The catch

Every point is bought with **inference** compute, and the bill comes due on **every single query**. Prompting **rents** the reasoning.

## The alternative

Pay **once**, at training time, and **own** the skill: **amortisation**. One question stands in the way: **what reward teaches thinking?**

Numbers: Raschka, *Build a Reasoning Model (From Scratch)*, 2025.

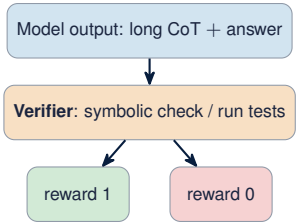
# Verifiable rewards: the engine of reasoning models

To **own** the skill, train it in. Could we just **SFT** on reasoning chains? Three obstacles: long chains are **impractical to write at scale**; SFT **caps** the model at human styles; and a **natural reward** sits right there: *did it solve the problem?*

So we return to **RL**, with one decisive change [4, 5]: replace Section 2's **learned, hackable** reward model with an **automatic verifier**.

- **Format** reward: matches `<think>...</think>` answer.
- **Accuracy** reward: answer matches ground truth; for code, **run the tests**.

One **outcome bit** per attempt: 1 if the answer verifies, 0 if not.




**Exact, automatic, free**: RL runs for millions of episodes with no human in the loop.

Outcome-supervised verifiers [4]; RL from verifiable rewards at scale [5].

# What counts as a verifier?

RL from verifiable rewards needs *some* verifier. They come in three flavours, and the differences decide everything downstream.

## Mathematical

Run code, evaluate an expression, run the unit tests, compare to ground truth: `sympy.simplify(a) == ref`.

$\Rightarrow$ **exact and cheap**. The engine of R1-style training.

## Claim

Check a factual statement against a **knowledge base** or retrieval index: "the P/B ratio uses book value" $\rightarrow$ true.

$\Rightarrow$ **noisy**, and bounded by **KB coverage**.

## Policy

Check a constraint on the output: length, format, safety, style: `len(out) <= 500`.

$\Rightarrow$ **cheap, partial**: it says nothing about correctness.

Real systems **chain** them: syntax $\rightarrow$ tests $\rightarrow$ safety, each layer catching a different error mode. **Remember the claim verifier**: it returns in Fin-PRM, handed a knowledge base that speaks finance.

Verifier taxonomy after Lahlou, NETYS 2026 [11].

# GRPO: drop the critic, let the group be the baseline

Which optimiser feeds on this bit? **PPO**'s last expensive leftover is the **value network** (critic): a second LLM-sized model that must predict reward from partial chains. **GRPO** [25], sketched in Section 2, removes it. For each prompt, sample a **group** of $G$ responses, score them with the verifier, and normalise **within the group**:

$$\hat{A}_i = \frac{r_i - \mathrm{mean}(r_1, \ldots, r_G)}{\mathrm{std}(r_1, \ldots, r_G)}.$$

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t}\min\left(\rho_{i,t}\hat{A}_i,\ \mathrm{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}_i\right)\right] - \beta\, D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})\,.$$

**Same skeleton as PPO** (maximise advantages, stay close), but the **group** replaces the critic and the **verifier** replaces the reward model.

**Keep this recipe in your pocket:** in Section 4, the very same GRPO trains a small model to orchestrate frontier LLMs.

GRPO [25]. On the running 0.6B example, 50 GRPO steps lift MATH-500 from 15.2 to **47.4** (Raschka, 2025).

# GRPO vs PPO: who computes the advantage?

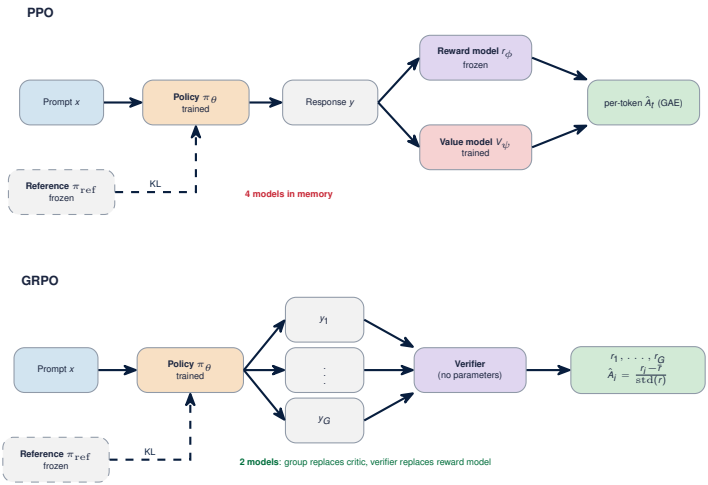
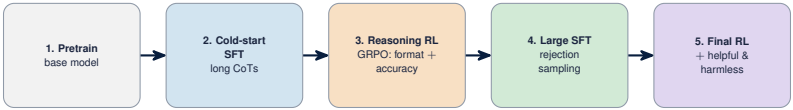



Figure after [25].

# The DeepSeek-R1 recipe, and distillation




- The **whole pipeline of this lecture reappears**: pretrain, SFT, RL, alignment, now with reasoning in the loop [5].
- **No human annotators** for the reasoning RL: the maths checks itself. **Exact, automatic, free**, end to end.
- Inside the `<think>` block, **reflection and backtracking emerge** during RL, never explicitly taught.

## Distillation

A small **student** is SFT'd on the **teacher's** full chains. A 32B student recovers most of a 671B teacher's maths ability; our 0.6B example reaches **45.0** on MATH-500 in $\sim$3h, no RL at all. **Hold that thought for Fin-PRM.**

Renting bought 52.0 at 85$\times$ per query; owning buys $\sim$47 at 1$\times$, forever. **Amortisation, delivered.**

R1 pipeline [5]; 0.6B numbers: Raschka, 2025.

# The outcome bit is blunt

**Exact, automatic, free**; and **blind**. The bit judges only the **destination**, never the **route**, and that blindness fails in two ways [27].

### Failure mode 1: a false positive, fully rewarded

**Q:** A 5-year bond issued in 2022; we are in 2026. Years to maturity?

**Chain:** "2022 + 5 = 2026, so it matures this year… hmm, maturity is 2027, so 1 year."

**Answer:** 1 year ✓ ⇒ outcome reward = 1, flawed step and all.

Reinforce enough of these and the model learns that **sloppy chains pay**, as long as the last line lands.

### Failure mode 2: no credit assignment

A 40-step chain earns **one bit at the very end**. Which step earned it, or lost it? The bit never says. File this away: it returns at full force in Section 4, when one bit must pay a whole **team**.

### What we actually want

A reward that reads the **process**: score **every step**, so credit lands on the step that earned it, and blame on the step that lost it. And all of this assumes the bit is *right*. Next: when it is not.

Process- vs outcome-based feedback: [27].

# What if the verifier is wrong?

Blind was structural; in practice the bit is also **noisy**: LLM judges are fooled by confident phrasing, and parsers reject correct answers in an unexpected format. Model it as a **noisy channel**: the true bit $r^\star \in \{0, 1\}$ is unobserved, and the observed $\tilde{r}$ flips it with rate $\rho_+$ (accepts a wrong answer) or $\rho_-$ (rejects a correct one).

## What noise does to GRPO

The noisy gradient is the true one, **shrunk**:

$$\nabla \tilde{\mathcal{L}}_{\mathrm{GRPO}} \;=\; (1 - \rho_+ - \rho_-)\, \nabla \mathcal{L}_{\mathrm{GRPO}}.$$

Ten percent noise each way and you train on **80% of the gradient**, stalling at a strictly worse optimum.

## The fix, two stages

Estimate $\hat{\rho}_+, \hat{\rho}_-$ on a held-out slice, then apply a **label-noise correction** [18] inside the GRPO advantage, rescaling to recover the gradient.

## Results

Up to $+6.7$**pp** on GSM8K and $+1.5$**pp** on APPS over uncorrected GRPO, with off-the-shelf reward models.

Noise can be measured and corrected, **when a bit exists**. In finance, strike three: often it does not.

El Mansouri, Izzati, Seddik, Lahlou. *Noise-corrected GRPO: From Noisy Rewards to Unbiased Gradients.* ICML 2026 [15].

# And in finance, there is often no bit at all

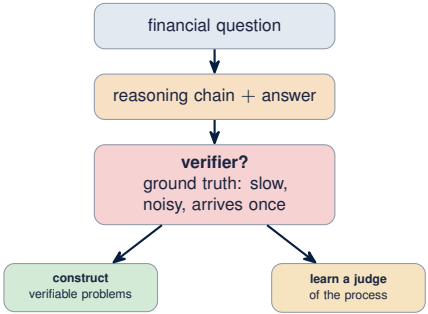
A blunt, **noisy** bit is still the *lucky* case. Math and code at least **have** one, computed for free: extract the boxed answer, run the tests. That is exactly why they dominate reasoning research.

In finance, the bit itself is often missing:

- Is this **investment thesis** correct? Ground truth arrives **slowly, noisily, once**.
- A chain can read beautifully and still **hallucinate a definition** or **violate a regulation**.

Two ways out:

- **Construct** verifiable problems: symbolic financial CoT benchmarks [33].
- **Learn a judge** of the reasoning itself: a **process reward model**, the reward the previous slide asked for. Our route for the rest of this section.



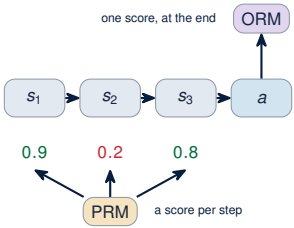

Constructing verifiable financial reasoning: [33].

# Process reward models: score the thinking

An **outcome reward model** (ORM) scores the final answer. A **process reward model** (PRM) scores **every step** of the chain [13, 27]:

$$r_t \;=\; R_\phi(s_t \mid x,\, s_{<t}), \qquad t = 1, \ldots, T.$$

- A **dense** signal: credit and blame land on the step that earned them.
- On hard competition math, process supervision **beats** outcome supervision [13].




The PRM catches the flawed $s_2$ that the outcome bit forgave.

## Where do step labels come from?

**Humans**: PRM800K pays experts to mark ∼800,000 math steps [13]; gold, but slow, costly, per-domain. **Monte Carlo**: define a step by its **consequences**: sample $N$ completions from it and count how many finish correctly, $\text{value}(s_t) \approx \frac{1}{N}\sum_i \mathbf{1}\{\text{completion } i \text{ correct}\}$ [29]. No humans: the outcome bit, **recycled** into step labels. **Keep this estimator in mind.**

# Step-level preferences without a judge: PORT

A PRM is one way to act on steps. **PORT** [12] shows another: skip the judge entirely, and run **Section 2's DPO** at the **step level** of a chain.

## The recipe

SFT on (prefix, next step); then DPO on triplets **(prefix, chosen step, rejected step)**. The scarce ingredient is the **rejected** step, so PORT manufactures it for free:

- **Digit corruption**: swap random digits in a correct step; plausible shape, wrong numbers.
- A **weak LLM** writes the rejected step.

**No human annotators, no reward model.**

## Results (Falcon2-11B)

$+8.5\%$ relative on GSM8K, $+18.7\%$ relative on AQuA; the gains **transfer** to non-math reasoning (ARC).

## The callback

The **Bradley–Terry** loss, in its **third** appearance: reward model, DPO, now per step.

Cheap and general, but the contrast is **synthetic**. Finance needs a judge that **knows finance**; enter Fin-PRM.

Lahlou, Abubaker, Hacid. *PORT: Preference Optimization on Reasoning Traces.* NAACL 2025 [12].

# Fin-PRM: a process reward model that speaks finance

## The gap

Existing PRMs are trained on **general and STEM** data (GSM8K, MATH, code) with generic criteria. Financial reasoning is **structured and symbolic**, bound by **regulation**, **factually unforgiving**, and **consequential**. A math judge does not speak finance.

## The paper

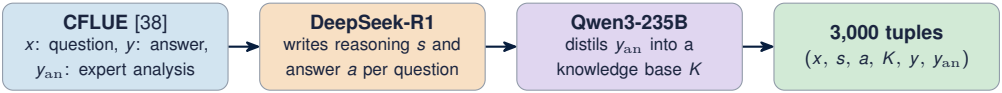
**Fin-PRM** [39]: the first **domain-specialised** process reward model for financial reasoning, from the team behind Alibaba's Qwen DianJin financial models.

## Three ideas to hold on to

1. A **two-level** reward: judge every **step**, and the **whole trajectory**.
2. Rewards **grounded in expert knowledge**, not just in logic.
3. One trained judge, **three jobs**: select data offline, shape RL online, rank answers at test time.

Zhu, Zhou, Jiang, Li, Guo, Chen, Zhang. Fin-PRM, arXiv:2508.15202 [39].

# Fin-PRM's data: give the judge something to grade




## The teacher is the reasoner we just built

The traces come from **DeepSeek-R1**. **Distillation**, the trick we held from the R1 slide, reappearing as **data synthesis**.

## Why a knowledge base $K$

Term–definition pairs ("P/B ratio: market price over book value"), extracted from the **expert analyses**: the **claim verifier** from the taxonomy, finally given a KB that speaks finance.

One tuple, four checks: **logic** ($s$ vs $y_{\text{an}}$), **facts** ($s$ vs $K$), **outcome** ($a$ vs $y$), **coverage** ($s{+}a$ vs $K$). These four checks become the **reward signals**, next.

Dataset construction: Fin-PRM [39]; CFLUE benchmark: [38].

# The step-level reward: three signals, one score

$$r_{\text{step}}(s_t) \;=\; \alpha\, r_{\text{imp}} \;+\; \beta\, r_{\text{qual}} \;+\; \gamma\, r_{\text{acc}} \qquad \text{(weights: softmax over the three signals; label} = 1 \text{ iff } r_{\text{step}} \geq 0.5\text{)}$$

## $r_{\text{imp}}$: consequences

From $s_t$, sample **8 completions** (Qwen2.5-7B-math); count how many reach the right answer: $r_{\text{imp}} = \#\text{correct}/8$. *Math-Shepherd's estimator* [29], verbatim.

## $r_{\text{qual}}$: LLM judge

Qwen3-235B scores **logical soundness**, **step correctness**, **progress to target**, each in $[0, 1]$; average the three.

## $r_{\text{acc}}$: the facts

$\frac{1}{2}\, r_{\text{proc}} + \frac{1}{2}\, r_{\text{know}}$: is the step **valid toward** $y$, and are its claims **verified in** $K$? Both binary. **The finance-specific piece.**

### Worked example (illustrative signals)

"Intermediary business is off-balance-sheet and generates fee income…" $r_{\text{imp}} = \frac{6}{8} = 0.75$; $r_{\text{qual}} = 0.85$; $r_{\text{acc}} = 1.0$ $\Rightarrow$ weights $\approx (0.30,\ 0.33,\ 0.38)$ $\Rightarrow$ $r_{\text{step}} \approx 0.88 \geq 0.5$ $\Rightarrow$ label $= 1$ ✓

*(The paper's case study scores this step* 0.769*; the sub-scores behind it are not published.)*

Step-level reward design: Fin-PRM [39].

# The trajectory-level reward: outcome + coverage

Step scores alone can be gamed: **every step polished, final answer wrong**. So each **whole trajectory** is scored too:

$$R_{\text{traj}} = \underbrace{r_{\text{out}}}_{\mathbf{1}\{a=y\}} + \eta \underbrace{r_{\text{cover}}}_{\frac{\#\ K\text{-terms used}}{\#\ \text{required}}}, \qquad \eta = 1.5, \qquad \text{label} = 1 \iff R_{\text{traj}} \geq 1.25.$$

**Pass ✓**

Correct answer, moderate coverage:
$r_{\text{out}} = 1$, $r_{\text{cover}} = 0.6 \Rightarrow R_{\text{traj}} = 1.9 \geq 1.25$.

**Fail ×**

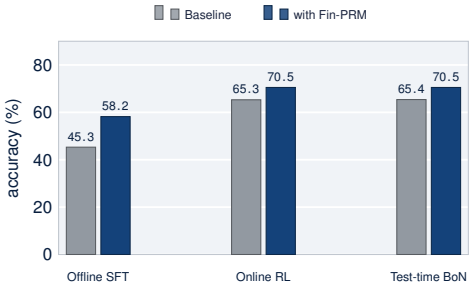
Wrong answer, good coverage:
$r_{\text{out}} = 0$, $r_{\text{cover}} = 0.8 \Rightarrow R_{\text{traj}} = 1.2 < 1.25$.

The threshold sits at the **midpoint** of the $[0, 2.5]$ range, and the **outcome dominates**: with a wrong answer, only near-perfect coverage ($r_{\text{cover}} > 5/6$) could sneak past. **Name-dropping the right terms cannot rescue a wrong trade.**

Trajectory-level reward design: Fin-PRM [39].

# One judge, three jobs

The labels train **Qwen2.5-7B-Instruct** into a scorer with two modes, a score per **step** and per **trajectory**, via a dual loss $\mathcal{L} = \mathcal{L}_{\text{step}} + \lambda\, \mathcal{L}_{\text{traj}}$ (binary cross-entropy on both). Four models to train one. Then the one judge serves three jobs:




Gains of $+12.9$, $+5.2$ and $+5.1$ points; Fin-PRM also beats Qwen2.5-Math-PRM-7B, a strong general math PRM [37], on CFLUE and FinQA [3, 38].

## 1. Offline: select the data

Keep only teacher trajectories the judge scores highly, then SFT on them. **Section 2's SFT, with curated data.**

## 2. Online: shape the RL

Dense per-step rewards inside RL: **a PRM plugged into this section's GRPO loop**, blended with the outcome reward ($r_{\text{t1}} = \frac{1}{2}\, r_{\text{out}} + \frac{1}{2}\, \hat{R}$).

## 3. Test time: rank the answers

Sample $N$ answers, let the judge pick: **best-of-$N$**. Self-consistency's majority vote, upgraded to a **judged** vote.

Applications and headline gains: Fin-PRM [39]; vs a rule-based outcome-only reward, the in-text RL gain is $+3.3$ points.

# Step back: it is all search

One frame unifies the section: every method is a **search over reasoning paths**, and differs only in how it **explores** and how it **scores** [11].

| **CoT** | **Self-consist.** | **ToT** | **GRPO** | **PRM / Fin-PRM** |
|---|---|---|---|---|
| one path, depth-first | *N* paths, majority vote | a tree, with backtracking | rollout search, gradient-weighted | same search, a better scorer |

## Explore: propose promising paths

Temperature sampling, trees, more rollouts, longer thinking. The whole **elicit** story, and half of **own**.

## Score: tell good paths from bad

Verifiers, outcome bits, noise corrections, PRMs, knowledge bases. The whole **judge** story.

Better explorers and better scorers **compound**. Section 4 widens the search space once more: from paths **within one model** to paths **across a team of models**.

Unifying frame: Lahlou, *A Gentle Introduction to LLM Reasoning*, NETYS 2026 [11].

# From reasoners to a team: the bridge to agents

Section 3, promise kept: we **elicited** reasoning with a prompt, **owned** it with an outcome bit, and **judged** it, even in finance, with process rewards.

- Both signals travel with us. The **pass/fail bit** will train the orchestrators of Section 4.
- **Judging intermediate work** returns too, reborn as a *role* an orchestrator can assign.
- And **GRPO** comes along, for a third section in a row.
- Research is already moving from lone reasoners to **teams**: CoRe trains models to **rescue** each other's failures [16].

## Bridge to Section 4

A reasoner, however well rewarded, is still **one model**: one style of thinking, one set of weights, one set of blind spots. And on today's benchmarks, **the best model changes from task to task**.

If no single model wins everywhere, the next step is not a bigger model. It is a system that **picks, sequences and checks** models: **agentic AI**.

SECTION 04

# Agentic AI: orchestrating models

04

# Why this exists: no single model wins everything

The frontier is no longer *one* model. Different providers now win different races, and the specialisations are getting **finer-grained**.

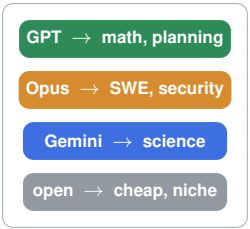
## Different champions, different races

- **Gemini** Deep-Think: **gold** at the IMO.
- **GPT-5.5**: disproved an 80-year-old **Erdős** conjecture.
- **Claude Mythos**: found **zero-day** flaws in OpenBSD/FreeBSD.

## Specialisation, even inside one domain

In competitive coding, **Gemini** is strong at *implementing* known algorithms, while **GPT** excels at *planning* and combining ideas for the hardest problems. No model dominates every cell.

*a jagged frontier*



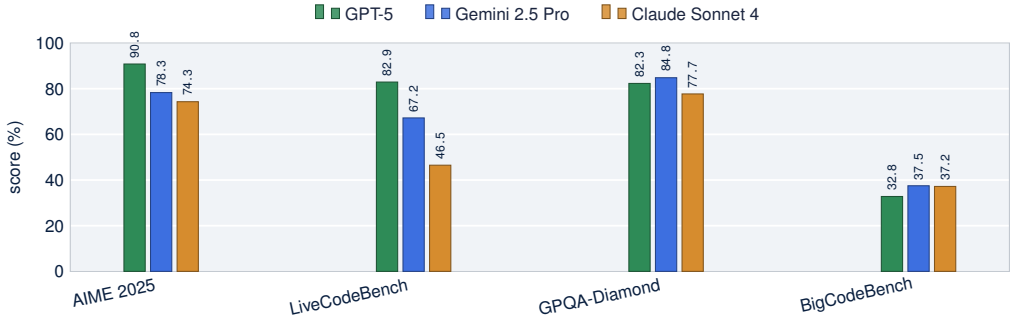

## The question

What if *one* system could engineer like the best coder, reason like the best mathematician, and **decide when each is needed**?

Motivating observations: Fugu technical report §1, June 2026 [23].

# The same story in numbers

Frontier models are **specialists in disguise**: on the same suite, the leader changes from task to task. If you could borrow the right model for each step, you would beat all of them.




GPT leads competition maths and live coding; Gemini leads graduate science and structured code. **The tallest bar moves.** And this is exactly the pool the two papers of this section will learn to conduct.

Numbers as reported in [19].

# Agentic workflows in plain language

**An LLM call**

Text in, text out. One shot, no actions.

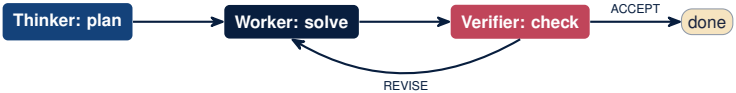
**An agent**

A model allowed to **act** (tools, code) and **react** to results, in a loop.

**A workflow**

A **plan** chaining several such steps, possibly across several models.

A useful trick is to give each call a **role**: a contract for what it returns.




Hold on to these three roles; the **Verifier** is Section 3's process judging, reborn as a role. The first paper of this section turns all three into a **learned** coordination scheme.

# The idea: orchestration as a new scaling axis

There are two familiar ways to buy more capability. Fugu adds a **third**.




## What Fugu is

A small model that **trains no frontier model of its own**. It learns to coordinate a pool of frontier workers, exposed as a **single API**. This is *macro-level model merging*: composition at the **behavioural** level, treating each model as a black box, no weights or architecture needed.

## Why it matters

- **Modular**: new models drop into the pool as they appear.
- **Controllable**: favour providers, exclude models, respect privacy.
- **Accessible**: frontier capability **without export-control risk**.

Orchestration as a scaling axis and macro-level merging: Fugu report §1–§2.

# Sakana Fugu: a team of models behind one API

## The product

**Sakana Fugu** (Sakana AI, June 2026) is an orchestration system exposed as a **single OpenAI-compatible API** [23]. You call one endpoint; inside, a learned orchestrator decides which frontier LLMs to use, in what order, and how to combine their outputs into one answer.

## The bet

Fugu trains **no frontier model of its own**. It is a small model that learned **how to put existing models to work together**. Progress from coordination, not from training something larger.

## Built on two ICLR 2026 papers

**TRINITY** *An Evolved LLM Coordinator.* A tiny model learns, by **evolution**, to pick one LLM per turn and give it a role: Thinker, Worker, or Verifier.

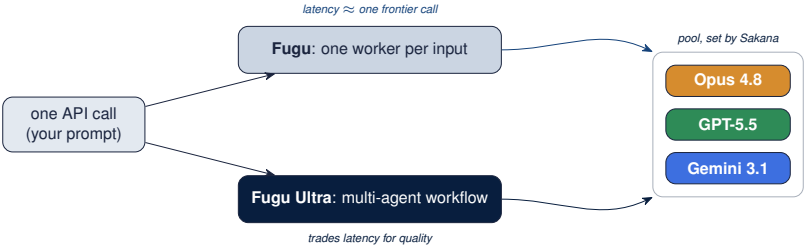
**The Conductor** *Learning to Orchestrate Agents in Natural Language.* A 7B model learns, by **reinforcement learning**, to write a full multi-agent plan in plain language.

We meet both in detail after the product tour.

Sakana AI, Fugu release (June 2026); technical report arXiv:2606.21228.

# One endpoint, two tiers

You pick a **tier** per call; the integration never changes. Both draw on the **same pool** (**Sakana fixes** it, not you); the tier only sets **how much coordination** happens behind the endpoint.




## Fugu vs Fugu Ultra

**Fugu** routes each query to the single best worker, so it is about as fast as one frontier call, yet often more accurate. **Fugu Ultra** builds a workflow over several workers per input, and can call itself recursively (slower, highest quality).

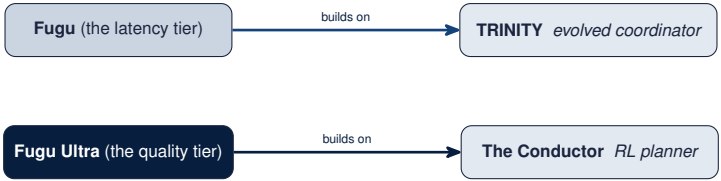
## Who picks what

- **Sakana** fixes the pool, the menu of workers.
- The **orchestrator** chooses which worker(s) to call, per query (learned, hidden from you).
- **You** can only **opt models out**, and only on Fugu; Ultra's pool is fixed.

Tier behaviour and pool control: Fugu technical report and release page, June 2026.

# Under the hood: two papers behind the two tiers

The product is one interface; underneath sit **two distinct learned orchestrators**, one per tier. The rest of this section opens each up, high level.




## Fugu ← TRINITY

A tiny coordinator picks one worker per input. Fugu keeps the idea but **drops the roles**, so the only decision is *which model*, which keeps latency low.

## Fugu Ultra ← Conductor

A language model writes a full multi-agent plan. Fugu Ultra adds production pieces: **function calling**, agent isolation and **shared memory**.

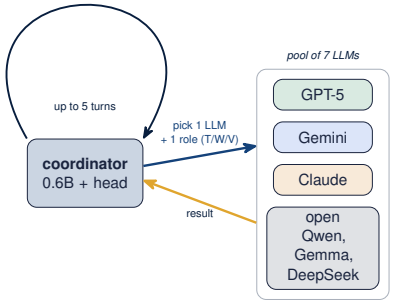
"Building on the Trinity and Conductor frameworks." Fugu technical report §3, June 2026.

# TRINITY in one picture

The first paper [34] asks: can a **tiny** model learn to **conduct** a pool of powerful LLMs, without ever touching their weights?

- The coordinator is a **0.6B** small model plus a head of about **10K** parameters. Total learnable: **under 20K**.
- It runs over up to **five turns**. Each turn it picks one worker and gives it one of three **roles**.
- It is trained by **evolution**, not gradient descent, on a single bit of feedback: did the final answer pass?

**Result:** a new state of the art on LiveCodeBench; zero-shot, it wins three of the four unseen tasks and ties the fourth, matching or beating every model in its own pool.




Xu, Sun, Schwendeman, Nielsen, Cetin, Tang. TRINITY. ICLR 2026 (2512.04695).

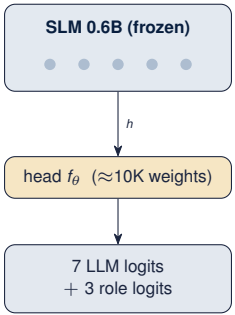
# The coordinator: a small model plus a tiny head

The coordinator never writes the answer. It only reads a hidden state and emits **logits** that choose an LLM and a role. Its generated text is thrown away; prompting is delegated to the pool.

**Two sets of trainable parameters:**

- A **lightweight head** after the final layer. It maps a hidden vector $h \in \mathbb{R}^{1024}$ to $L+3$ logits: $L$ to pick one of $L$ LLMs, **3** to pick a role. About **10K** weights.
- **Singular-value fine-tuning.** A few thousand extra numbers that lightly retune one backbone layer, sharpening the hidden state for routing.

Reading an **earlier** token (not the last) lets it decide **before** a full generation finishes: fast and cheap.




Only the gold head (plus a tiny backbone tweak) is trained. The 0.6B body stays frozen.

Parametrization, TRINITY §3.1, Fig. 2. ICLR 2026.

# What the coordinator is really doing




## The decision

The head outputs **10 scores**: 7 for "which LLM", 3 for "which role". A softmax over each group turns scores into a choice. The chosen LLM runs, its answer is appended, and the loop repeats.

## The only feedback

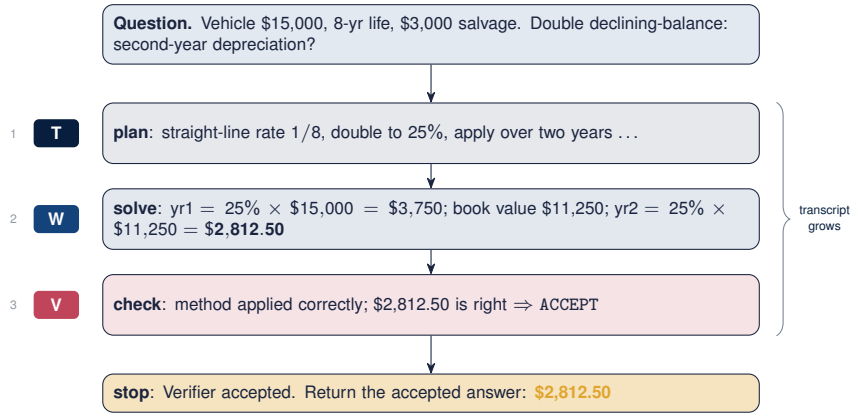
After the whole run, the final answer is compared to the **known correct answer**. That gives **one bit**: pass or fail. Nothing else.

**Training's goal:** tune the ≈10K head weights so that bit is "pass" as often as possible.

Problem setup, TRINITY §2.

# A trajectory: one full run, start to finish

The depreciation question from the paper. The transcript **grows**; each turn spends a different role until a Verifier accepts.

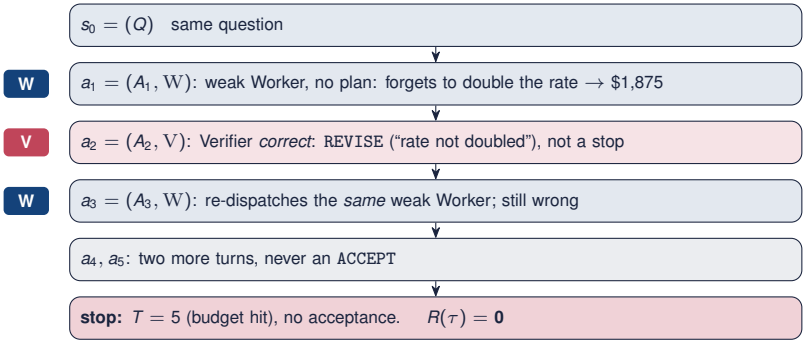



Each turn appends to the transcript; the next turn reads everything above it. Up to 5 turns; here it stops at 3. **The output is the accepted answer, not the whole transcript.**

Worked example, TRINITY Fig. 1 (right). ICLR 2026.

# The same weights, a run that fails

Identical $\theta$ and question as the passing run, but the policy's draws send it down a worse path. The reward is a single 0.



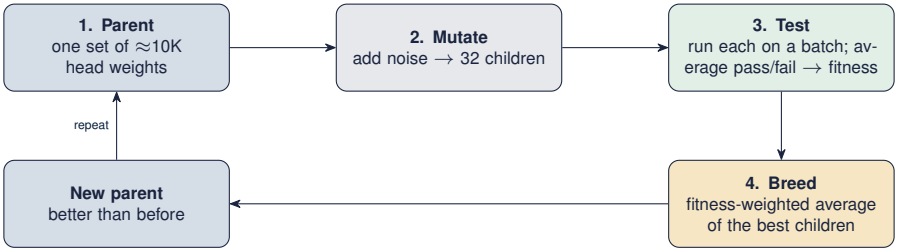

## The credit-assignment problem, in one image

One bit, smeared across five entangled turns: Section 3's **credit-assignment** curse. It never says turn 1 skipped planning, nor that the weak Worker was the culprit; worse, the Verifier at turn 2 did its job *perfectly*, yet sits inside a losing run. **Evolution sidesteps this**: it scores the *whole* weight vector by its average pass rate, never asking which turn failed.

Worked example mirrors TRINITY Fig. 1. ICLR 2026.

# How it learns

**mutate** → **test** → **breed**, over and over. Each loop is one “generation”.




## Why average the rewards

One run gives a noisy single bit. Running each child on **many** questions and averaging turns that into a stable fitness score.

## What it discovers

Purely from win/lose signals, it learns **who is good at what**: coding → GPT-5, math → Gemini, knowledge → Claude. Nobody labelled this.

sep-CMA-ES training loop, TRINITY §3.3; learned specialization, Fig. 6 & 15.

# Why evolution, not gradient descent?

The single reward bit sits at the *end* of a multi-turn run. That makes the obvious tools fail, and hands the win to a method that never differentiates.

## Three obstacles to a gradient

- **Non-differentiable**: the run is black-box API calls, there is no $\nabla_\theta J$.
- **Credit assignment**: one bit, smeared across up to five entangled, variable-length decisions, good and bad moves in the same run.
- **Weak coupling**: each of the ≈10K head weights barely moves the scalar reward, so REINFORCE's per-parameter gradients are **low-SNR**.

## What sep-CMA-ES does instead

It never asks "which turn failed". It scores the *whole* weight vector by its average pass rate and walks toward the winners, exploiting **block-$\varepsilon$ separability**: the signal lives within tiny blocks (one per logit), so a diagonal covariance is near-optimal.

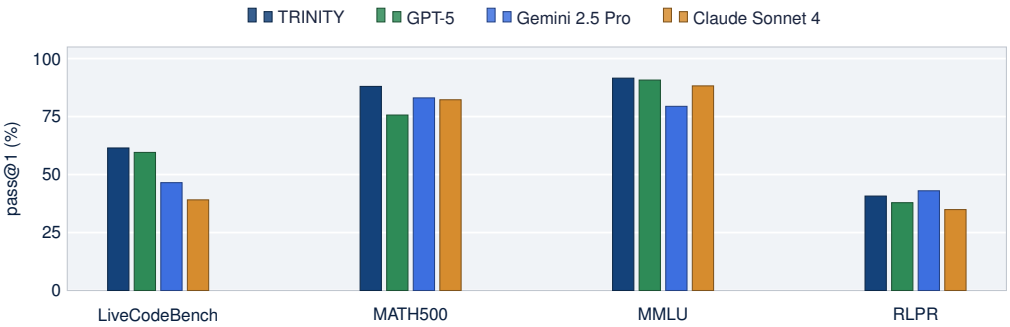



Same budget. Evolution tops RL, random search, and even supervised imitation, whose labels also become intractable over multi-turn runs.

Optimiser comparison, TRINITY Tab. 4; separability & budget, §3.3 and App. A.1.

# TRINITY's report card: the constrained-budget view

Capped budget for all; the single-model bars are the paper's **fixed-agent** ablation: one LLM kept, same tri-role loop. TRINITY leads on **three of the four** training tasks (the Gemini variant keeps RLPR). Unconstrained, TRINITY sets a new **LiveCodeBench state of the art** at **86.2** (GPT-5: 83.8), and it **generalises zero-shot** to held-out tasks.




Ablation setting, TRINITY Tab. 2; unconstrained SOTA, §4.4 [34].

# Ablations, and the honest limitation

**Every piece earns its place.**

- Drop the **tri-role** scheme ⇒ big losses on MATH500 and RLPR.
- Read the **last** token instead of the penultimate ⇒ LiveCodeBench collapses.
- Remove **agent selection** (fix one LLM, keep the roles) ⇒ clear degradation.

## The honest limitation

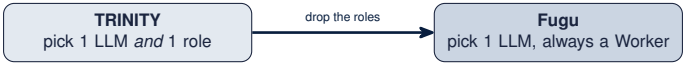
TRINITY can **plan** tool use but cannot yet **act** on tools: it orchestrates conversation, not execution.

Closing that gap is precisely the job of the **production tiers** we meet next.

Ablations, TRINITY Tab. 3 and §5. ICLR 2026.

# From TRINITY to Fugu: the latency tier

Fugu *is* TRINITY, scaled to a production setting where response time matters as much as quality. The lineage is direct; two things change.




## What stays

The same **decision-only** parametrization: read an *early* hidden state, apply a lightweight head, dispatch. No autoregressive decoding, so the orchestrator adds almost nothing. Latency $\approx$ **one frontier call**.

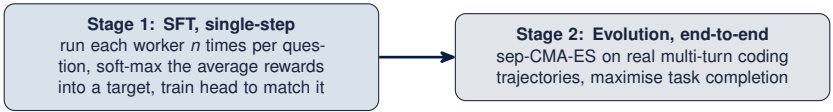
## What changes

- **No roles.** Every selected model runs as a Worker. The coordination space shrinks to "which model", cutting latency.
- **Multi-turn.** Optimised for end-to-end interactive tasks, not just single-step questions.

Fugu parametrization vs Trinity: Fugu technical report §3.1, Fig. 2.

# Fugu's two-stage training

TRINITY trained the head by evolution alone. Fugu **warm-starts** it with supervised learning first, then refines on real interactive tasks.




## Why soft targets, not one label

Supervising on a distribution $p(j) \propto \exp(\bar{r}_j/\tau)$ rather than a single "best worker" makes selection **robust** when several workers are similarly capable, and needs no generation from the orchestrator.

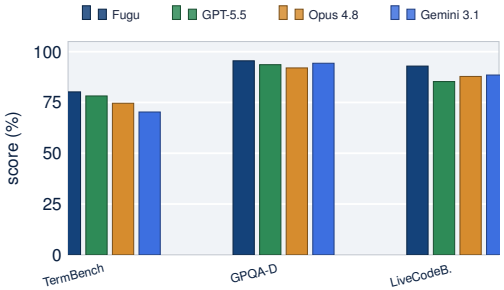
## Why end-to-end trajectories

Real Claude Code / Codex / OpenCode runs add repo context, tool calls and execution feedback. They reveal what scores hide: some models **reason** well but **operate tools** poorly, and the reverse.

Two-stage training: Fugu technical report §3.1.2–§3.1.3.

# Fugu in production: one worker, frontier-call speed

Despite choosing a *single* model per input, Fugu can beat the best individual model, because the choice is made **per step**.




## Per-step adaptivity, in practice

On Terminal Bench, Fugu **alternates** GPT-5.5 and Opus-4.8 across a solution, calling in Opus at **critical debugging points**. One model at a time, but a different one when the moment calls for it.

## The operating point

Latency comparable to a direct frontier call, yet SOTA on GPQA-Diamond (tying Ultra) and ahead of every worker on Terminal Bench. **Speed and quality, not a trade.**

**Next:** the opposite design. Instead of a head that picks, a language model that **writes the whole plan**.

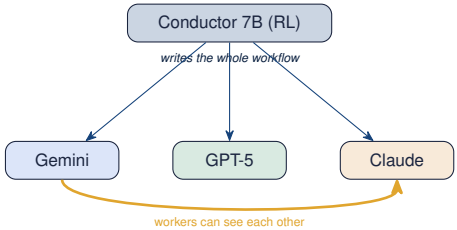
Fugu technical report §4.1.2–§4.1.3, Model Card (Tab. 1). Baselines provider-reported.

# The Conductor in one picture

The second paper [19] takes the opposite route to TRINITY. Instead of a tiny head emitting logits, the Conductor is a **7B language model** (from a Qwen2.5 checkpoint) trained with **reinforcement learning** to *write* a complete agentic workflow, in plain language, for each question.

- Its output is a **plan**: which worker does what, in which order, seeing which earlier results.
- Because the plan is natural language, it can express **chains**, **trees**, **best-of-N**, even **recursion**.
- It is rewarded only for a **parseable plan** whose **final answer is correct**.

**Result:** a 7B model that lifts its much larger workers to a new frontier on LiveCodeBench and GPQA-Diamond.




Nielsen, Cetin, Schwendeman, Sun, Xu, Tang. The Conductor. ICLR 2026 (2512.04388).

# The output: three lists that define a workflow

After its private chain-of-thought, the Conductor emits **three Python lists of equal length**, parsed into a workflow of up to **five steps**. That is the whole interface, no fixed scaffold.

- `model_id` : which numbered worker runs each step.
- `subtasks` : the natural-language instruction for that step.
- `access_list` : which earlier results that step may see (or `all`, or `[]`).

Workers are shown as bare numbers (Model 0, Model 1, . . . ) with **no brand names**, so the Conductor learns competence from **reward**, not reputation.

## Example (count complete subarrays)

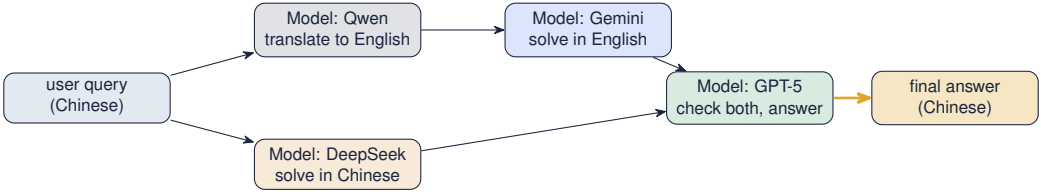
```
model_id = [2, 0]
subtasks = [
 "Devise an efficient algorithm...",
 "Implement it in Python"]
access_list = [[], ["all"]]
```

Step 1 (Model 2) plans the algorithm with no context; Step 2 (Model 0) implements it, seeing step 1's output.

Conductor output structure, §3.1, Fig. 2. ICLR 2026.

# A worked schematic: translation meets mathematics

A request arrives in **Chinese** that needs both translation and math. The Conductor builds a small **tree**: two independent solution paths, then a third worker checks both before answering, in Chinese.

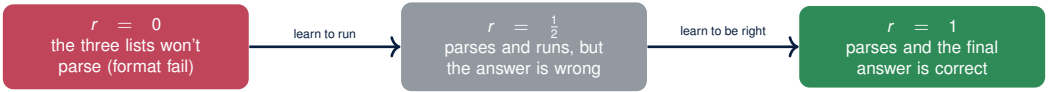



Each arrow is an entry in an `access_list`; the branches are parallel paths; GPT-5's step has both attempts in context. A **tree topology**, written entirely in language, with no hand-built scaffold.

Conductor App. C, Fig. 11 (ICLR 2026).

# The reward ladder: a tiny curriculum

The reward never explains *why* a run was bad. It returns one of three numbers, and that ladder teaches format first, correctness second.




## Why the partial credit matters

Early on, almost all reward comes from the $0 \to \frac{1}{2}$ jump: "first, write a runnable program". Once format is solved, every group is full of $\frac{1}{2}$'s and the only way to stand out is the $\frac{1}{2} \to 1$ jump. Without the $\frac{1}{2}$, well-formed-but-wrong would look identical to garbage.

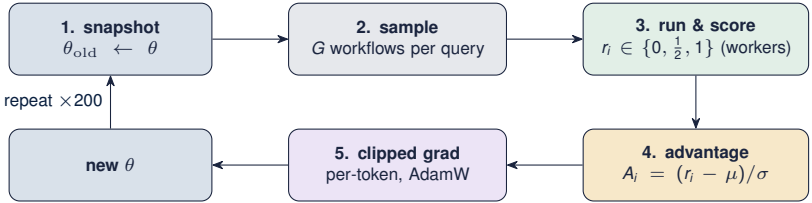
## The budget

200 GRPO iterations, batch **256** (4 queries $\times$ 64 rollouts), on **2×H100**, no KL, no value network. Cheap because the *workers* carry the reasoning load.

Reward ladder and training budget: Conductor §3.1, §4.1, App. A.1.

# One GRPO iteration, end to end

Snapshot the policy, sample $G$ completions per query, execute them on the frozen workers to get rewards, form advantages, take one clipped per-token AdamW step. Repeat 200 times.



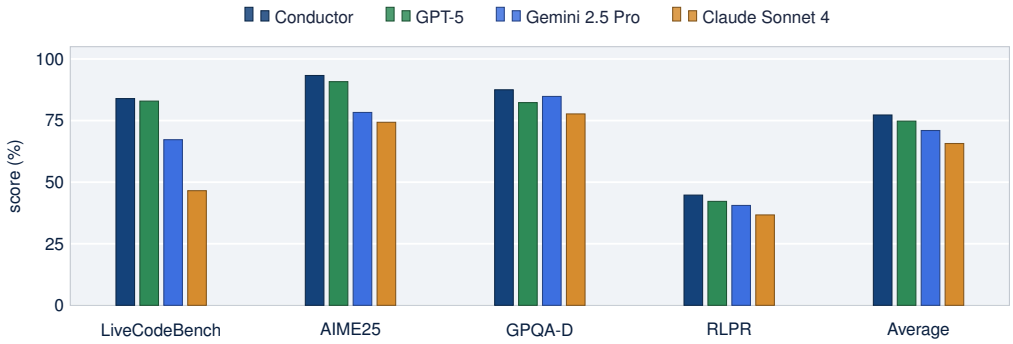

## The callback to Sections 2 and 3

This is **exactly** the GRPO recipe from Sections 2 and 3: the group of siblings replaces the critic, a verifier-style check replaces the learned reward. Only the policy changed, from a **solver** writing answers to a **planner** writing workflows.

GRPO training loop: Conductor §3.1, §4.1, App. A.1; GRPO [25].

# Results: a 7B Conductor sets a new frontier

Unconstrained setting, against the best public numbers for each worker. The Conductor is highest on every task shown, and on average.




The gains on GPQA-Diamond and AIME25 are around 3 points, the size of a **generational** jump, here delivered across math, code and science at once.

Conductor Tab. 1 (unconstrained); the Average column spans every task in that table, not only the four plotted here. ICLR 2026.

# From the Conductor to Fugu Ultra: the quality tier

Fugu Ultra *is* the Conductor, scaled to production and taught to drive a team of **tool-using** agents over long horizons. Same core recipe, three pieces added.




## What stays

The same output: three lists defining who does what, seeing what. Trained with **GRPO, no KL**, over a pool of **Gemini-3.1-Pro, Opus-4.8, GPT-5.5**; workflows up to five steps; the Conductor may name itself.

## What is new

- **Function calling** inside multi-agent workflows.
- **Intra-workflow isolation** of each agent's tool trajectory.
- **Persistent shared memory** across workflows.

Fugu Ultra builds on the Conductor: Fugu technical report §3.2.

# Three production pieces for tool-using teams

Letting any agent call functions inside a multi-agent plan creates a **memory** problem. Fugu Ultra resolves it with one tension, balanced two ways.

## 1. Function-calling workflows

Any agent may call a tool at any time, so the system tracks **which agent** emitted each call and **where** it sits in the workflow, routing each function-call loop back to the right agent.

## 2. Intra-workflow isolation

Within a workflow, an agent sees others **only through the access list**. This prevents *orchestration collapse*, where the first agent to touch the environment steers all the rest.

## 3. Persistent shared memory

*Across* workflows, agents **do** see prior tool calls, so they don't rediscover the same artifacts or repeat calls, and keep the background context of a multi-turn conversation.

## The balance

**Isolate within** a workflow (each agent free to find its own path); **share across** workflows (no wasted rediscovery). Freedom and memory at once.

Function calling, isolation, shared memory: Fugu technical report §3.2.2.

# Fugu Ultra's emergent playbook

The same per-question adaptivity the Conductor showed, now over frontier workers and real tools. Three recurring strategies:

## Debate & aggregation

A tree with two independent attempts and an **adaptive aggregator**: Gemini synthesises a trivia-heavy question, GPT a math-heavy one. Fixed-aggregator systems cannot switch the synthesiser; this is the edge.

## Build & debug

GPT-5.5 builds; Opus-4.8 is called in at critical moments to enumerate concrete risks and expose vulnerabilities, then hands findings back.

## Bring in a specialist

On a FEAL cryptanalysis task, Opus builds a chosen-plaintext attack, then GPT is tasked *as a math specialist* to re-derive the differential from first principles. Cross-domain expertise, composed on demand.

## Why it works

The aggregator, the debugger, the specialist, all **chosen per question**. That adaptivity is exactly what rigid multi-agent scaffolds lack.

Optimal strategies and topologies: Fugu technical report §4.4.

# The scoreboard: Fugu against its own workers

Now that both tiers are open, one table with every headline benchmark. Fugu and Fugu Ultra are evaluated against the *same* frontier models they orchestrate. **Bold** marks the best in each row.

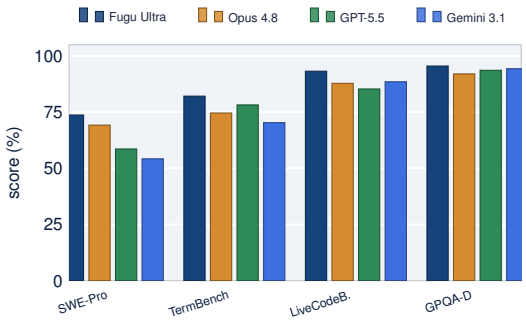
| Benchmark | Fugu Ultra | Fugu | Opus 4.8 | Gemini 3.1 | GPT-5.5 |
|---|---|---|---|---|---|
| SWE-Bench Pro | **73.7** | 59.0 | 69.2 | 54.2 | 58.6 |
| Terminal Bench 2.1 | **82.1** | 80.2 | 74.6 | 70.3 | 78.2 |
| LiveCodeBench | **93.2** | 92.9 | 87.8 | 88.5 | 85.3 |
| LiveCodeBench Pro | **90.8** | 87.8 | 84.8 | 82.9 | 88.4 |
| Humanity's Last Exam | **50.0** | 47.2 | 49.8 | 44.4 | 41.4 |
| CharXiv Reasoning | **86.6** | 85.1 | 84.2 | 83.3 | 84.1 |
| GPQA Diamond | **95.5** | **95.5** | 92.0 | 94.3 | 93.6 |
| SciCode | 58.7 | **60.1** | 53.5 | 58.9 | 56.1 |
| $\tau^3$ Banking | 20.6 | **21.7** | 20.6 | 8.4 | 20.6 |
| Long Context Reasoning | 73.3 | **74.7** | 67.7 | 72.7 | 74.3 |
| MRCRv2 | 93.6 | 86.6 | 87.9 | 84.9 | **94.8** |

Read it as: on **most** rows a learned orchestrator beats every public model it draws on, with the agentic-coding gains the size of a **generational** upgrade.

Fugu technical report, Model Card (Tab. 1), June 2026. Baselines provider-reported.

# Where it wins, and where it does not

Against its own pool, Fugu Ultra tops most benchmark rows, with gains on agentic coding the size of a generational upgrade. The honest exceptions all share one cause: the orchestrator cannot exceed what its workers can reach.




Fugu Ultra clears every model in its pool on these four.

## The honest exceptions

- **MRCRv2** (long-context recall): GPT-5.5 **94.8** edges Ultra **93.6**.
- **HLE**: Fable 5 leads at **53.3** vs **50.0**, but Fable 5 is **not in the pool**.
- **SciCode**: Gemini **58.9** just tops Ultra **58.7** (though *Fugu* wins it at 60.1).

## Read the claim correctly

On public, swappable models a learned orchestrator beats every single one of them on **most** tasks, at competitive cost. Numbers are vendor-reported and scaffold-dependent. And the ceiling stands: Fugu can be no better than the **union** of what its pool can do.

Fugu technical report, June 2026 (model card Tab. 1). Baselines provider-reported.

# Beyond benchmarks: where orchestration pays off

Sakana also stress-tests Fugu Ultra on open-ended tasks, where success needs sustained exploration, not a single step. A gallery, all vs strong single-model agents under identical scaffolds.

## Research and engineering

- **AutoResearch**: lowest validation BPB on an autonomous ML-training loop (0.9774 mean, best 0.9748).
- **CAD**: a working mechanical iris that opens and closes, where baselines leave gaps or weak links.
- **Rubik's cube**: shortest solutions, **19.72** HTM on average, under the 20-move worst-case bound (God's number); Fugu trades one move for a $\sim 35\times$ faster solver.

## Reasoning under pressure

- **Classical kana reading order**: best NED **0.776**, ahead of every frontier baseline, on a task with no training data.
- **Blindfold chess**: wins against all three frontier baselines and a 2100-Elo Stockfish, no board ever shown.
- **Online trading**: strongest 50-week return, **+19.43%**, compounding gains from week-to-week feedback.

These are illustrative, scaffold-matched comparisons, not aggregate win rates. The trading result is a single 50-week run, not a track record.

Performance beyond benchmarks: Fugu technical report §4.3 and App. B.

SECTION 05

# Conclusion

05

# Two routes to the same destination

Both papers achieve **test-time composition**: stronger results from a fixed pool of models, with no access to their weights. They differ in almost every design choice, and Fugu ships both, one per tier.

| | TRINITY | The Conductor |
|---|---|---|
| Powers tier | **Fugu** (latency) | **Fugu Ultra** (quality) |
| Coordinator | 0.6B small model + tiny head | full 7B language model |
| Learnable params | under 20K | all 7B (RL finetune) |
| Training signal | sep-CMA-ES (evolution) | GRPO (reinforcement learning) |
| What it outputs | logits: pick worker + role | a workflow written in language |
| Coordination unit | three role contracts (T/W/V) | subtasks + access lists |
| Per-input shape | one worker per turn, up to 5 turns | a workflow: avg $\approx$ 3, max 5 steps |
| Topologies | chains of roles over turns | best-of-N, chains, trees |
| Recursion | not supported | yes, a new scaling axis |

## What they share

No worker weights touched; both **match or beat every model** in their pool; both **generalize zero-shot**; both set records on LiveCodeBench.

## The trade

TRINITY is **tiny and fast**; the Conductor is **expressive and flexible**. The product exposes that trade directly, as two tiers.

TRINITY (2512.04695) and the Conductor (2512.04388), ICLR 2026; Fugu report (2606.21228).

# What to remember

1. **Transformers**: self-attention builds contextual embeddings; modern LLMs are **decoder-only** next-token predictors.
2. **Pretrain** $\rightarrow$ **SFT** $\rightarrow$ **preference tuning** turns a text completer into an aligned assistant; PPO, DPO and **GRPO** (the group replaces the critic) optimise the same objective.
3. **Reasoning** needs a reward for the thinking: verifiable **outcome bits** are cheap but **blunt**; **PRMs** score every step, and **Fin-PRM** specialises the judge to finance with knowledge-grounded rewards.
4. No single model wins everywhere: the frontier is **jagged**, and the tallest bar moves.
5. **Orchestration** is a third scaling axis: TRINITY evolves a tiny router; the Conductor writes whole workflows with GRPO. Both match or beat every model in their pool.
6. Each chapter is **bounded** by the last, which is exactly what motivates the next.

## The arc of the lecture

One decoder $\rightarrow$ aligned assistant (Vanilla LLMs) $\rightarrow$ thinks, with rewards that judge the thinking (Reasoning & PRMs) $\rightarrow$ commands a team of models (Agentic AI).

**Thank you.**

# References I


[1] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4): 324–345, 1952.

[2] T. B. Brown et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33*, 2020.

[3] Z. Chen et al. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[4] K. Cobbe et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[5] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[6] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems 36*, 2023.

[7] J. Hoffmann et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[8] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

[9] E. J. Hu et al. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[10] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35*, 2022.

[11] S. Lahlou. A gentle introduction to LLM reasoning. Invited talk, NETYS 2026, 2026.

[12] S. Lahlou, A. Abubaker, and H. Hacid. PORT: Preference optimization on reasoning traces. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics*, 2025.

[13] H. Lightman et al. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

# References II


[14] A. Madaan et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36*, 2023.

[15] O. E. Mansouri, F. A. Izzati, M. E. A. Seddik, and S. Lahlou. Noise-corrected GRPO: From noisy rewards to unbiased gradients. In *International Conference on Machine Learning*, 2026.

[16] K. Mishra, M. Aubakirov, M. Takáč, N. Lukas, and S. Lahlou. CORE: Collaborative reasoning via cross teaching. In *International Conference on Machine Learning*, 2026.

[17] N. Muennighoff et al. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

[18] N. Natarajan, I. S. Dhillon, P. Ravikumar, and A. Tewari. Learning with noisy labels. In *Advances in Neural Information Processing Systems 26*, 2013.

[19] E. Nielsen, E. Cetin, P. Schwendeman, Q. Sun, S. Xu, and Y. Tang. The conductor: Learning to orchestrate agents in natural language. In *International Conference on Learning Representations*, 2026.

[20] L. Ouyang et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35*, 2022.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

[22] R. Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems 36*, 2023.

[23] Sakana AI. Sakana fugu: Frontier capability from learned orchestration. Technical report, Sakana AI, technical report arXiv:2606.21228, 2026.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

# References III


[25] Z. Shao et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[26] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36*, 2023.

[27] J. Uesato et al. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

[28] A. Vaswani et al. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 2017.

[29] P. Wang et al. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

[30] X. Wang et al. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

[31] J. Wei et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35*, 2022.

[32] J. Wei et al. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.

[33] Z. Xie et al. FinChain: A symbolic benchmark for verifiable chain-of-thought financial reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026.

[34] S. Xu, Q. Sun, P. Schwendeman, E. Nielsen, E. Cetin, and Y. Tang. TRINITY: An evolved LLM coordinator. In *International Conference on Learning Representations*, 2026.

[35] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36*, 2023.

[36] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

# References IV


[37] Z. Zhang et al. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025.

[38] J. Zhu et al. Benchmarking large language models on CFLUE: A Chinese financial language understanding evaluation dataset. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

[39] Y. Zhu et al. Fin-PRM: A domain-specialized process reward model for financial reasoning in large language models. *arXiv preprint arXiv:2508.15202*, 2025.