

The Art of Scaling Reinforcement Learning Compute for LLMs

Rishabh Agarwal

Reinforcement Learner

RL Training: **The New Scaling Frontier**

Pre-training teaches a model about the world...

...but RL unlocks the capabilities that matter most:

 **Test-time thinking**

OpenAI o1/o3, DeepSeek R1

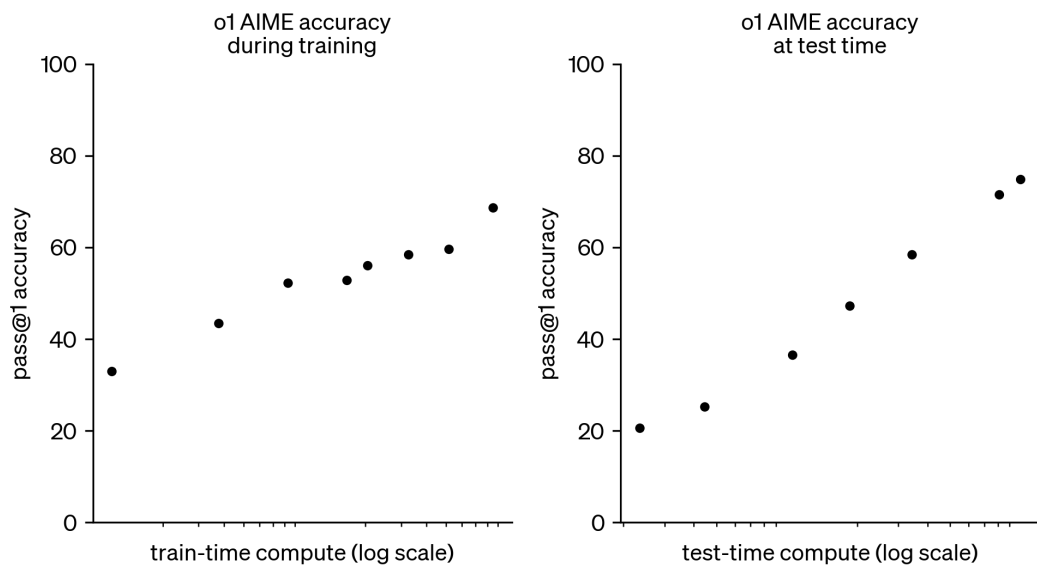
 **Agentic capabilities**

Kimi K2, Claude Code, Cursor

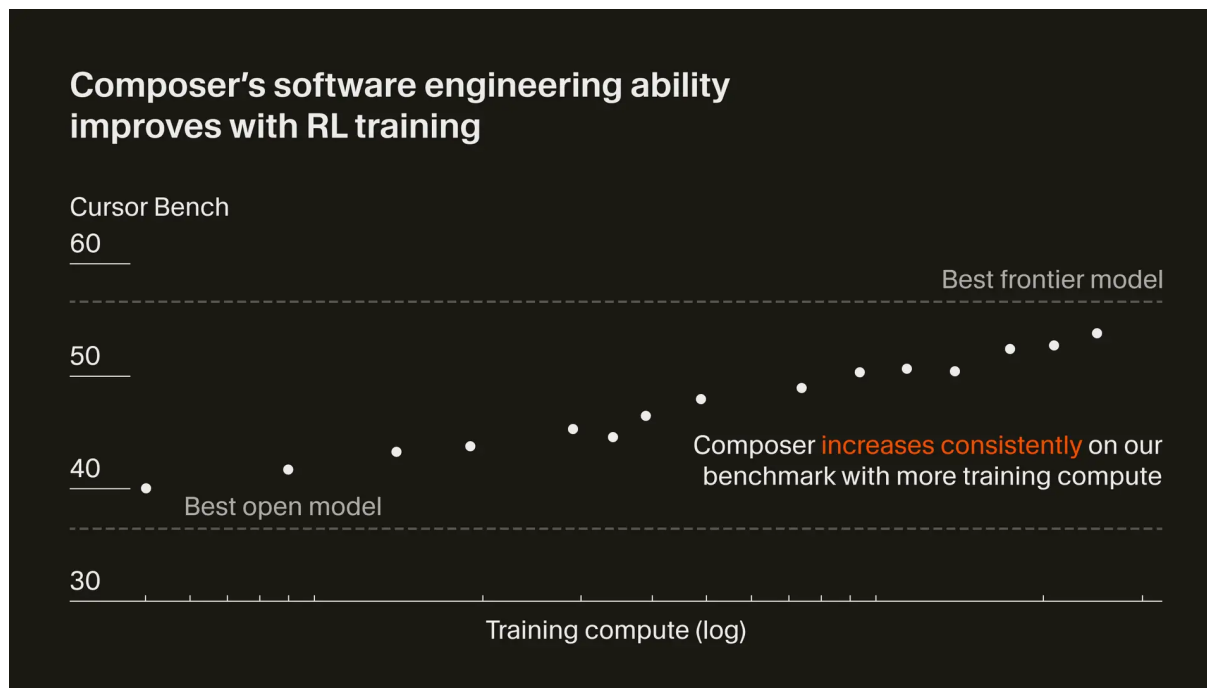
 **Scientific discovery**

Axiom, Periodic Labs etc

Scaling RL Compute for LLMs



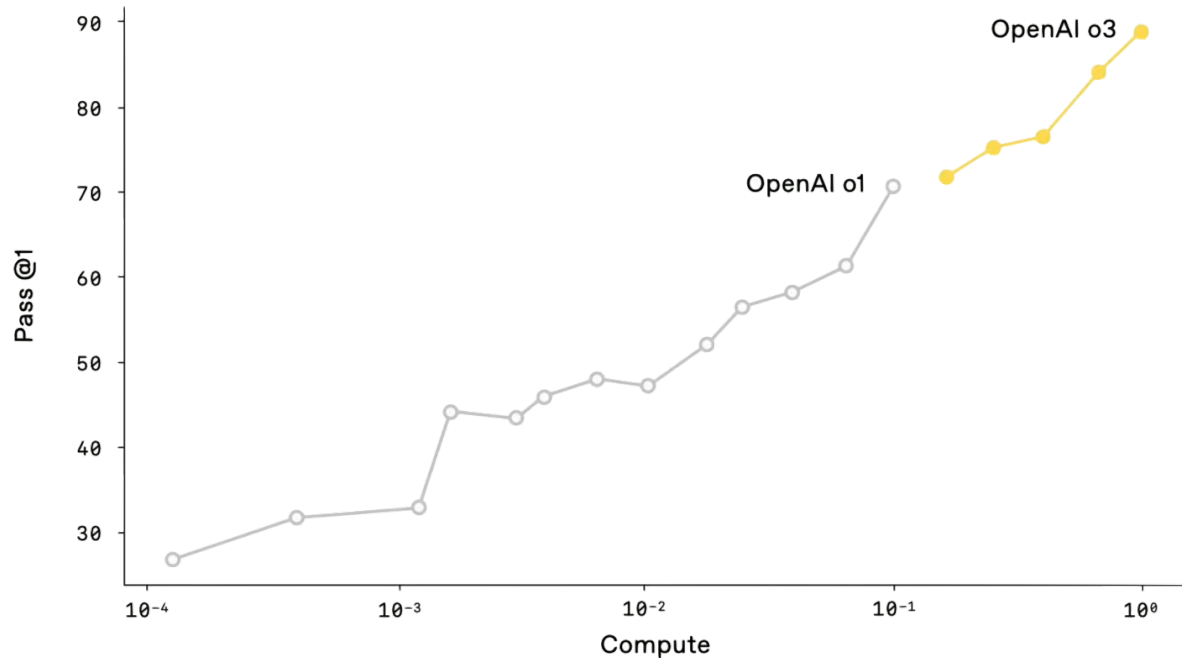
Test-time reasoning



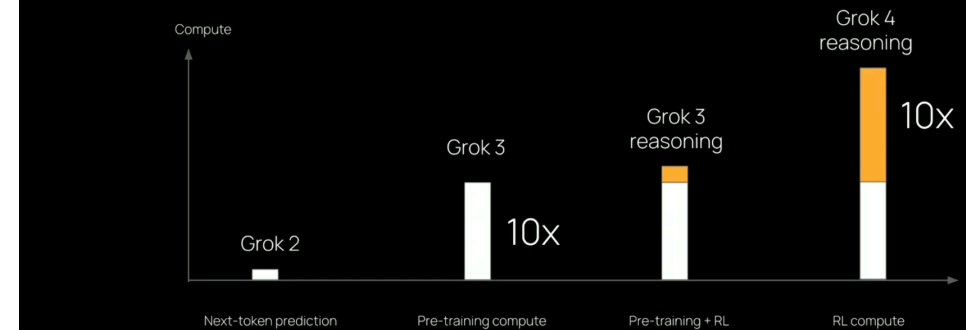
Agentic capabilities (Coding)

RL compute is exploding

AIME (2022/2023) performance during training



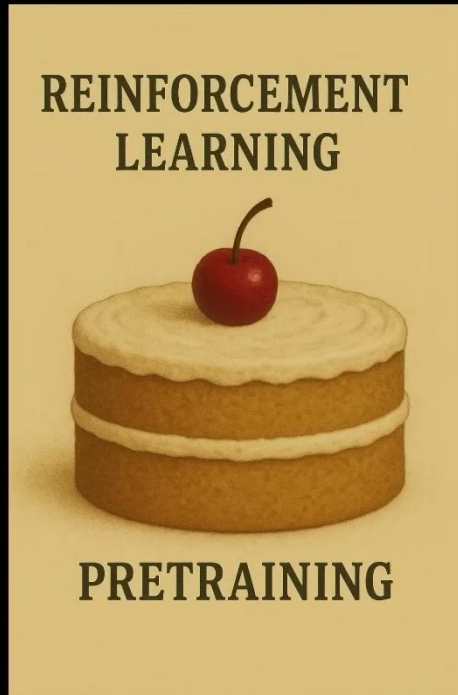
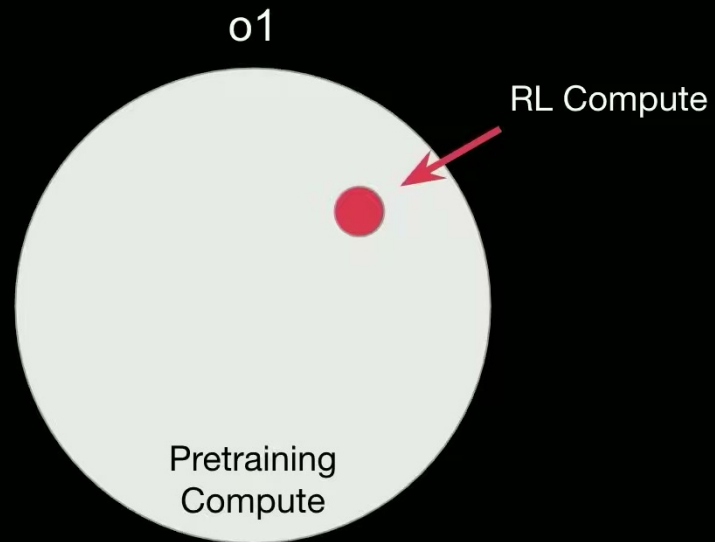
Ludicrous rate of progress



10x more RL compute from o1 to o3 or Grok 3 $\rightarrow$ Grok 4

RL compute is exploding

Scaling Reinforcement Learning



How Much Information is the Machine Given during Learning?

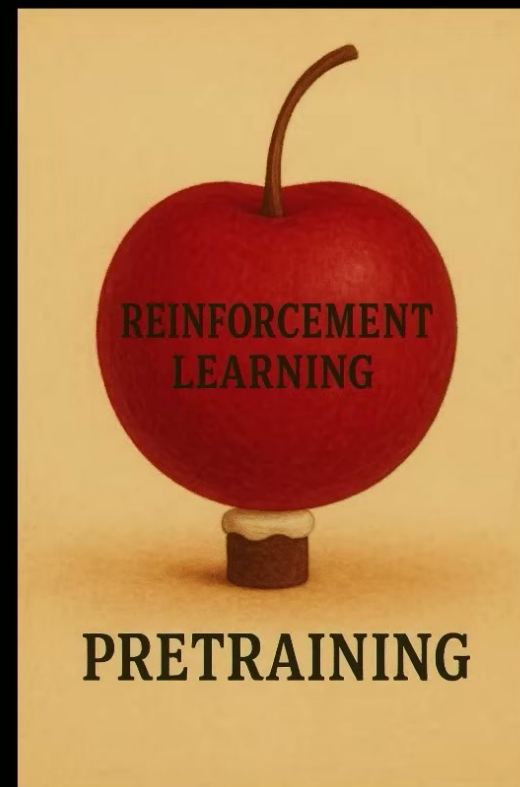
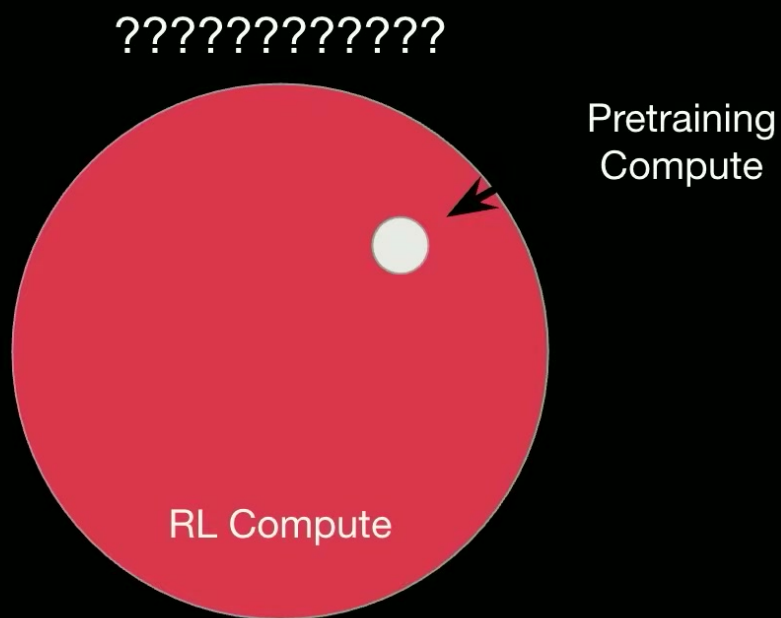
- ▶ **"Pure" Reinforcement Learning (cherry)**
 - ▶ The machine predicts a scalar reward given once in a while.
 - ▶ **A few bits for some samples**
- ▶ **Supervised Learning (icing)**
 - ▶ The machine predicts a category or a few numbers for each input
 - ▶ Predicting human-supplied data
 - ▶ **10→10,000 bits per sample**
- ▶ **Self-Supervised Learning (cake génoise)**
 - ▶ The machine predicts any part of its input for any observed part.
 - ▶ Predicts future frames in videos
 - ▶ **Millions of bits per sample**



© 2019 IEEE International Solid-State Circuits Conference 1.1: Deep Learning Hardware: Past, Present, & Future 99

RL compute is exploding

Scaling Reinforcement Learning



Scaling RL: More **Art** than **Science**

RL compute has scaled massively, but our understanding of how to scale RL has not kept pace.

-  **No scaling framework** Can't predict which methods will work when scaling compute 10x or even 100x
-  **Ad-hoc solutions** DAPO, MiniMax, Magistral, Kimi etc — tailored to specific contexts
-  **Academic exclusion** With no way to identify promising methods a priori, progress requires large-scale experiments most labs can't afford

Can we bring the predictability of pre-training scaling laws to RL?

Why care about predictable scaling?

The Bitter Lesson in AI

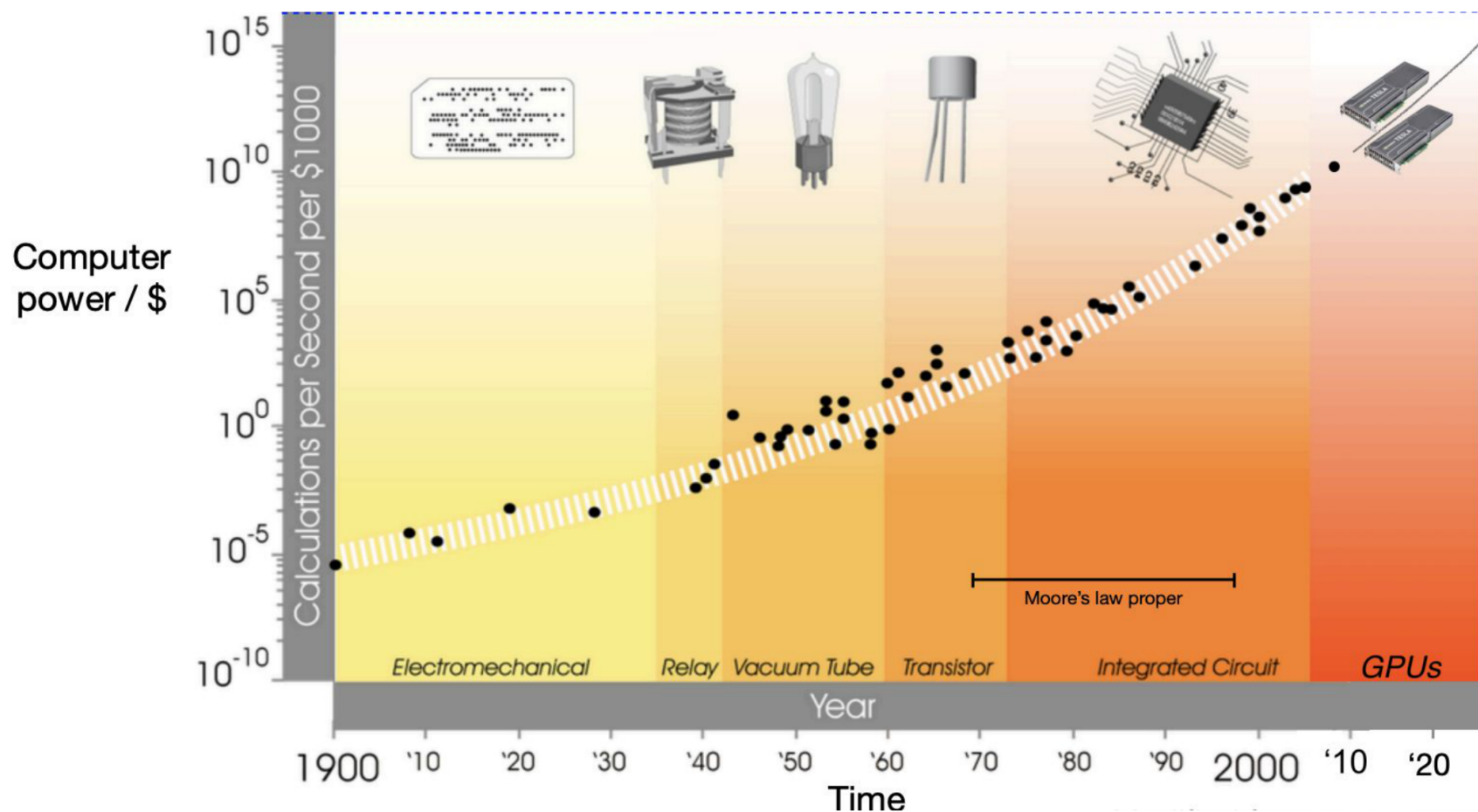
*“The biggest lesson that can be read from 70 years of AI research is that general **methods that leverage computation** are ultimately the **most effective**, and by a large margin.”*

*“The two methods that **seem to scale arbitrarily** in this way are **search and learning**.”*

– Richard Sutton

<http://www.incompleteideas.net/InIdeas/BitterLesson.html>

Premise: Compute cost is decreasing exponentially!

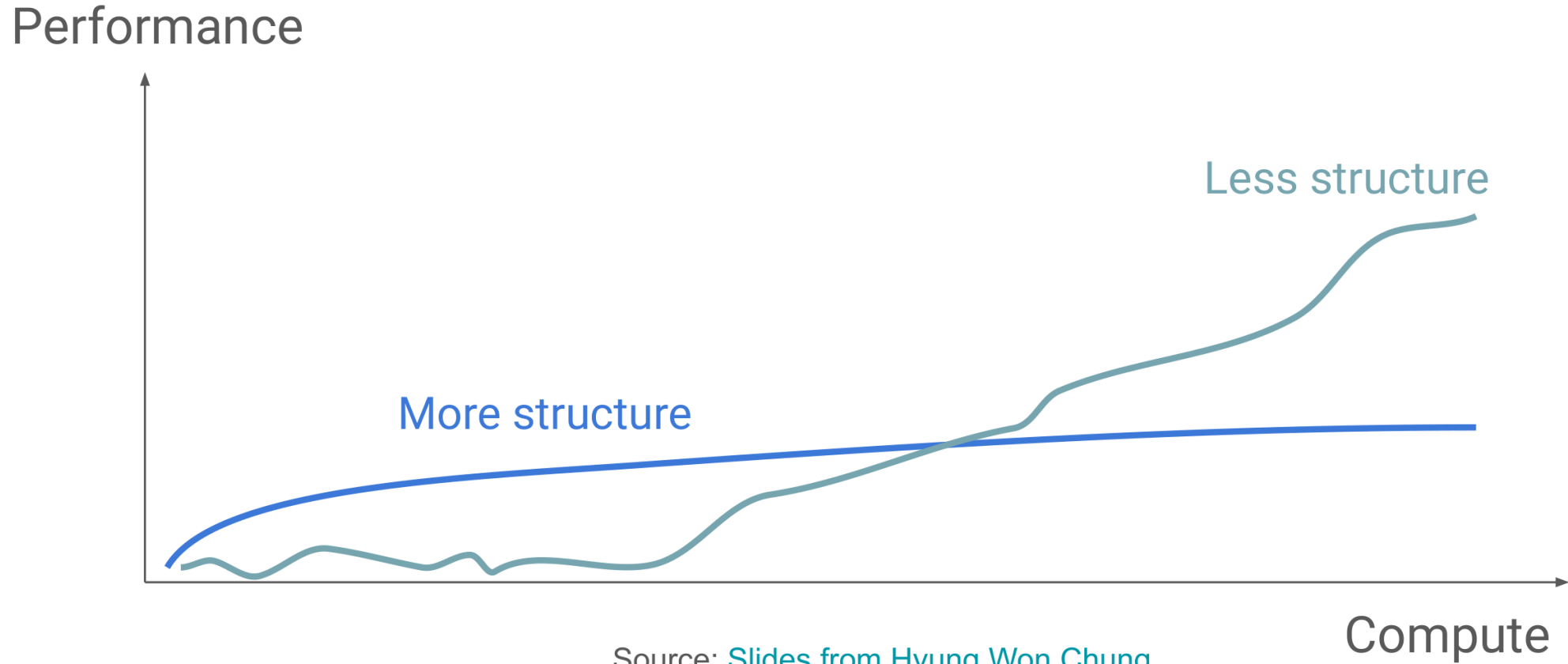


Source: [WAIC Slides from Rich Sutton](#)

Roughly 10x
compute for a
fixed cost every
5 years.

We need
**scalable
methods** that
can better
**leverage
computation**

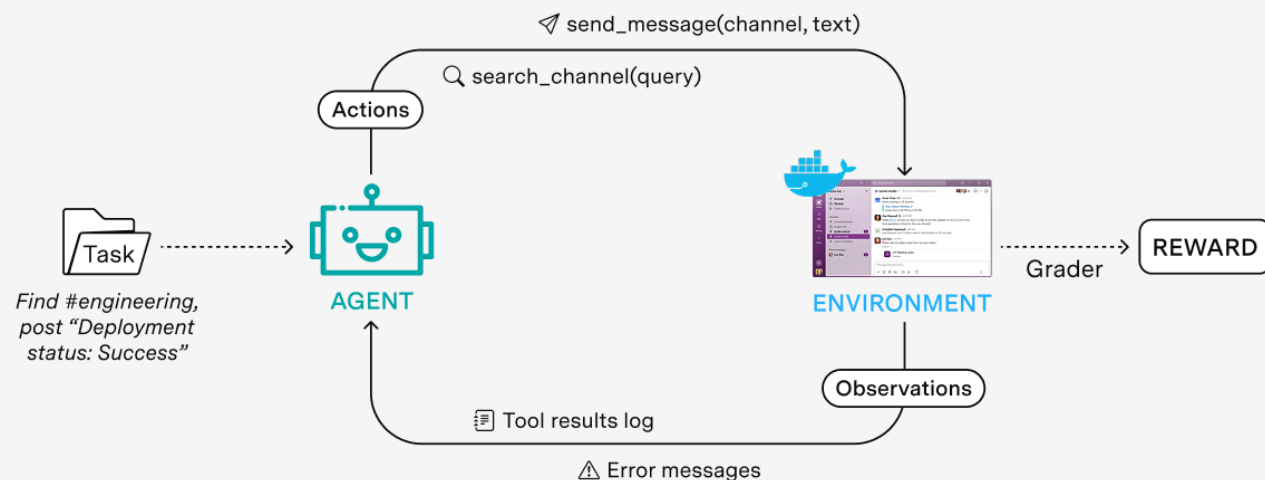
Reminder: The Bitter Lesson



We need **scalable methods** that can better **leverage computation**

RL for reasoning LLMs: A very short primer

Example RL environment: Agent interacting with simulated Slack workspace



EPOCH AI | CC-BY

epoch.ai

General RL

Policy

Initial State

Action

State

RL for LLMs

LLM

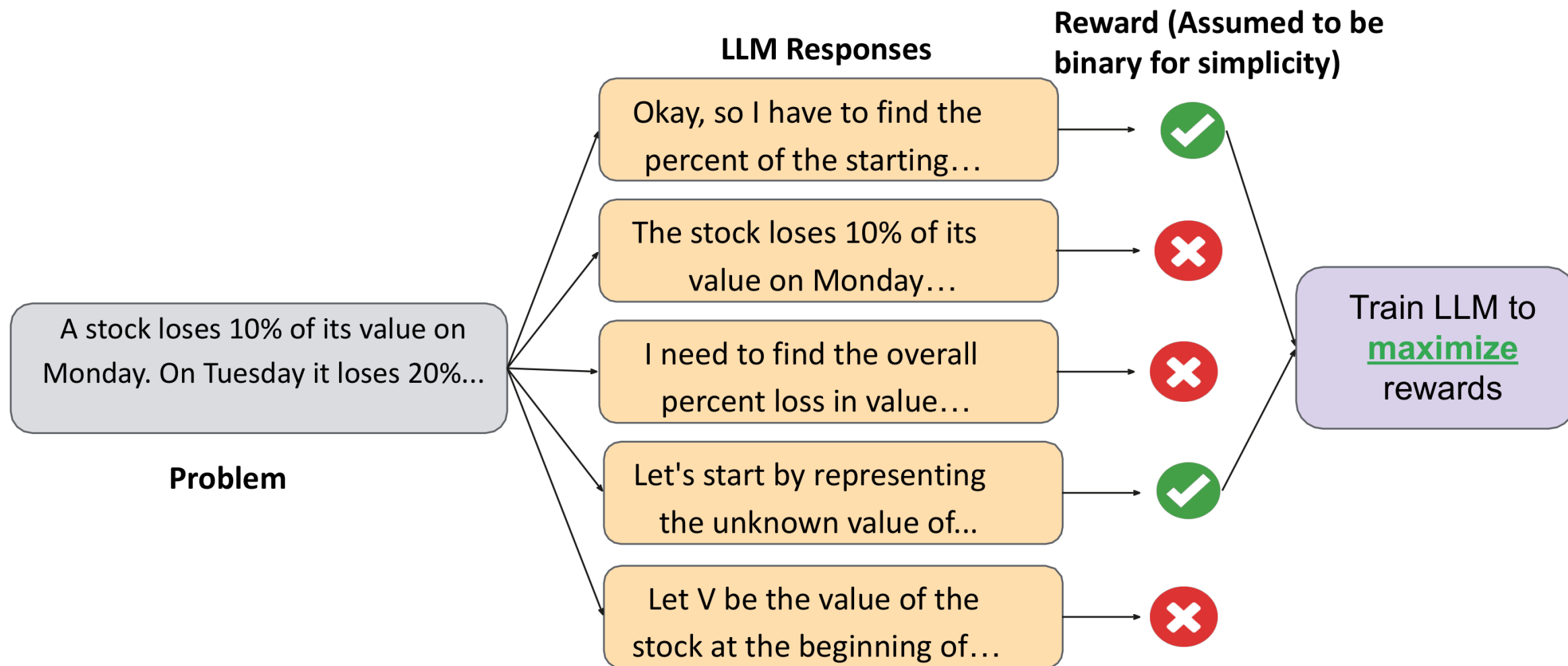
Prompt

Output (Tokens or Full Completion)

Prompt + Output

Source: Cameron Wolfe

RL for reasoning LLMs: A very short primer



RL for reasoning LLMs: A very short primer

RL Objective

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)]$$

Expand the expectation

$\int_{\tau} P(\tau|\pi_\theta) R(\tau) d\tau$

$\sum_{\tau} P(\tau|\pi_\theta) R(\tau)$

Continuous

Discrete

RL Objective

$\theta^* = \operatorname{argmax}_{\theta} J(\pi_\theta)$

Solution (Gradient Ascent)

$\theta_{k+1} = \theta_k + \alpha \nabla_{\theta} J(\pi_\theta)|_{\theta_k}$

$\longrightarrow$

RL for reasoning LLMs: A very short primer

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \int_{\tau} \nabla_{\theta} p_{\theta}(\tau) R(\tau) d\tau \\ &= \int_{\tau} p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) R(\tau) d\tau \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log p_{\theta}(\tau) R(\tau)]\end{aligned}$$

Computed on **self-generated** LLM outputs

Next-token prediction loss **weighted** by rewards!

What's the problem with this objective? (Assume binary rewards)

RL for reasoning LLMs: A very short primer

Policy Gradient

Sample / Completion

Average Reward / Baseline

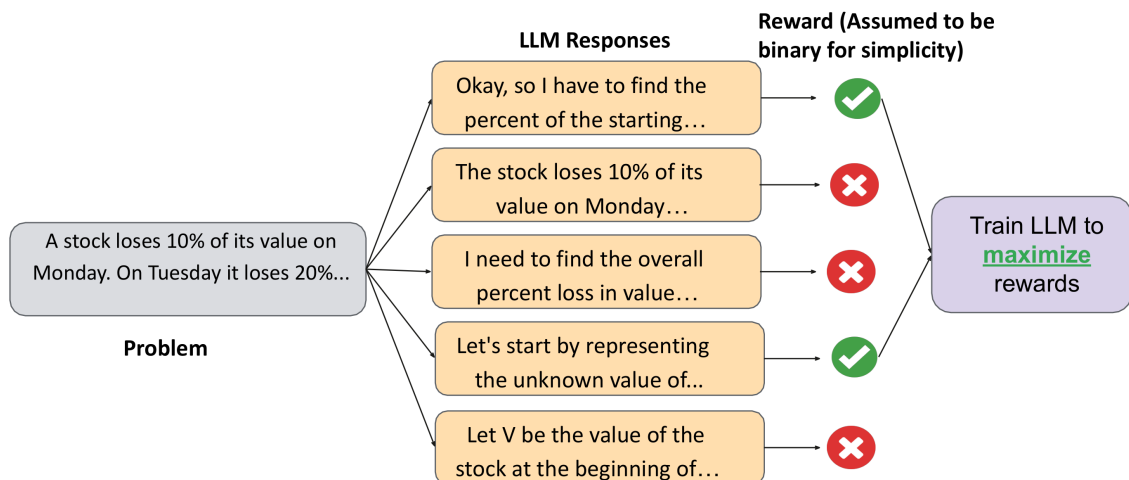
$$\boxed{\nabla_{\theta} J(\pi_{\theta})} = \mathbb{E}_{\underbrace{x \sim \mathcal{D}}_{\text{Prompt}}, \underbrace{y \sim \pi_{\theta}}_{\text{Sample / Completion}}} [\nabla_{\theta} \log \pi_{\theta}(y|x) (\underbrace{R(x, y)}_{\text{Sample Reward}} - \underbrace{\bar{R}}_{\text{Average Reward / Baseline}})]$$

Sample Reward

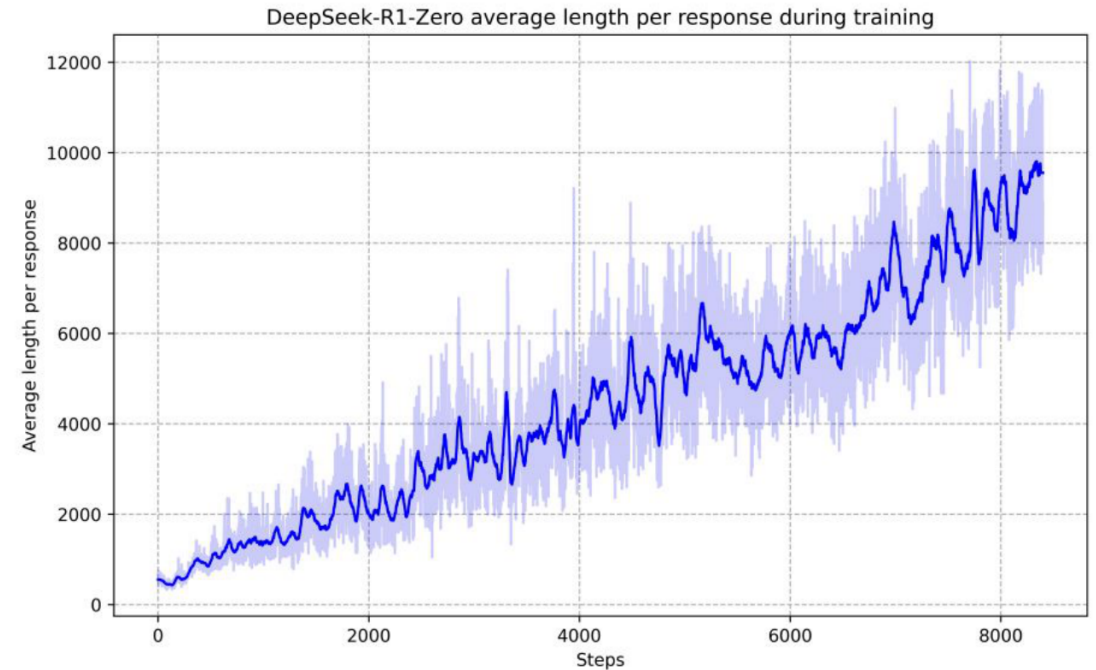
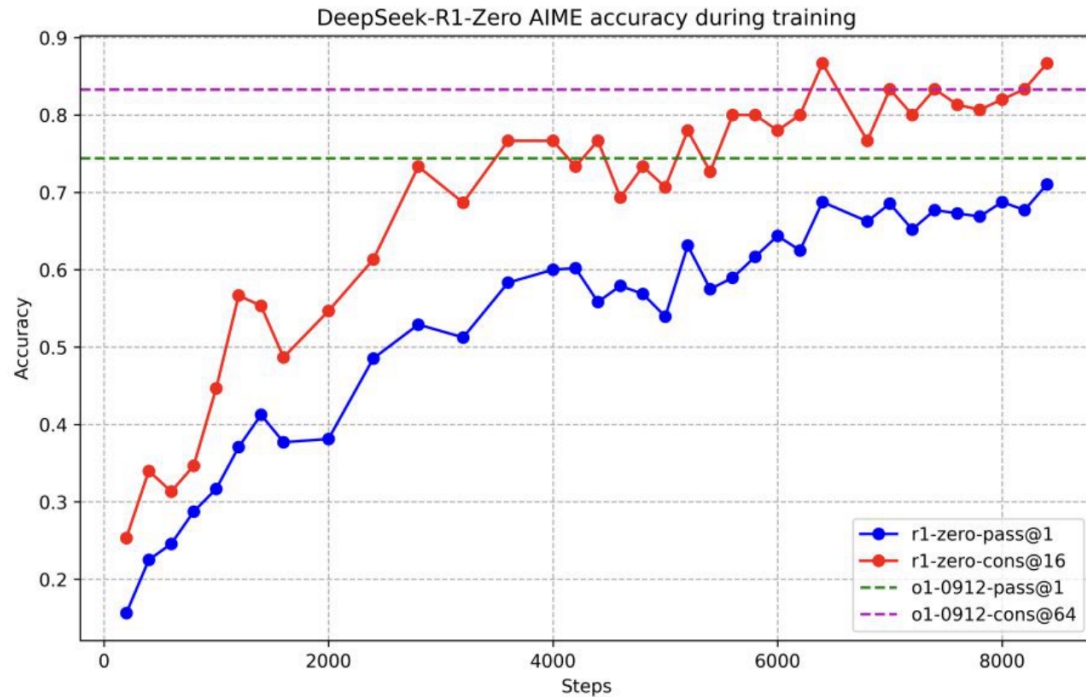
Advantage for
i-th sample

$$A(x, y_i)$$

Correct answers: Positive advantage → increase probability
Wrong answers: negative advantage → decrease probability



Reinforcement Learning Enables Test-time Scaling for LLMs



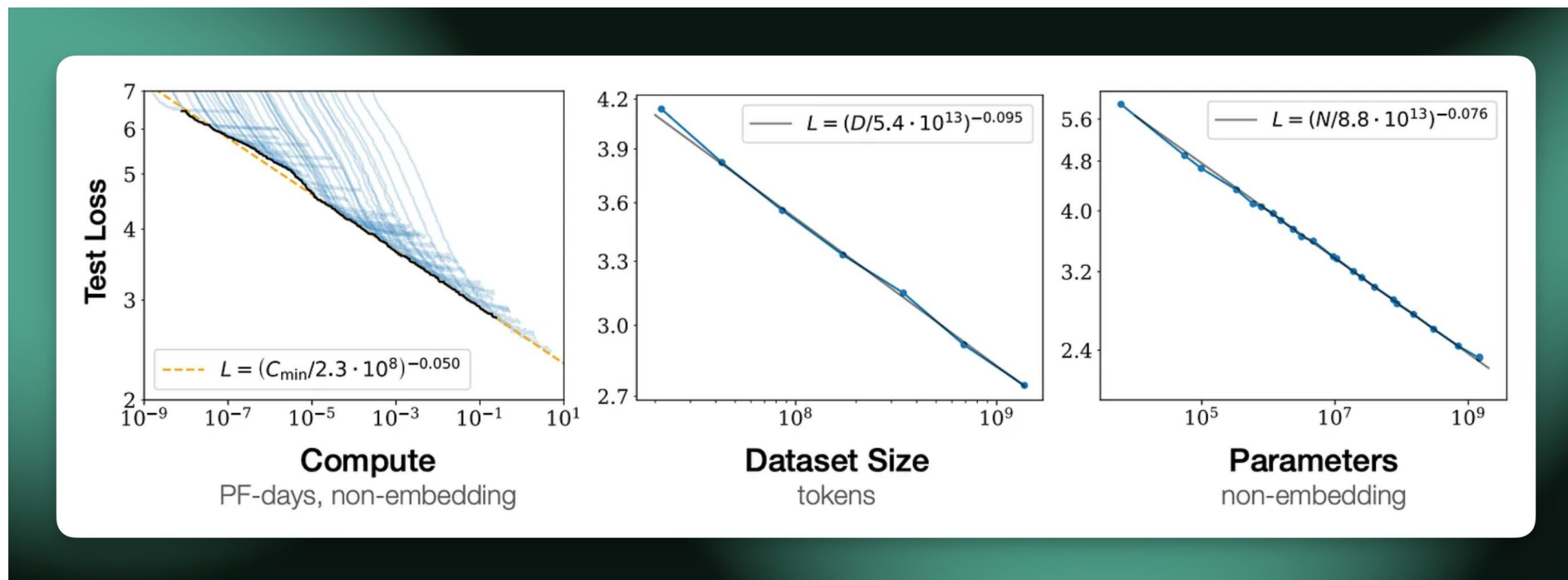
Long CoT reasoning chains contain behaviors such as: (1) Error correction, (2) Trying multiple strategies, (3) Breaking down problems into smaller steps, (4) Double Checking answers

Part 1:

A Framework for

RL Scaling

Pre-training scaling laws



Loss $\propto C^{-\alpha}$ (power law)

- Unbounded — always improving
- Relates downstream test loss to pre-training compute
- Well-understood next-token prediction objective

Extrapolation with Scaling laws

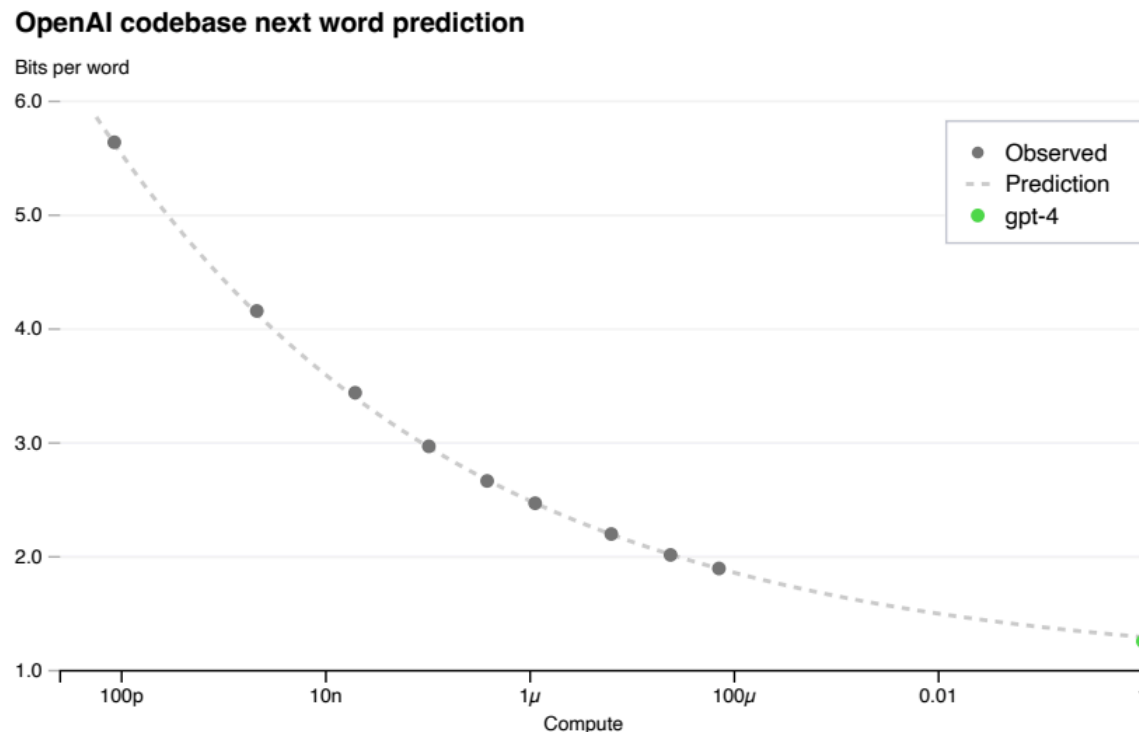


Figure 1. Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across different amounts of training compute. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4’s final loss. The x-axis is training compute normalized so that GPT-4 is 1.

What about scaling **RL training**?

Pre-training: well-understood scaling laws. RL training for reasoning:

Many algorithms

GRPO, DAPO, PPO,
CISPO, GSPO, IcePop, ...

Many design choices

Normalization, curriculum...

No scaling framework

Can't predict from
small runs

Misleading early results

Best at 5K $\neq$ best
at 100K GPU-hrs

Can we bring the same predictability to RL training?

Why does this **matter**?

For Researchers

Evaluate proposals cost-effectively.
Run 10K GPU-hrs instead of 100K.

For Practitioners

Predict performance before committing.
Choose the right recipe for your budget.

For the Field

Brings RL closer to the predictability of
pre-training. Makes RL scaling a science.

This talk

400,000+ GPU-hours of systematic experiments

Llama-3 8B + Llama-4 Scout 17Bx16 MoE • Polaris-53K math • Nvidia GB200

1

A Framework for RL Scaling

Why sigmoids, not power laws? Ceiling vs efficiency.

2

Building ScaleRL

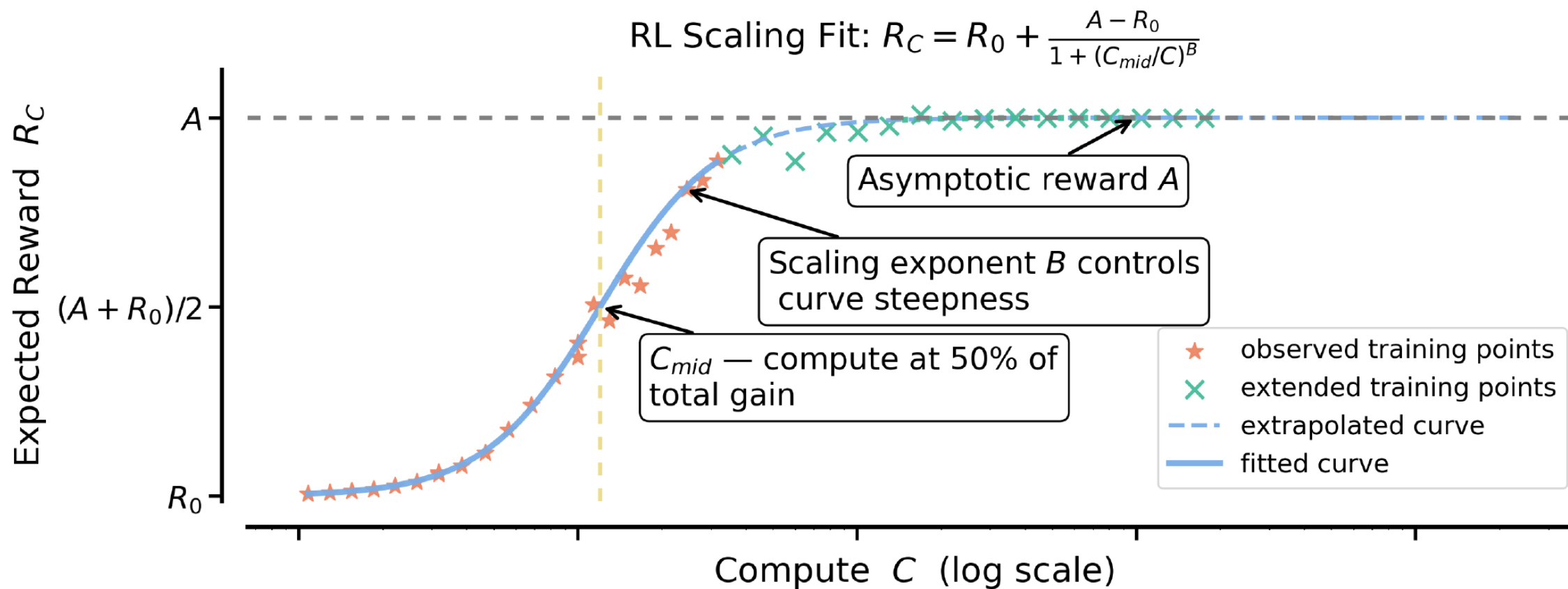
6 design choices, each ablated at scale

3

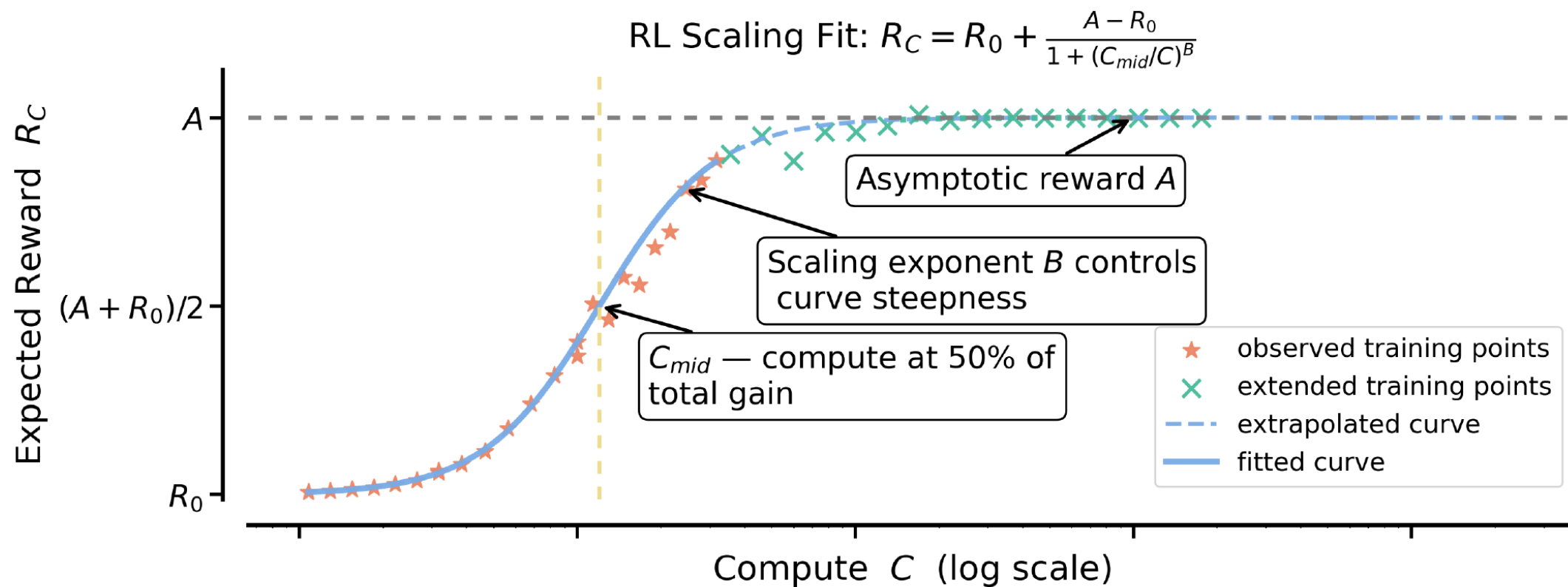
Scaling Along Different Axes

MoE 6x efficiency, gen length, batch, multi-task

RL scaling for LLMs is predictable!



RL scaling **differs** from pre-training



A

Asymptotic Ceiling
How high?

B

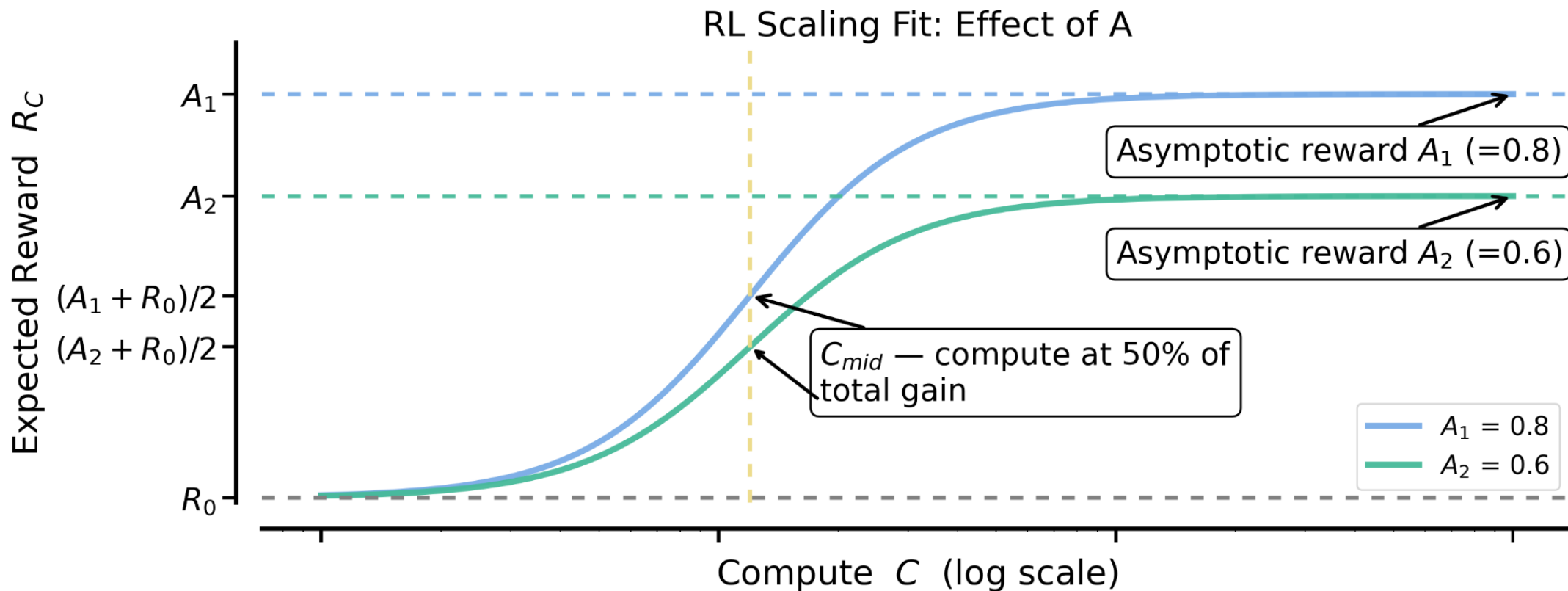
Compute Efficiency
How fast?

C_{mid}

Inflection
When saturate?

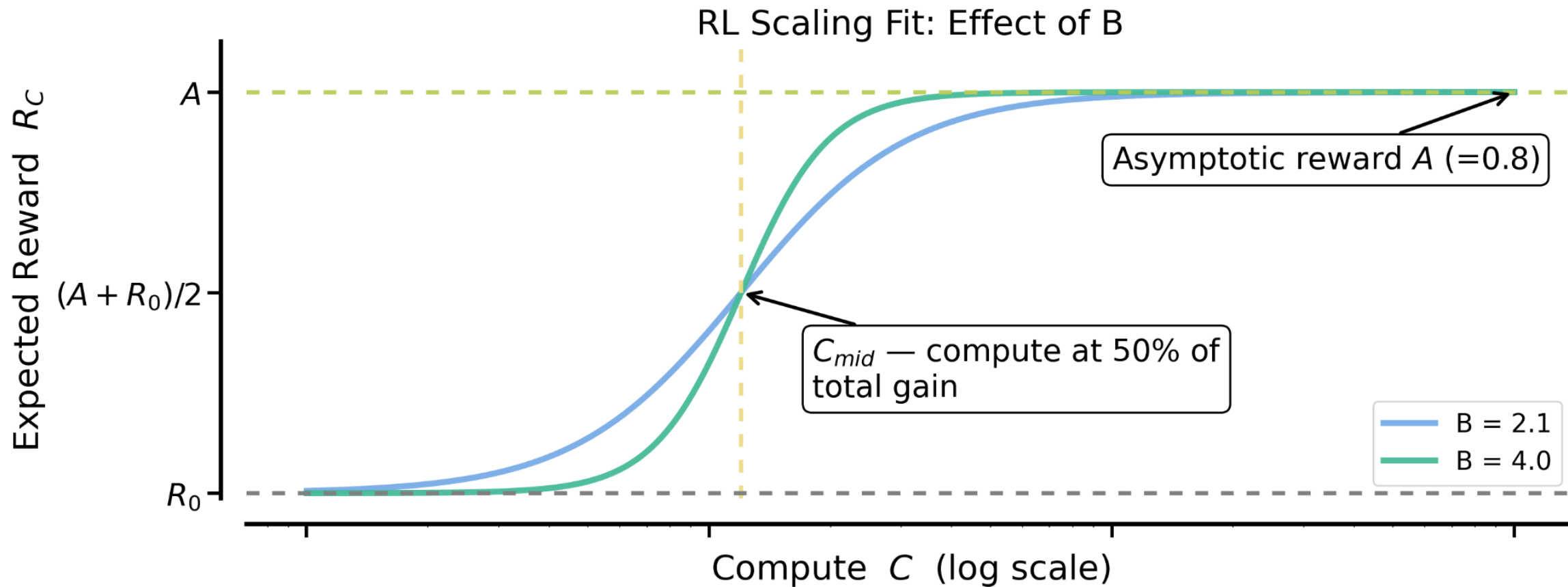
Understanding RL Scaling: Asymptotic Ceiling

A

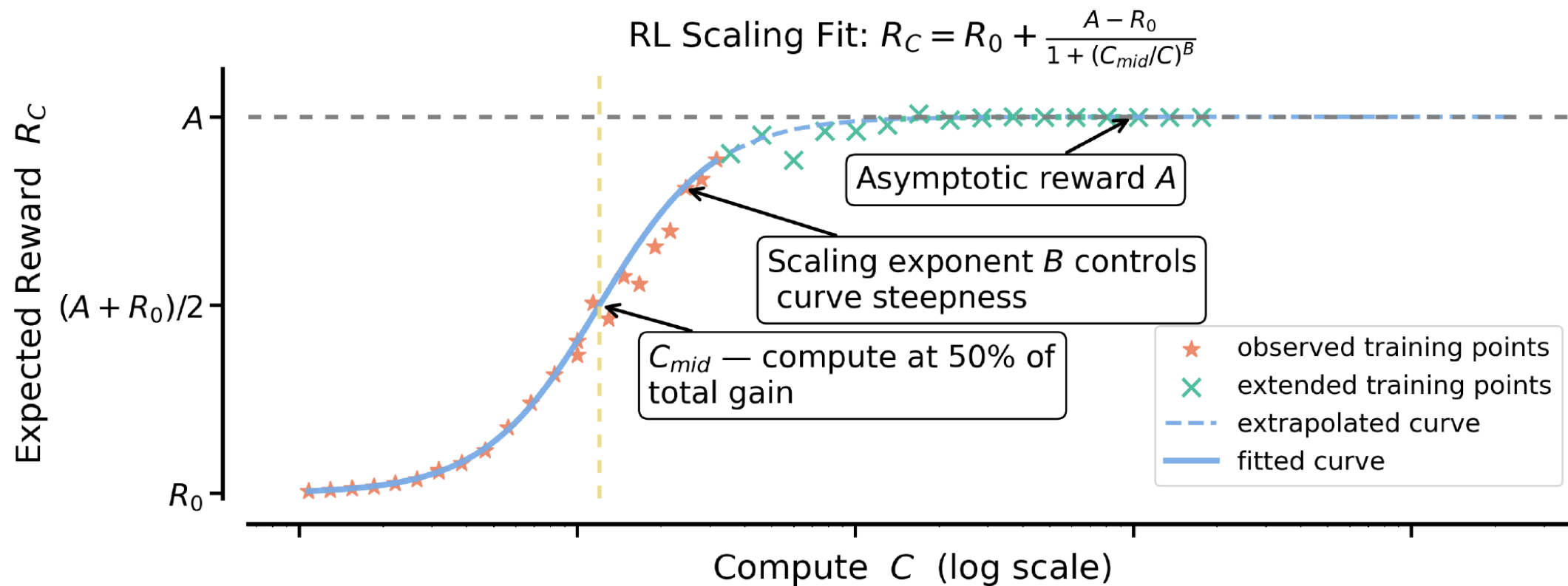


Understanding the functional form: Compute Efficiency

B



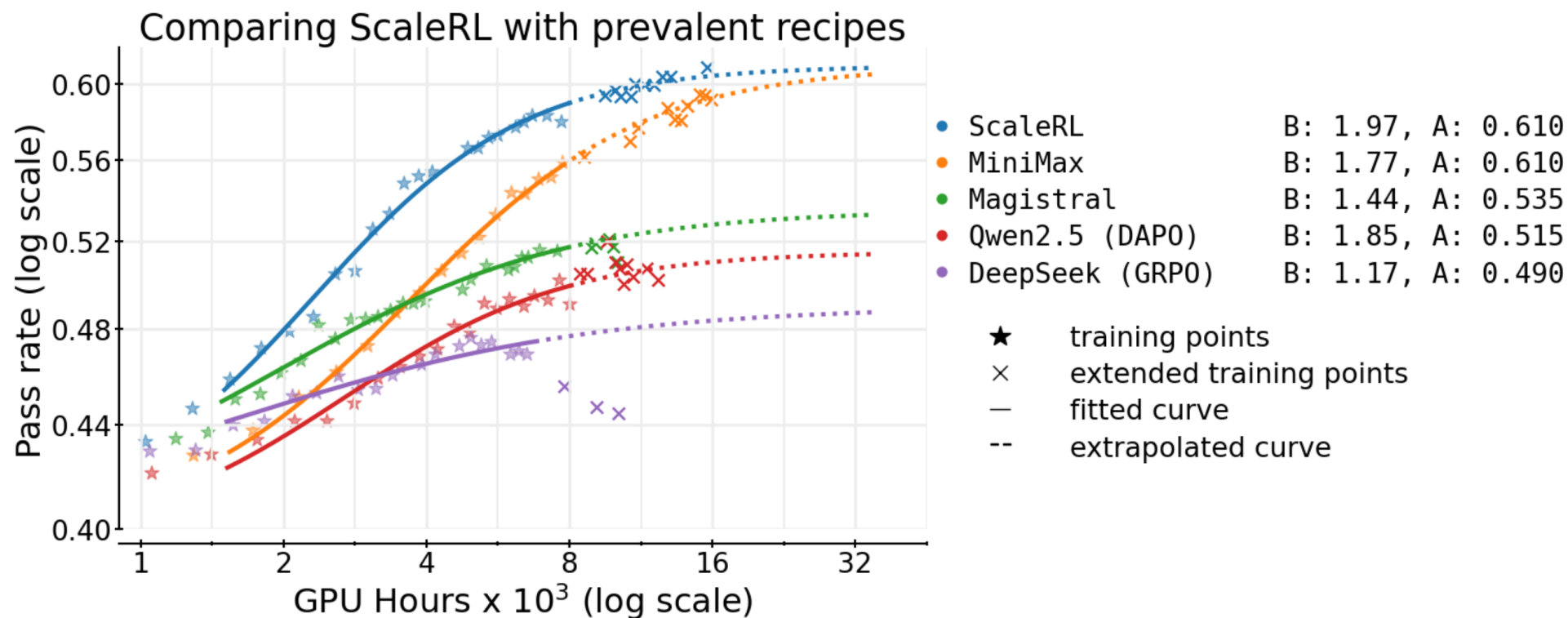
RL scaling **differs** from pre-training



Sigmoid! (bounded 0–1)

- Metric is pass rate / accuracy $\leq 100\%$
- Finite training problems
- Many interacting RL algorithmic choices

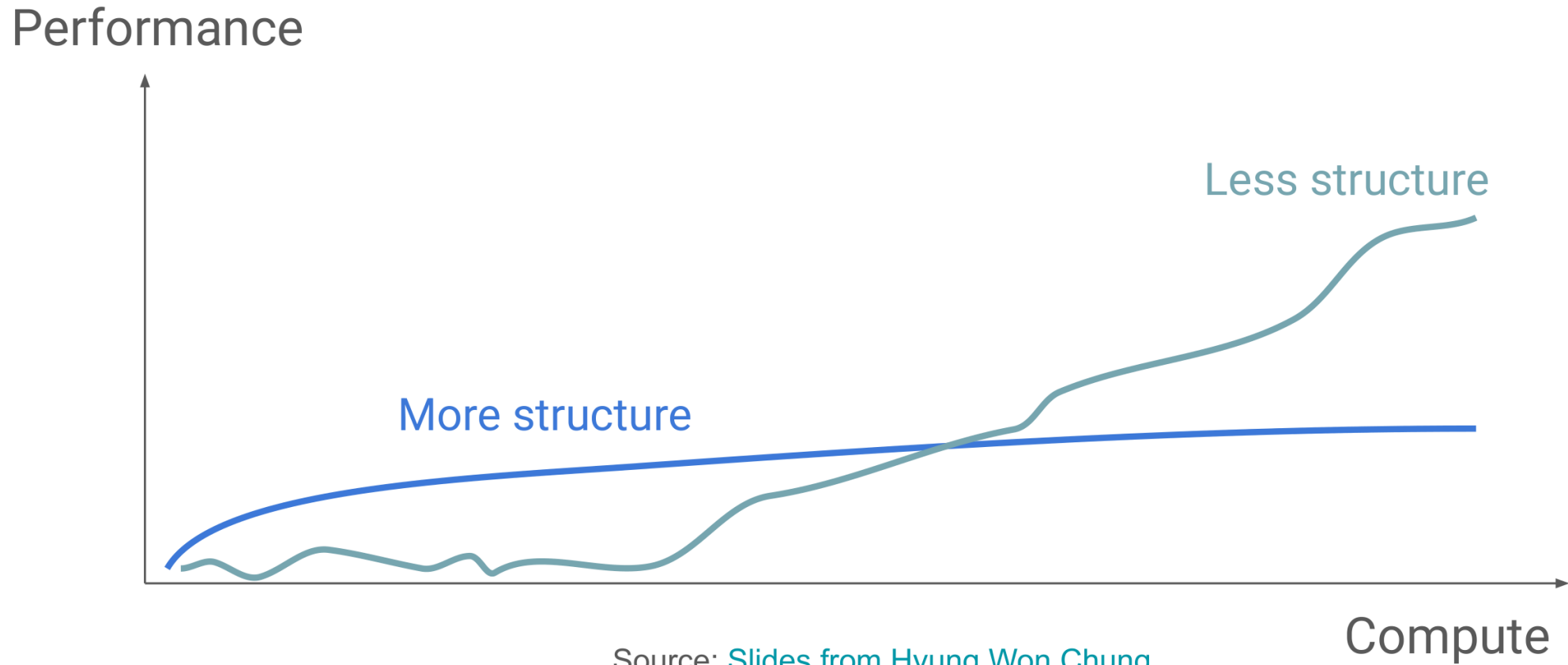
Different RL recipes, different ceilings



Algorithmic choices fundamentally limits what you can achieve.

You can't just train longer to fix a bad recipe!

Reminder: The Bitter Lesson



We need **scalable methods** that can better **leverage computation**

The best method at 5K GPU-hrs is **NOT** the best at 100K hrs

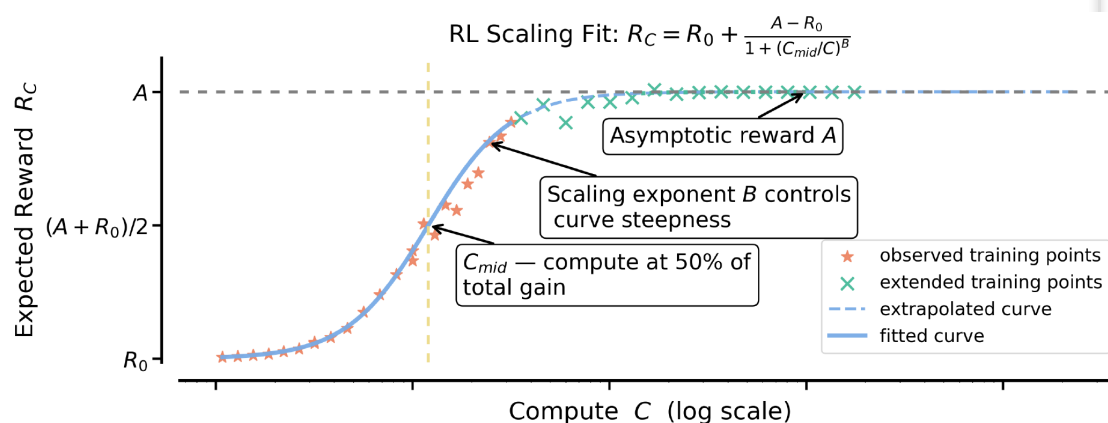
Without framework:

"Method X beats Y at 5K GPU-hrs.
Let's scale X to 100K!"

- ✗ X has higher B but lower ceiling
- ✗ Plateaus early, wasting 100K GPU-hrs

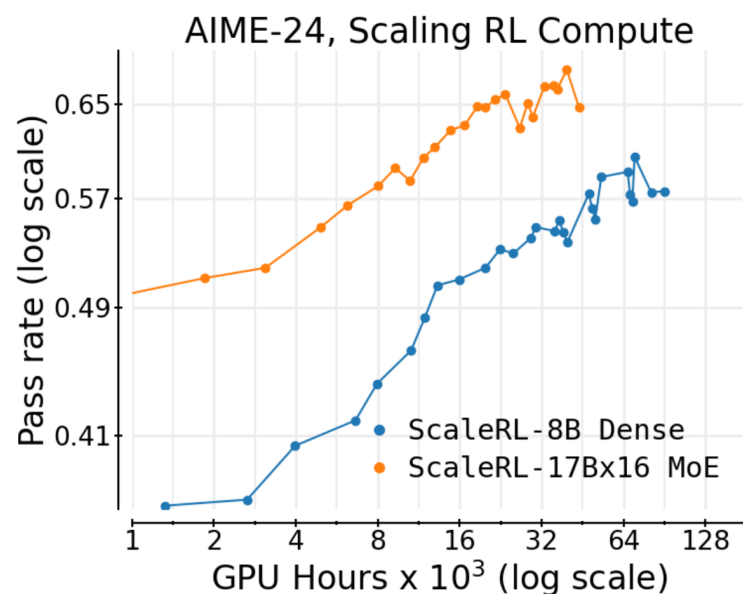
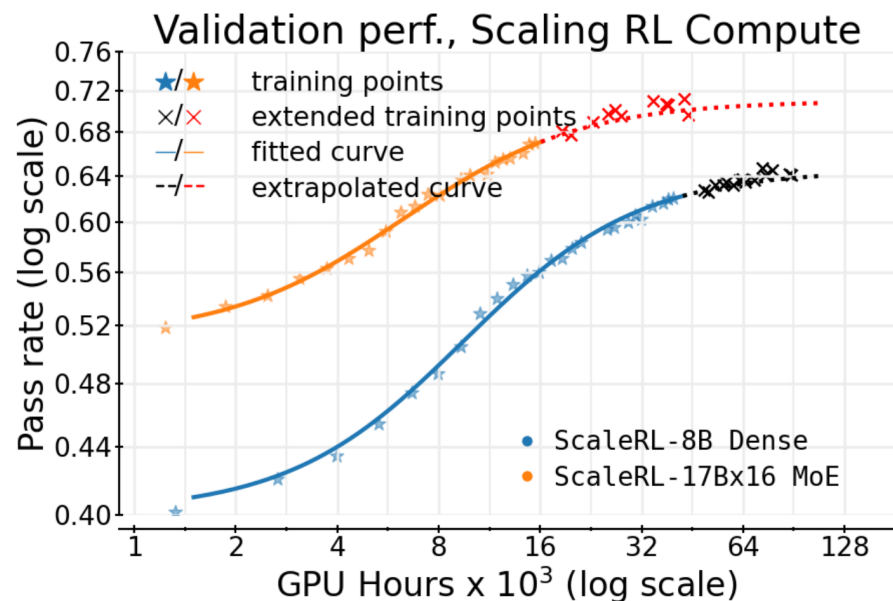
With framework:

"X has $B=2.1$, $A=0.55$.
Y has $B=1.5$, $A=0.62$.
Y will win at scale."



- ✓ Fit scaling curve on cheap 10K runs
- ✓ Predict the winner at 100K+

Predicting 100K GPU-hours run!



Fit sigmoid on early data (16–50K GPU-hrs)

→ Predict final performance at 2x-3x extrapolated GPU-hrs!

Things we do for Science: A 100,000 GPU Hour Run!

Things we learned so far:

- RL training curves follow sigmoid scaling patterns (not power laws).
- Different recipes have different performance ceilings (A).
- We can predict large-scale performance from small-scale runs!

Part 2:

Building the

ScaleRL Recipe

The Complete **ScaleRL** Recipe

• ceiling (A)

• efficiency (B)

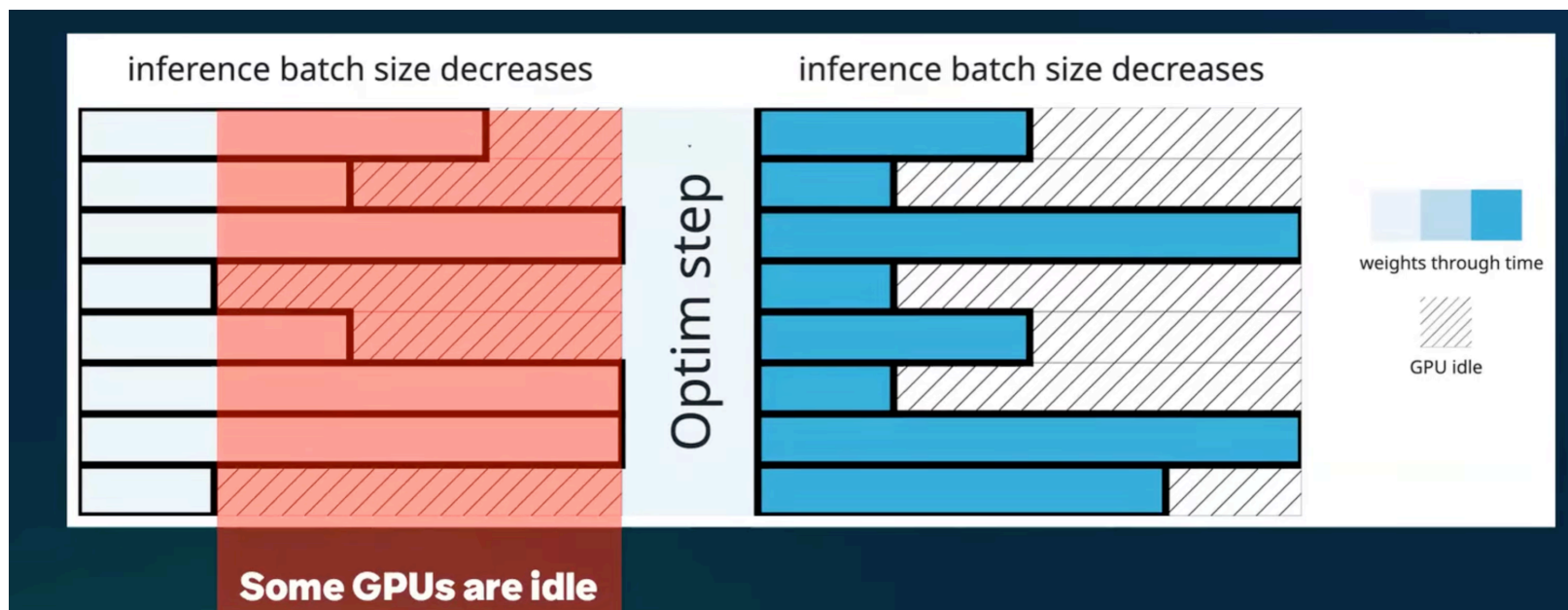
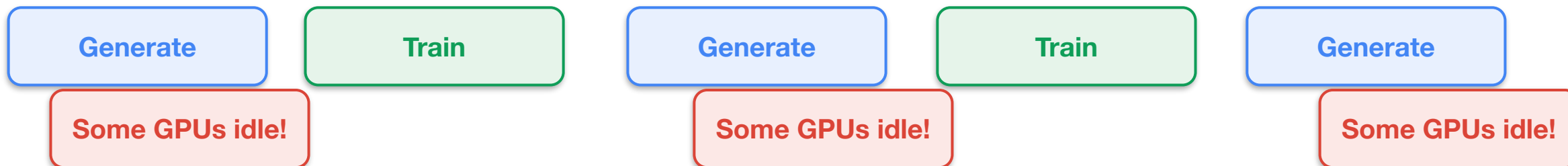
✓	CISPO loss	Truncated IS + vanilla gradient
✓	FP32 logits	Full precision at LM head
✓	PipelineRL-8	Async off-policy, 8 steps staleness
✓	Prompt-level loss	Equal prompt contribution
✓	Batch-level normalization	Stable advantage estimates
✓	Zero-variance filtering	Skip uninformative prompts
✓	No-positive resampling	Drop >90% solve-rate problems
✓	Forced interruptions	Controlled thinking budget

How RL design choices affect Scaling

● Loss Function	GRPO vs DAPO vs CISPO	→ ceiling (A)
● Training–Inference Discrepancy	FP32 logits at LM head	→ ceiling (A)
● Off-Policyness / Staleness	Off-policy-RL vs PipelineRL	→ efficiency (B)
● Normalization	Prompt-level loss + batch-level advantage	→ efficiency (B)
● Adaptive Sampling	Zero-variance filtering + pass-rate resampling	→ efficiency (B)
● Length Control	Forced interruptions for thinking budget	→ efficiency (B)

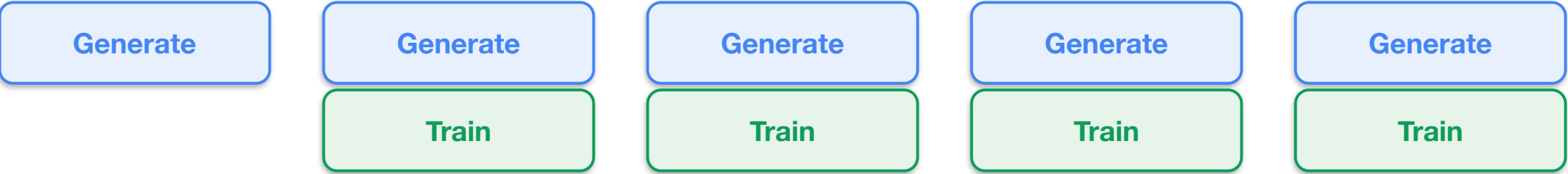
RL for LLMs: On-Policy RL is inefficient!

Synchronous On-Policy RL: GPUs sit idle significant % of the time

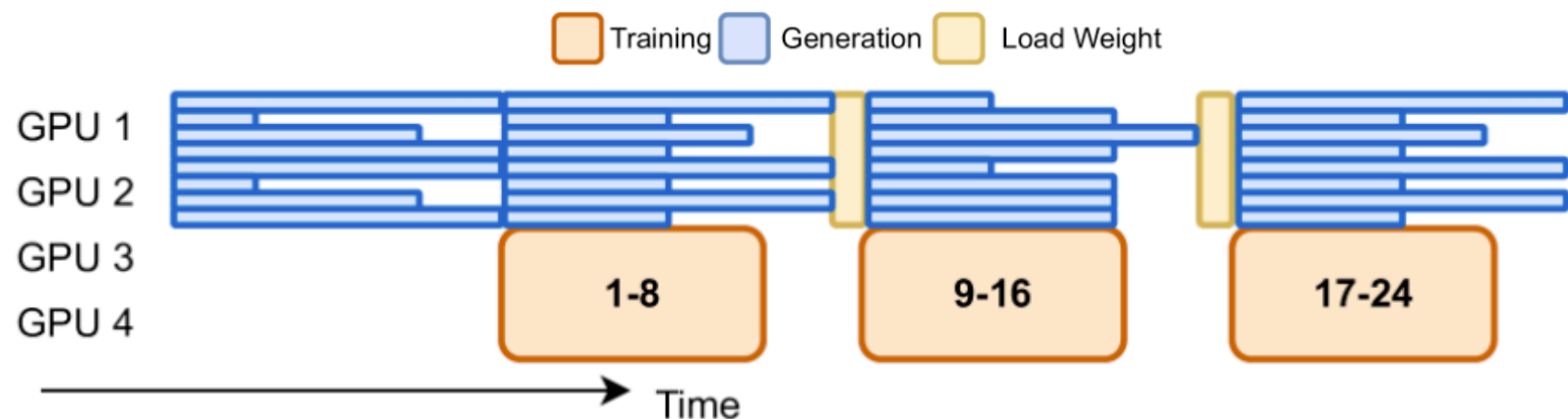


RL for LLMs: Async RL for efficiency

Async RL: generate & train simultaneously

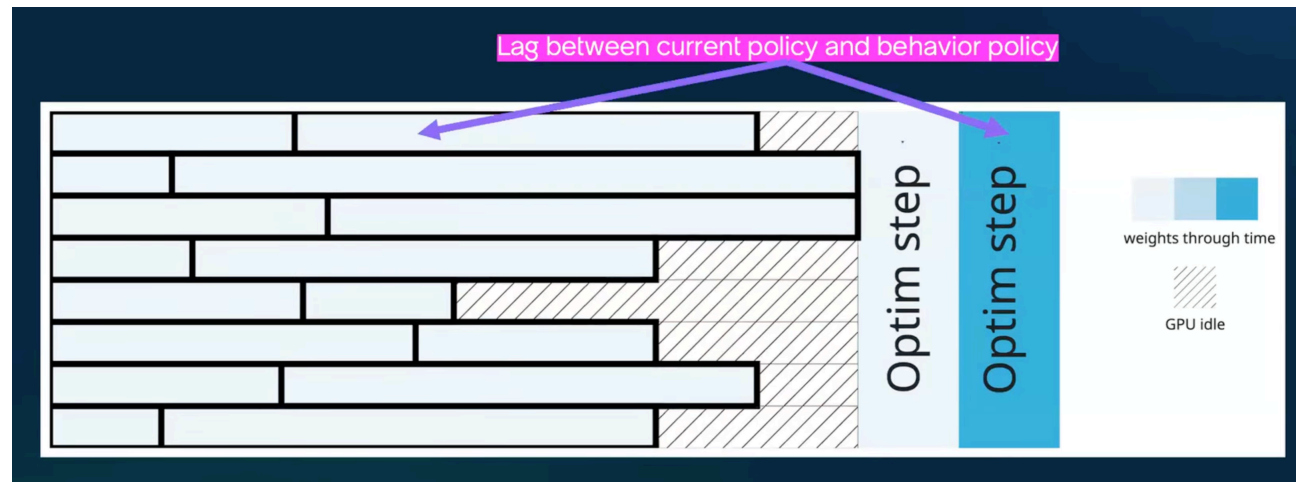
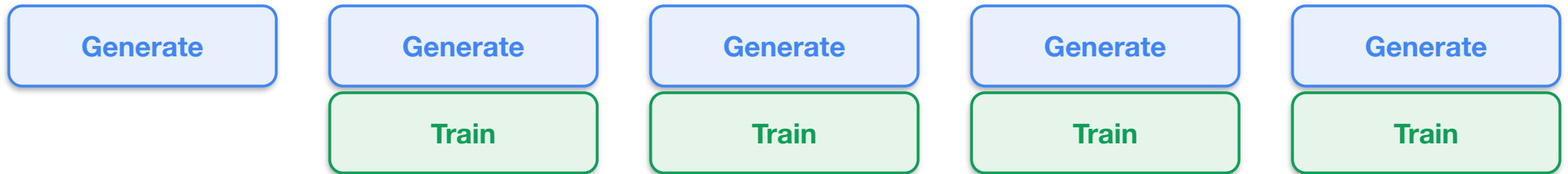


[ASYNCHRONOUS RLHF](#): FASTER AND MORE EFFICIENT OFF-POLICY RL FOR LANGUAGE MODELS



RL for LLMs: Async RL is off-policy!

Async RL: generate & train simultaneously



Off-Policy Training:

The data generating policy is stale (old) compared to training policy.

Async RL: PipeLine RL vs PPO-Off-Policy

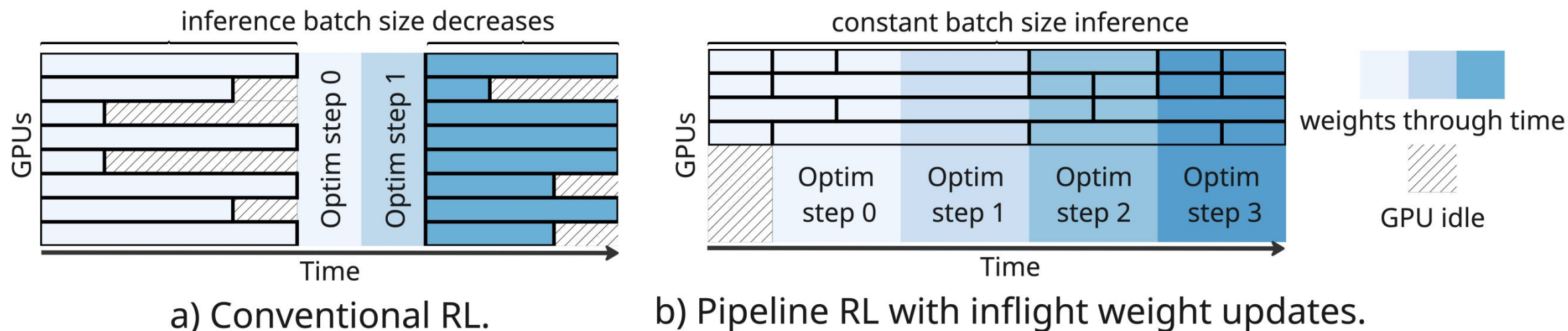
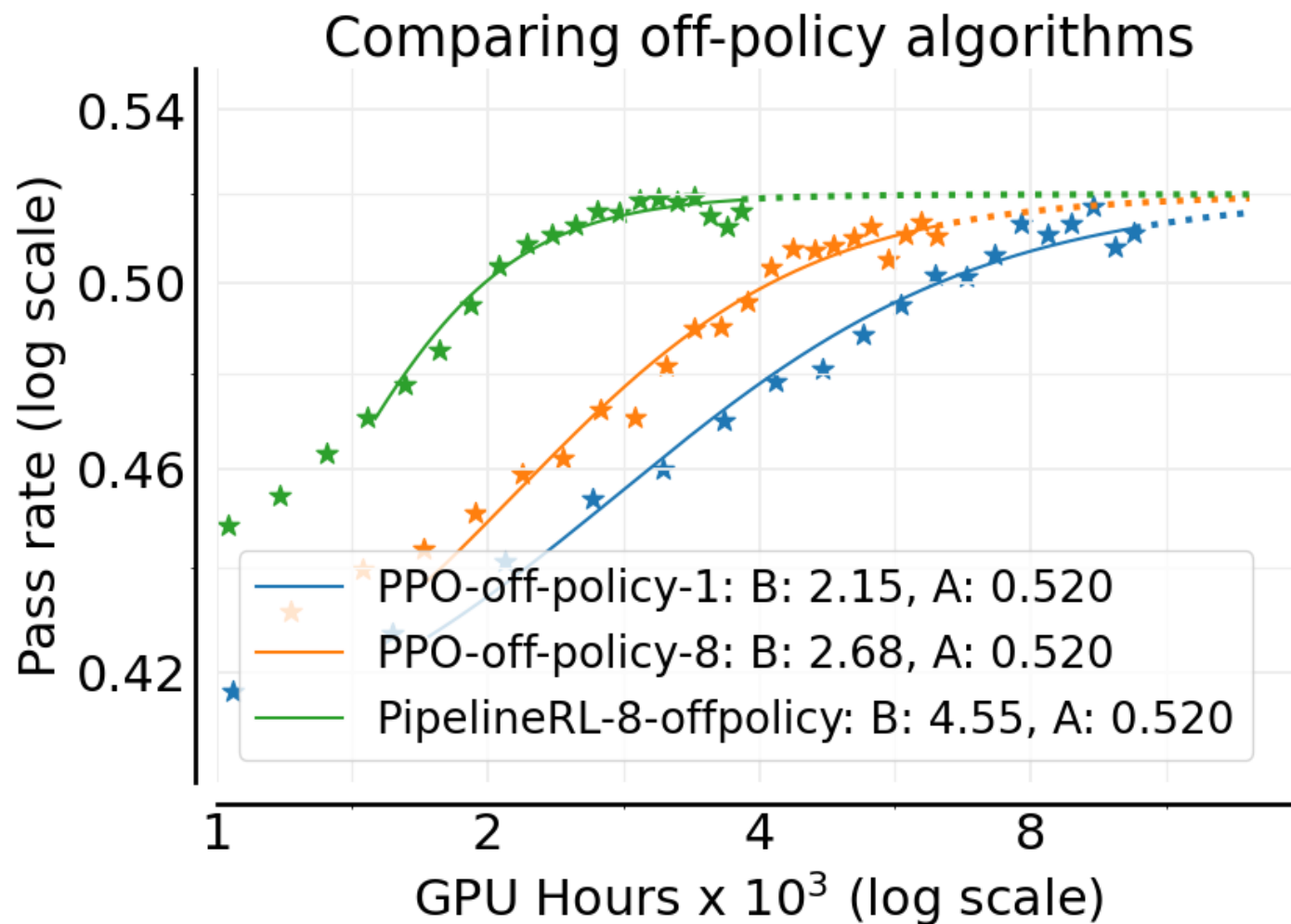


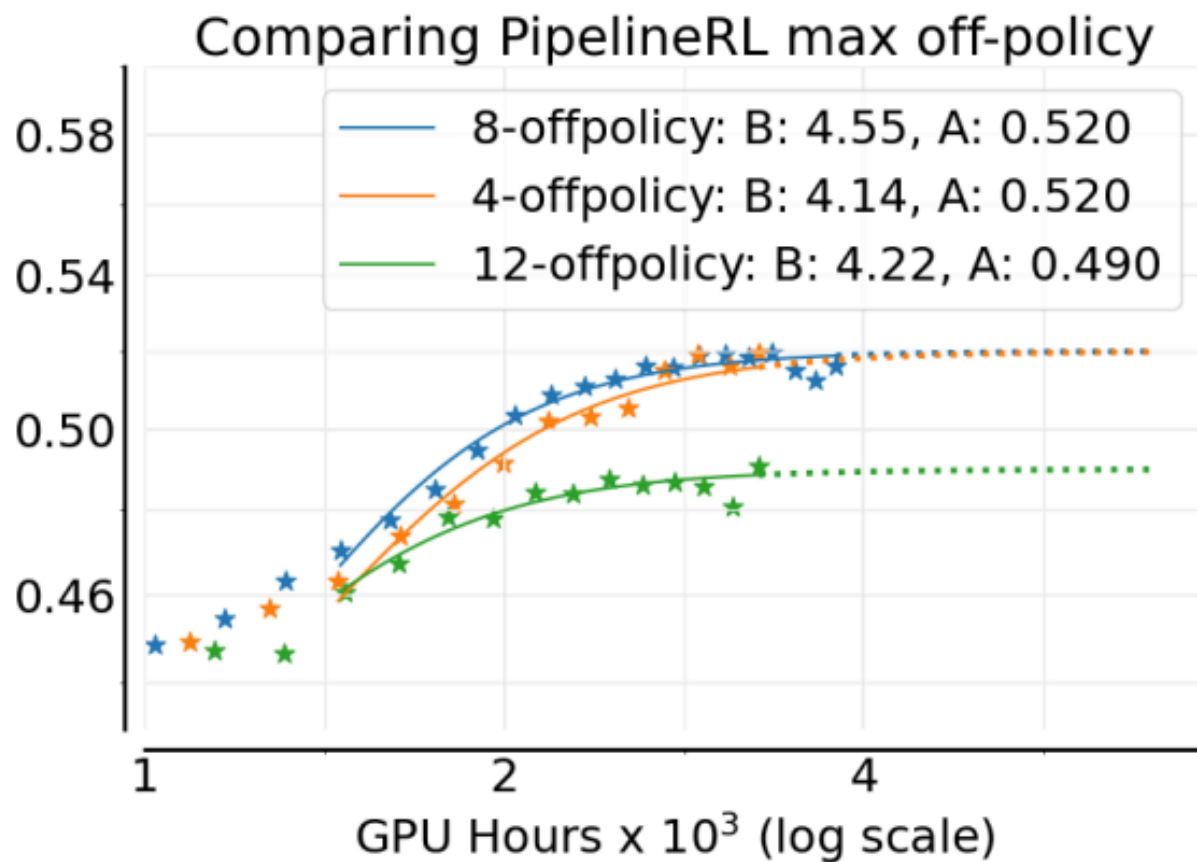
Figure 1: **a)** Conventional RL alternates between using all the GPUs for generation and then training. **b)** PipelineRL runs generation and training concurrently, always using the freshest model weights for generations thanks to the in-flight weight updates.

[Source: Pipeline RL](#)

Async RL: Pipeline RL vs. PPO-Off-Policy



Async RL: Off-Policyyness in Pipeline RL



RL for LLMs: Handling Off-policyyness via Importance Sampling

RL Objective

$$\overbrace{J(\pi_\theta)} = \mathbb{E}_{a \sim \pi_{\text{learner}}(\theta)}[R(a)] = \mathbb{E}_{a \sim \pi_{\text{sampler}}(\theta)} \left[\underbrace{\frac{\pi_{\text{learner}}(a, \theta)}{\pi_{\text{sampler}}(a, \theta)}}_{\text{importance ratio}} \cdot R(a) \right]$$

Current Policy

Policy Ratio
(different from reward r_t)

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$$

Old Policy (before any optimization steps / epochs)

Policy Gradient

$$\boxed{\nabla_\theta J(\pi_\theta)} = \mathbb{E}_{a \sim \pi_{\text{sampler}}(\theta)} \left[\frac{\pi_{\text{learner}}(a, \theta)}{\pi_{\text{sampler}}(a, \theta)} \cdot R(a) \cdot \nabla_\theta \log \pi_{\text{learner}}(a, \theta) \right]$$

GRPO / PPO Loss: Handling Off-policyyness

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \min \left(\overbrace{r_t(\theta) A(s_t, a_t)}^{\text{Unclipped Objective}}, \overbrace{\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) A(s_t, a_t)}^{\text{Clipped Objective}} \right) \right]$$

	$p_t(\theta) > 0$	A_t	Return Value of $\min$	Objective is Clipped	Sign of Objective	Gradient
1	$p_t(\theta) \in [1 - \epsilon, 1 + \epsilon]$	+	$p_t(\theta)A_t$	no	+	✓
2	$p_t(\theta) \in [1 - \epsilon, 1 + \epsilon]$	-	$p_t(\theta)A_t$	no	-	✓
3	$p_t(\theta) < 1 - \epsilon$	+	$p_t(\theta)A_t$	no	+	✓
4	$p_t(\theta) < 1 - \epsilon$	-	$(1 - \epsilon)A_t$	yes	-	0
5	$p_t(\theta) > 1 + \epsilon$	+	$(1 + \epsilon)A_t$	yes	+	0
6	$p_t(\theta) > 1 + \epsilon$	-	$p_t(\theta)A_t$	no	-	✓

$A > 0$

L^{CLIP}

r

$A < 0$

r

L^{CLIP}

Policy Ratio (different from reward r_t) $\rightarrow r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$

Current Policy

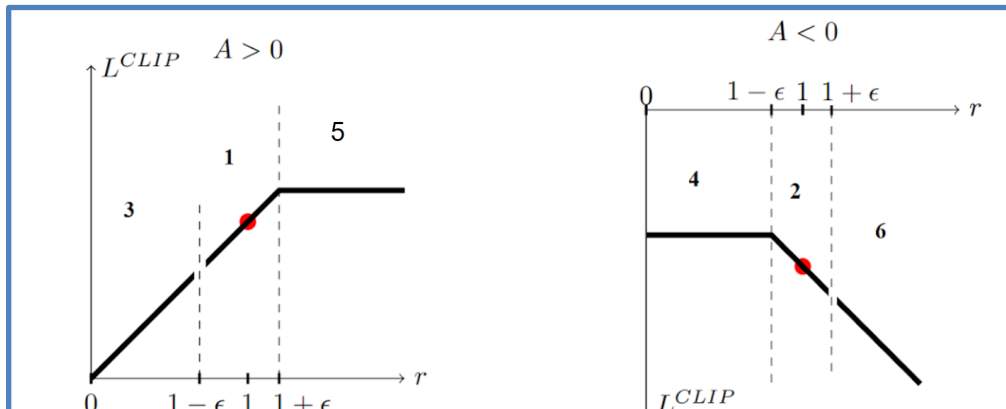
Old Policy (before any optimization steps / epochs)

DAPO Loss: GRPO + Asymmetric Clipping

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \min \left(\overbrace{r_t(\theta) A(s_t, a_t)}^{\text{Unclipped Objective}}, \overbrace{\text{clip}(r_{i,t}(\theta), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}) \cdot A(s_t, a_t)}^{\text{Clipped Objective}} \right) \right]$$

Clipping Range

Asymmetric



Policy Ratio (different from reward r_t) $\rightarrow r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$

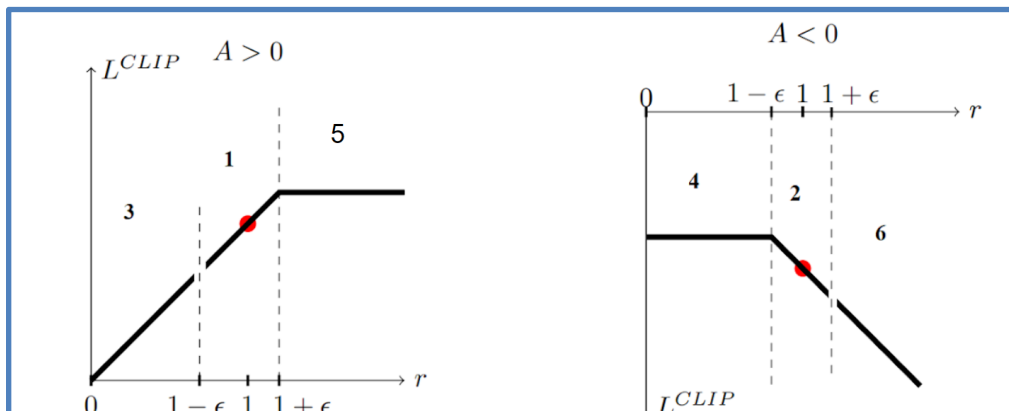
Current Policy $\downarrow$

Old Policy (before any optimization steps / epochs) $\uparrow$

DAPO Loss: GRPO + Asymmetric Clipping

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \min \left(\overbrace{r_t(\theta) A(s_t, a_t)}^{\text{Unclipped Objective}}, \overbrace{\text{clip}(r_{i,t}(\theta), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}) \cdot A(s_t, a_t)}^{\text{Clipped Objective}} \right) \right]$$

Clipping Range



Policy Ratio (different from reward r_t) $\rightarrow r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$

Current Policy $\downarrow$

Old Policy (before any optimization steps / epochs) $\uparrow$

GSPO Loss: Seq-level IS, Asym. Clip

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \min \left(\overbrace{r_t(\theta) A(s_t, a_t)}^{\text{Unclipped Objective}}, \overbrace{\text{clip}\left(r_{i,t}(\theta), 1 - \varepsilon_{\text{low}}, 1 + \varepsilon_{\text{high}}\right) \cdot A(s_t, a_t)}^{\text{Clipped Objective}} \right) \right]$$

Clipping Range └────────┘

Sequence-level Importance Sampling (biased)

Policy Ratio
(different from reward r_t)

→ $r_t(\theta)$:

$$\left(\frac{\pi_{\theta}(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)} \right)^{\frac{1}{|y_i|}} = \exp \left(\frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \log \frac{\pi_{\theta}(y_{i,t}|x, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x, y_{i,<t})} \right)$$

TIS / CISPO: Handling Off-policyyness

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \overbrace{\min(r_t(\theta), C)}^{\text{Truncated Importance Ratio}} A(s_t, a_t) \nabla_{\theta} \log \pi_{\text{learner}}(a_t \mid \theta) \right]$$

No masking, only clipping

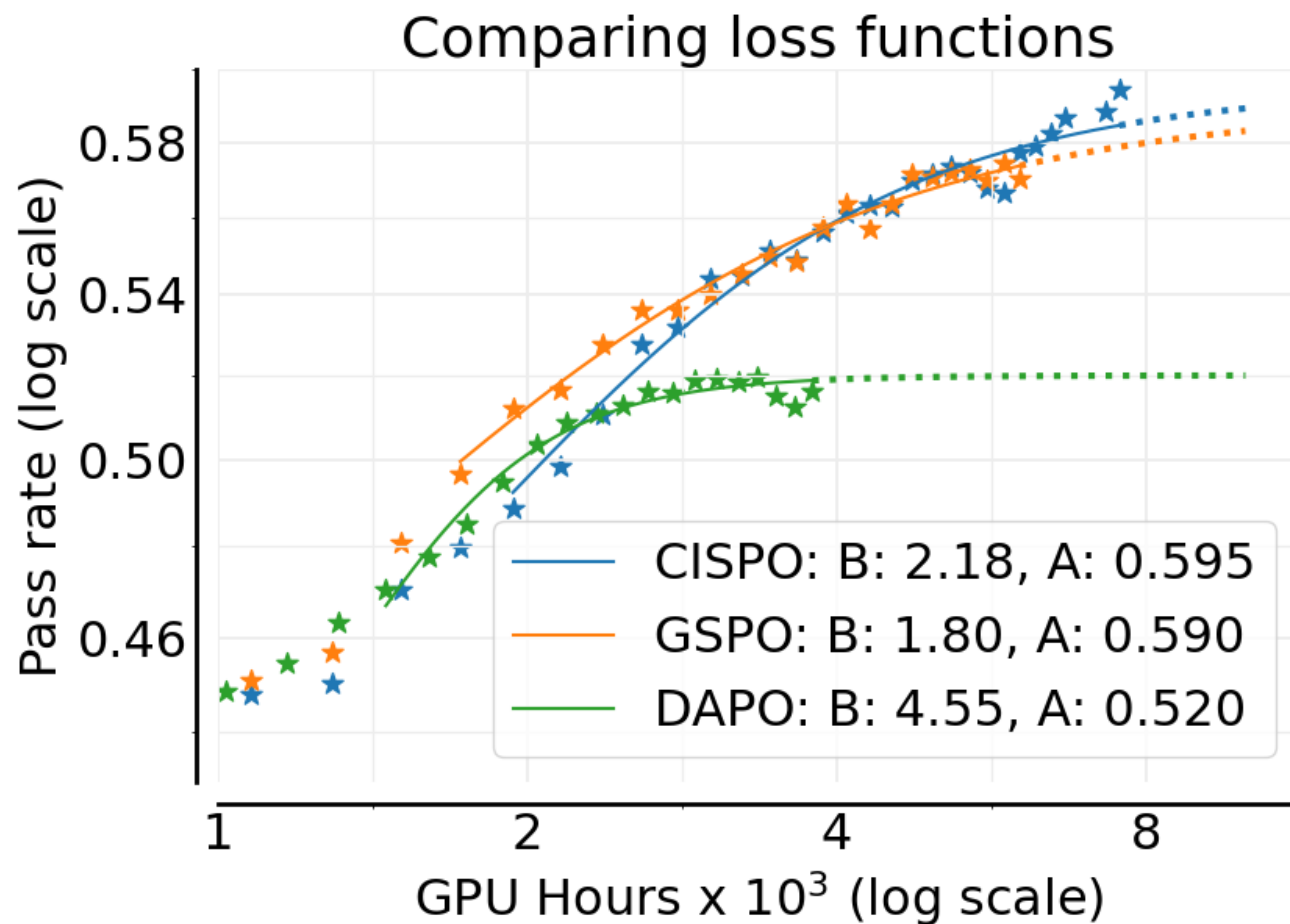
Policy Ratio
(different from reward r_t) $\rightarrow r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$

Current Policy $\rightarrow \pi_{\theta}(a_t | s_t)$

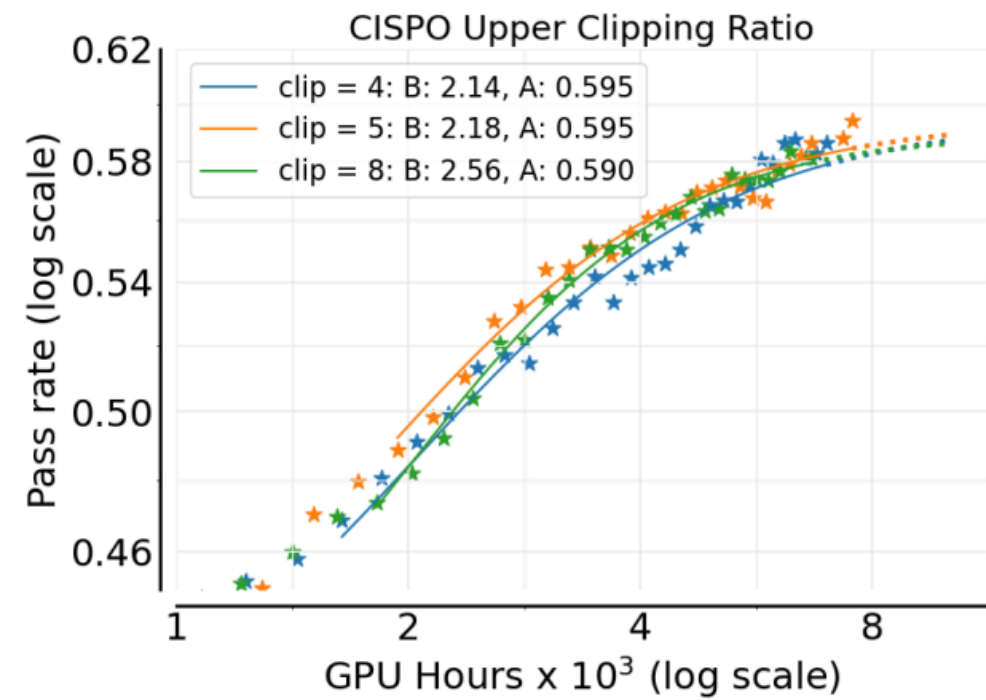
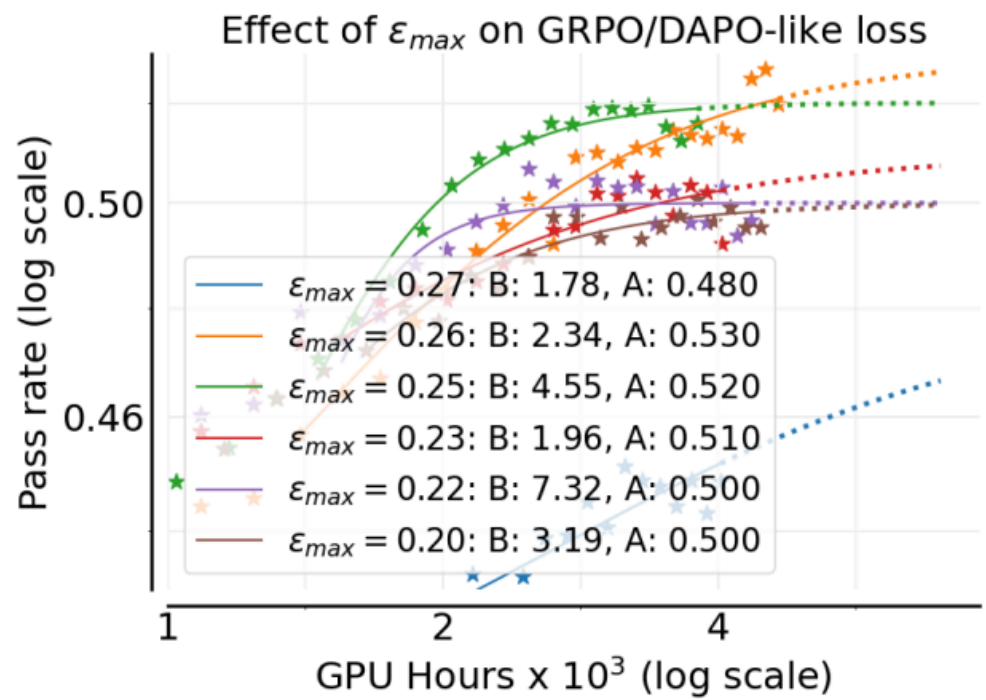
Old Policy (before any optimization steps / epochs) $\rightarrow \pi_{\theta_{\text{old}}}(a_t | s_t)$

Loss Function affects asymptotic ceiling

A



Compute scaling to determine stability and robustness!



RL for reasoning LLMs: A very short primer

Policy Gradient

Sample / Completion

Average Reward / Baseline

$$\boxed{\nabla_{\theta} J(\pi_{\theta})} = \mathbb{E}_{\underbrace{x \sim \mathcal{D}}_{\text{Prompt}}, \underbrace{y \sim \pi_{\theta}}_{\text{Sample / Completion}}} [\nabla_{\theta} \log \pi_{\theta}(y|x) (\boxed{R(x, y)} - \boxed{\bar{R}})]$$

Prompt

Sample Reward

Advantage for i-th sample

Leave one out average reward

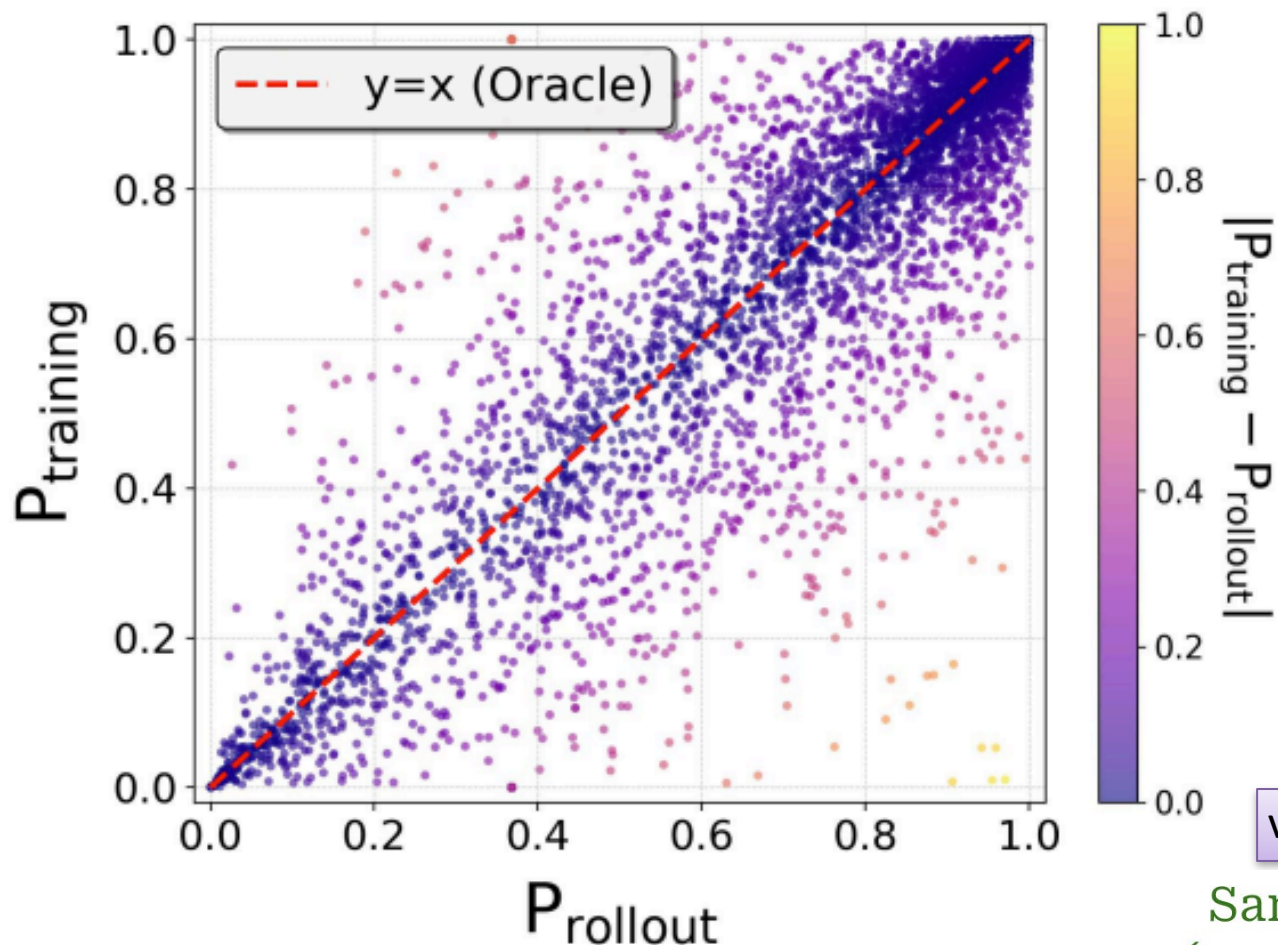
Reward for i-th sample

$$A(x, y_i) = \underbrace{R(x, y_i)}_{\text{Reward for i-th sample}} - \frac{1}{K-1} \sum_{j=1, j \neq i}^K R(x, y_j)$$

Advantage for i-th sample

$$A(x, y_i)$$

Training-Inference Discrepancy in RL



Training & inference are optimized for different purposes, resulting in mismatch due to different kernels, precision etc!

vLLM, SGLANG

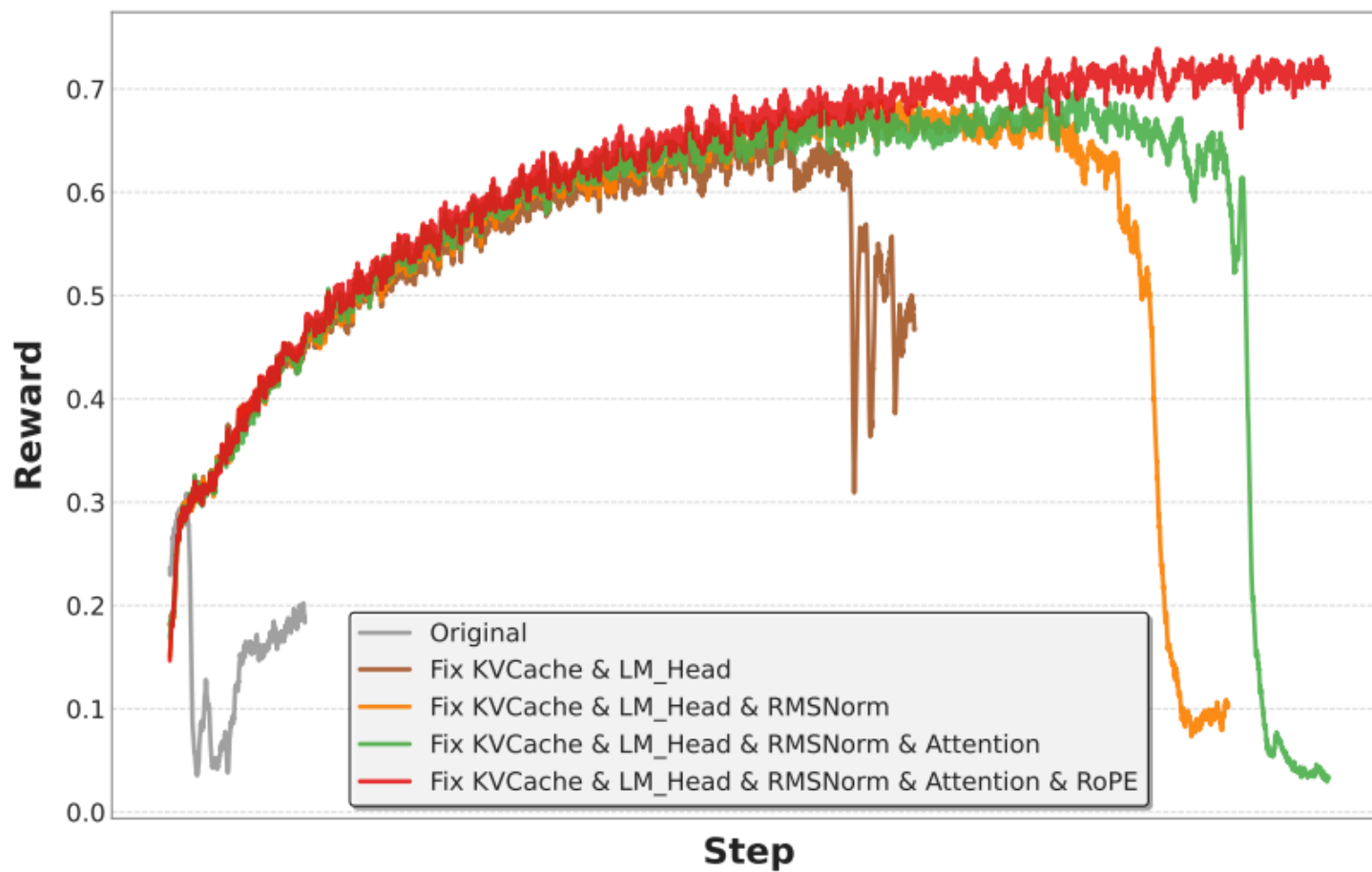
Sampler Engine
(used for rollouts)

e.g., Megatron

Learner Engine (used
for policy updates)

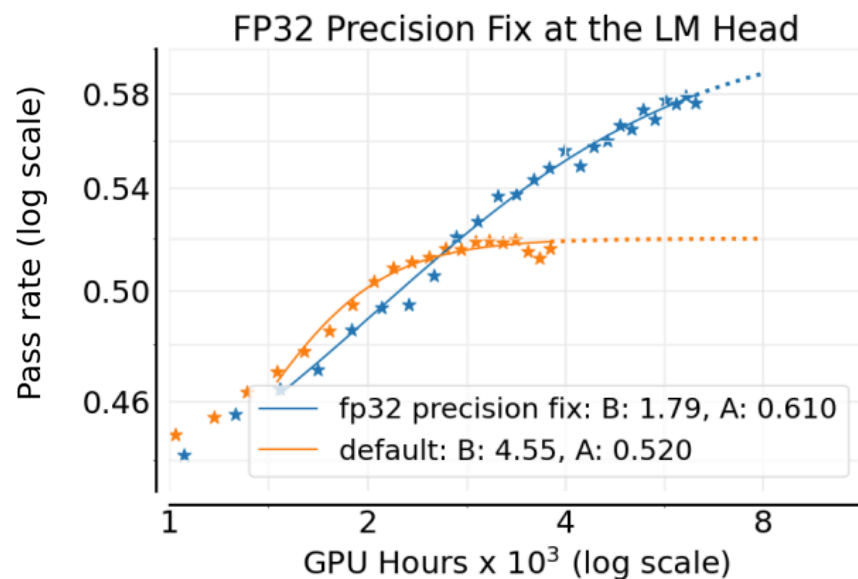
$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{a \sim \underbrace{\pi_{\theta_{\text{sampler}}}}_{\text{Sampler Engine}}, s \sim \mathcal{D}} [\nabla_{\theta} \log \underbrace{\pi_{\theta_{\text{learner}}}}_{\text{Learner Engine}}(a|s) \cdot R(a)]$$

Training-Inference Discrepancy in RL



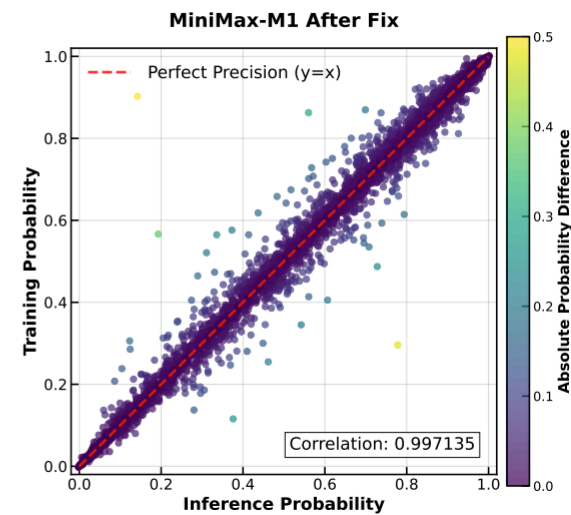
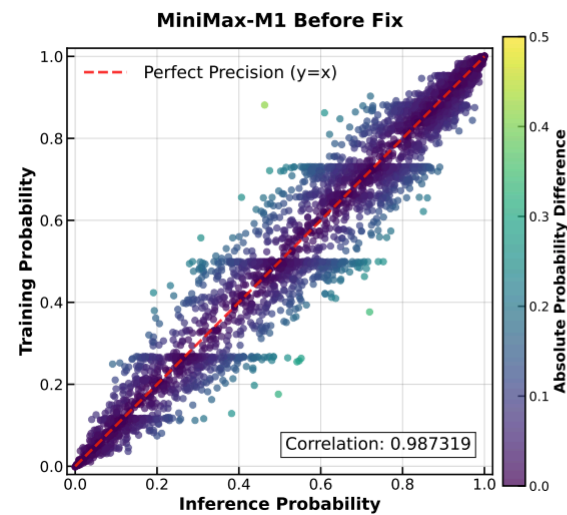
A silent killer — no error, just degraded learning dynamics.

Training-Inference Discrepancy — FP32 head affects ceiling (A)



+17% relative ceiling

**Minimizing discrepancy leads
to stable RL training!**



Specific fix: cast LM head **matmul**
to FP32.

Adaptive Sampling: Not all problems **teach** **equally**

Every GPU-hour should count

$2x + 3 = 7$	16/16 correct	→ Skip: zero variance, nothing to learn
Prove $P=NP$	0/16 correct	→ Skip: zero variance, too hard
Solve $3x + 7 = 22$	14/16 correct	→ Train! Mixed signal = learning
Factor $x^3 - 8$	5/16 correct	→ Train! High gradient signal

Adaptive Sampling: Zero-Variance Filtering

In each batch, some prompts have identical rewards across all K responses:

All correct ($r=[1,1,1,\dots,1]$):

- Advantage = 0 for every response
- Zero gradient.

All wrong ($r=[0,0,0,\dots,0]$):

- Same: zero advantage, zero gradient
- Problem is too hard right now!

Solution: Drop these prompts from the batch. Only train on informative examples.

[1, 1, 1, 1, 1, 1] → Skip! (too easy)

[0, 0, 0, 0, 0, 0] → Skip! (too hard)

[1, 0, 0, 1, 0, 1] → Train! (frontier)

Focuses compute on the learning frontier.

Adaptive Sampling: No-Positive Resampling

As training progresses, some prompts become "too easy".

These prompts still pass zero-variance filtering (not 100%), but contribute diminishing learning signal.

No-Positive Resampling:

Track historical pass rate per prompt.

If $\text{pass_rate}(x) \geq 0.9$ in any epoch, remove x from training.

Progressively focuses on harder problems as the model improves.

Training progression:

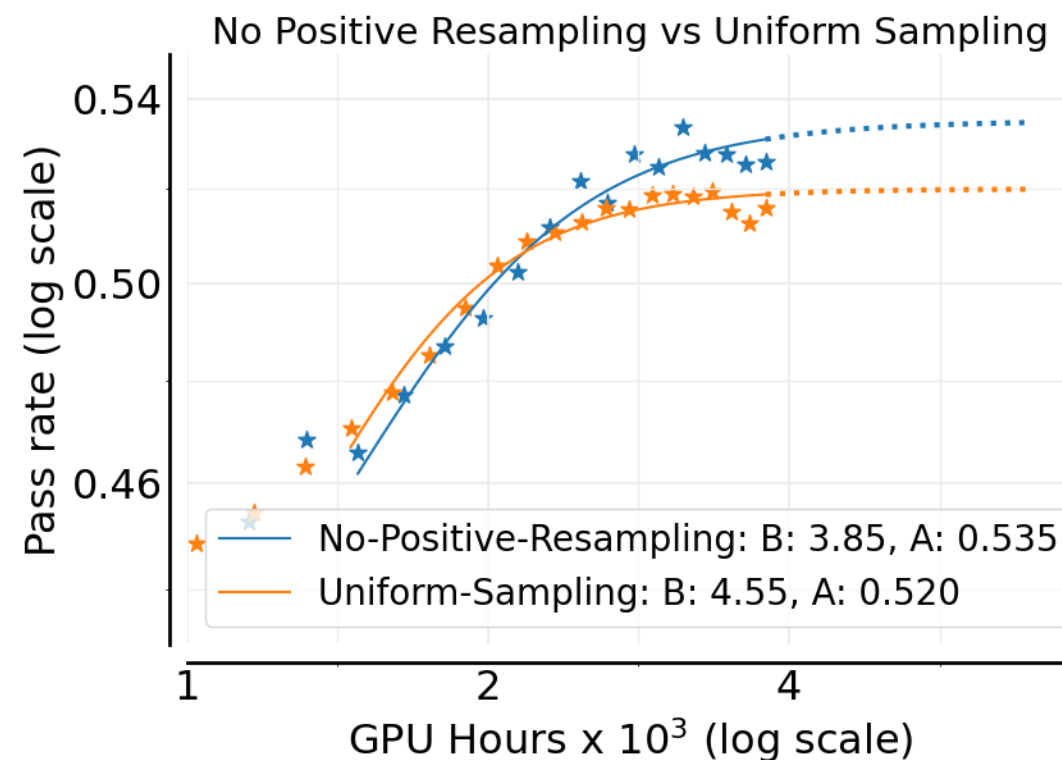
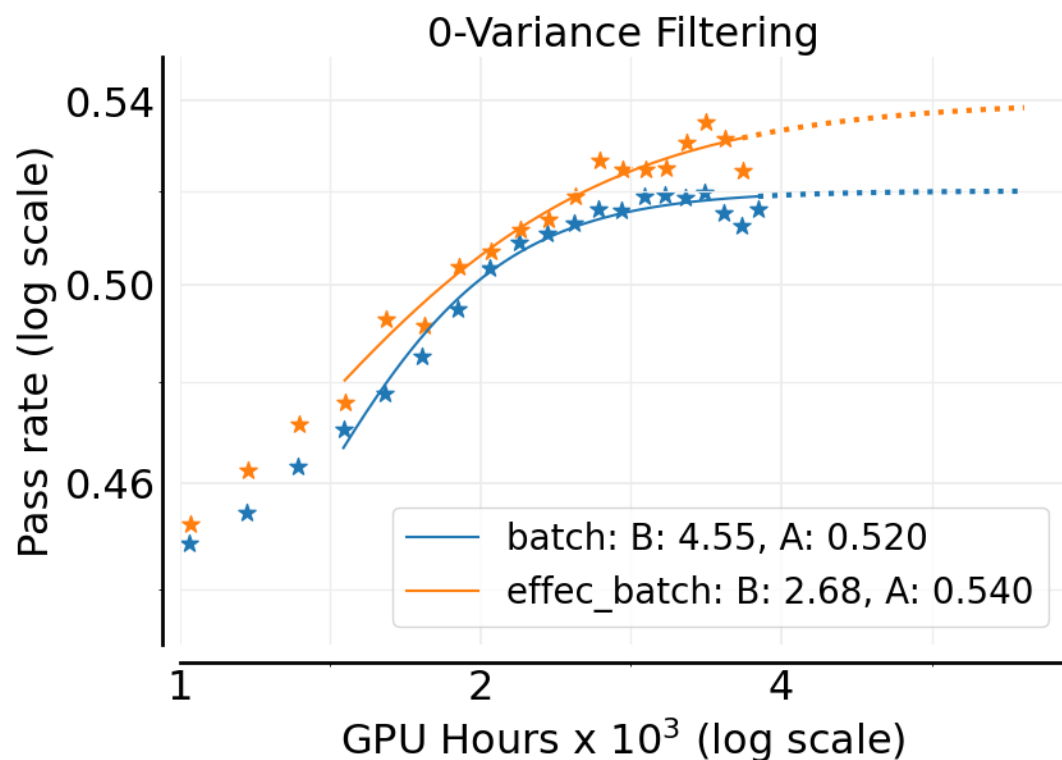
Epoch 1: 5000 problems in pool
Model solves 200 at >90% → drop them

Epoch 2: 4800 problems remain
Model solves 150 more → drop them

Epoch 3: 4650 problems remain
All are genuinely challenging!

Result: compute is never wasted on problems the model already mastered.

Adaptive Sampling improves Compute Efficiency (and possibly ceiling)!



RL Scaling: **Efficiency** vs **Asymptote**

Most design choices modulate compute efficiency (B) without substantially altering the ultimate performance ceiling (A).

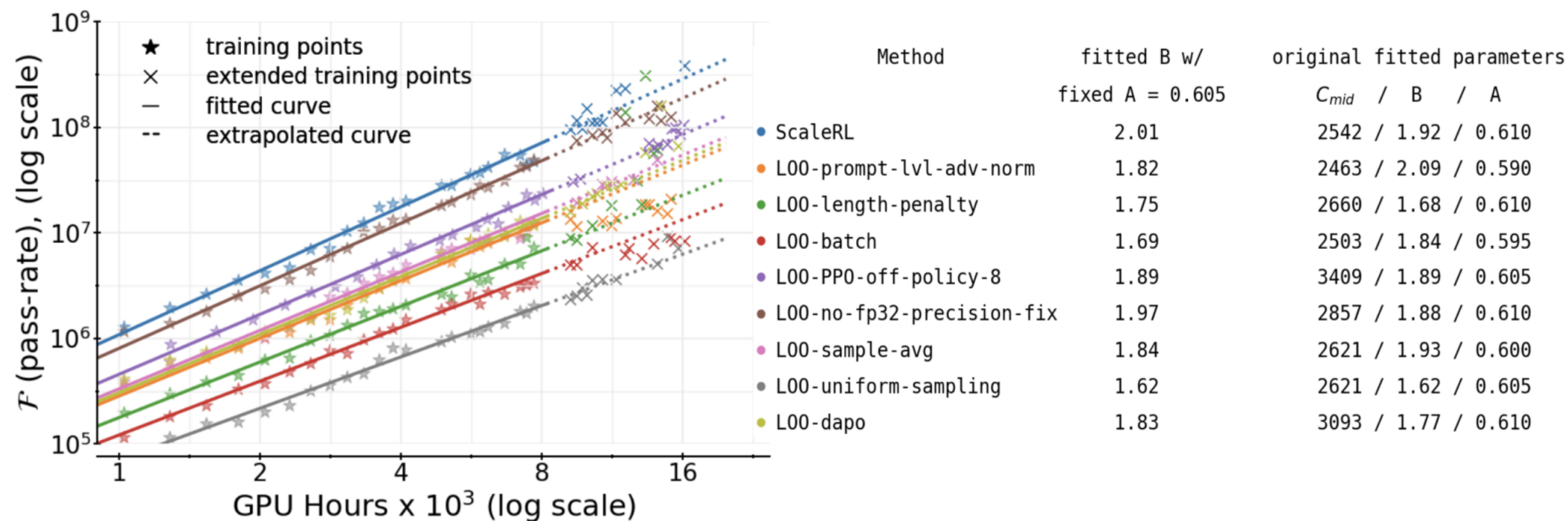
Affect Efficiency (B)

- Off-policy / Async-RL Setup
- Advantage / Length normalization
- Adaptive Sampling strategies
-

Affect Ceiling (A)

- RL algorithm (GRPO vs DAPO vs CISPO)
- Model architecture & size
- Generation length budget
- Training data quality

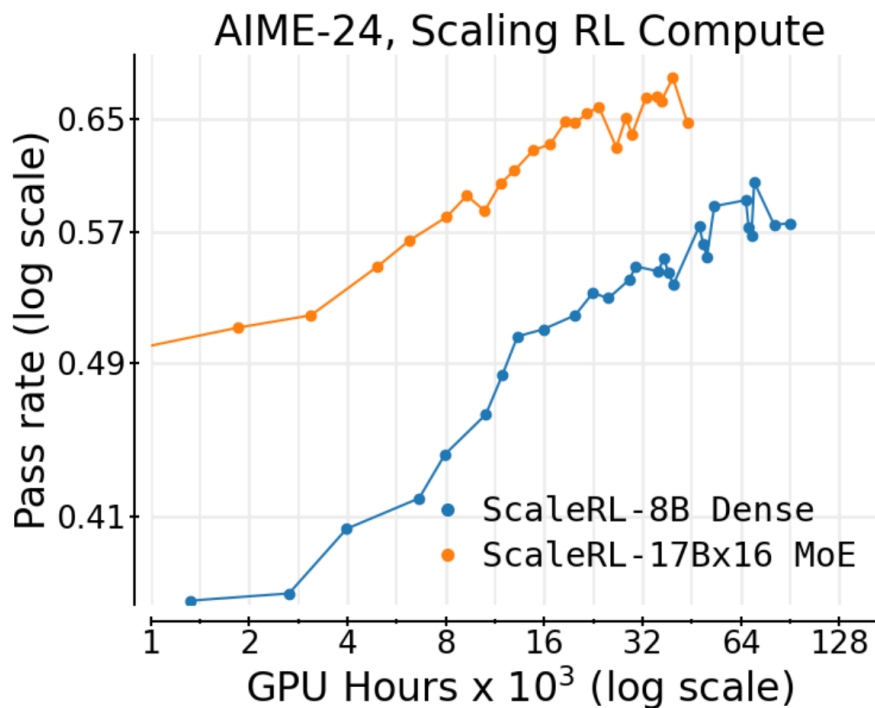
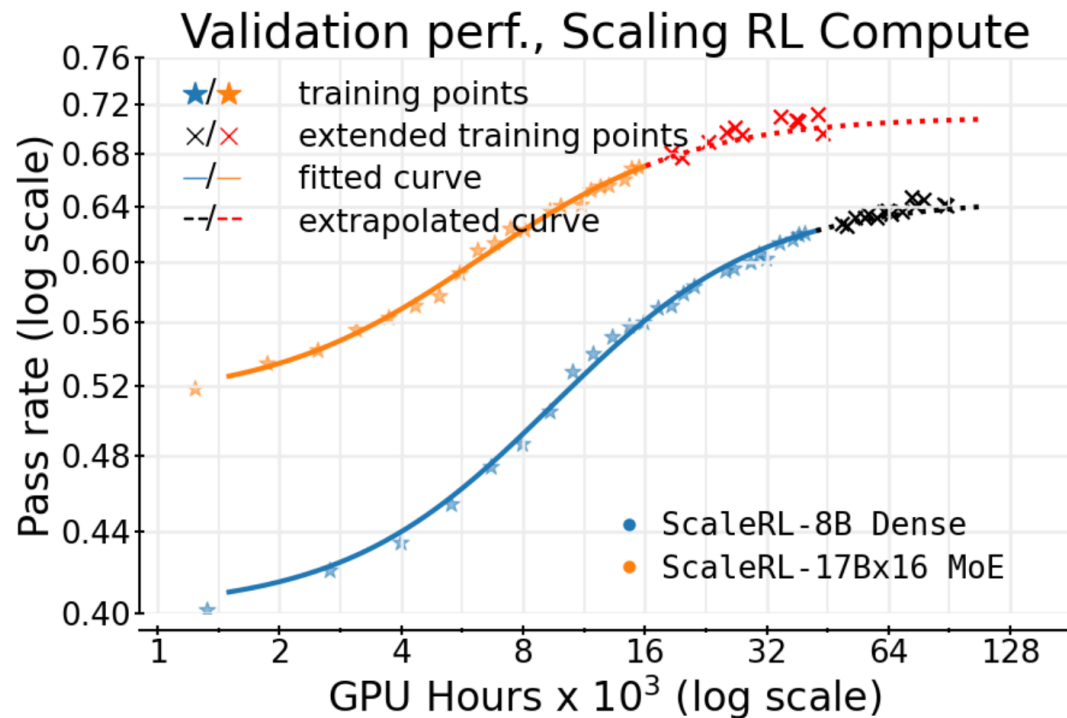
Leave-One-Out: every component in ScaleRL helps



Part 3:

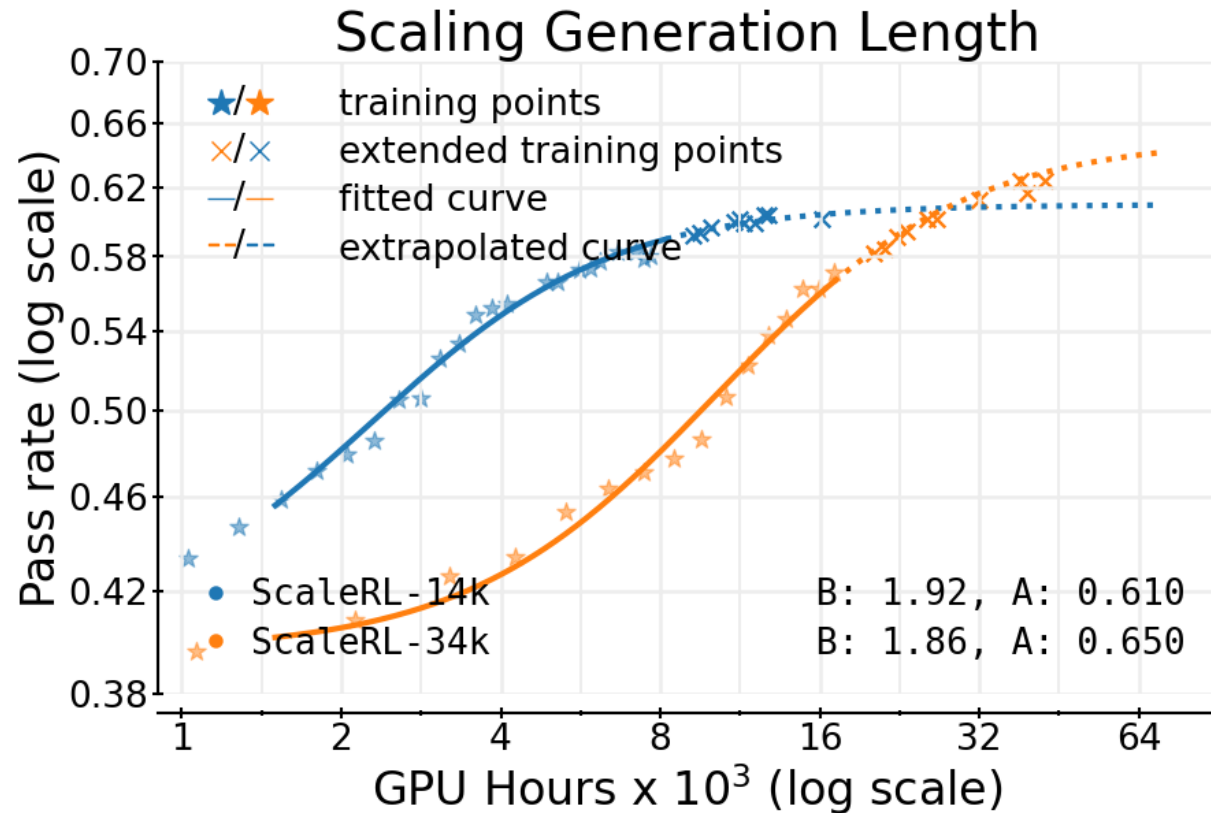
RL Knobs that affect Asymptote Ceiling

Model Size: Higher ceiling and asymptote!



Scout MoE (17Bx16) reaches 0.50 at ~7K GPU-hrs vs ~40K for 8B dense

Longer generations



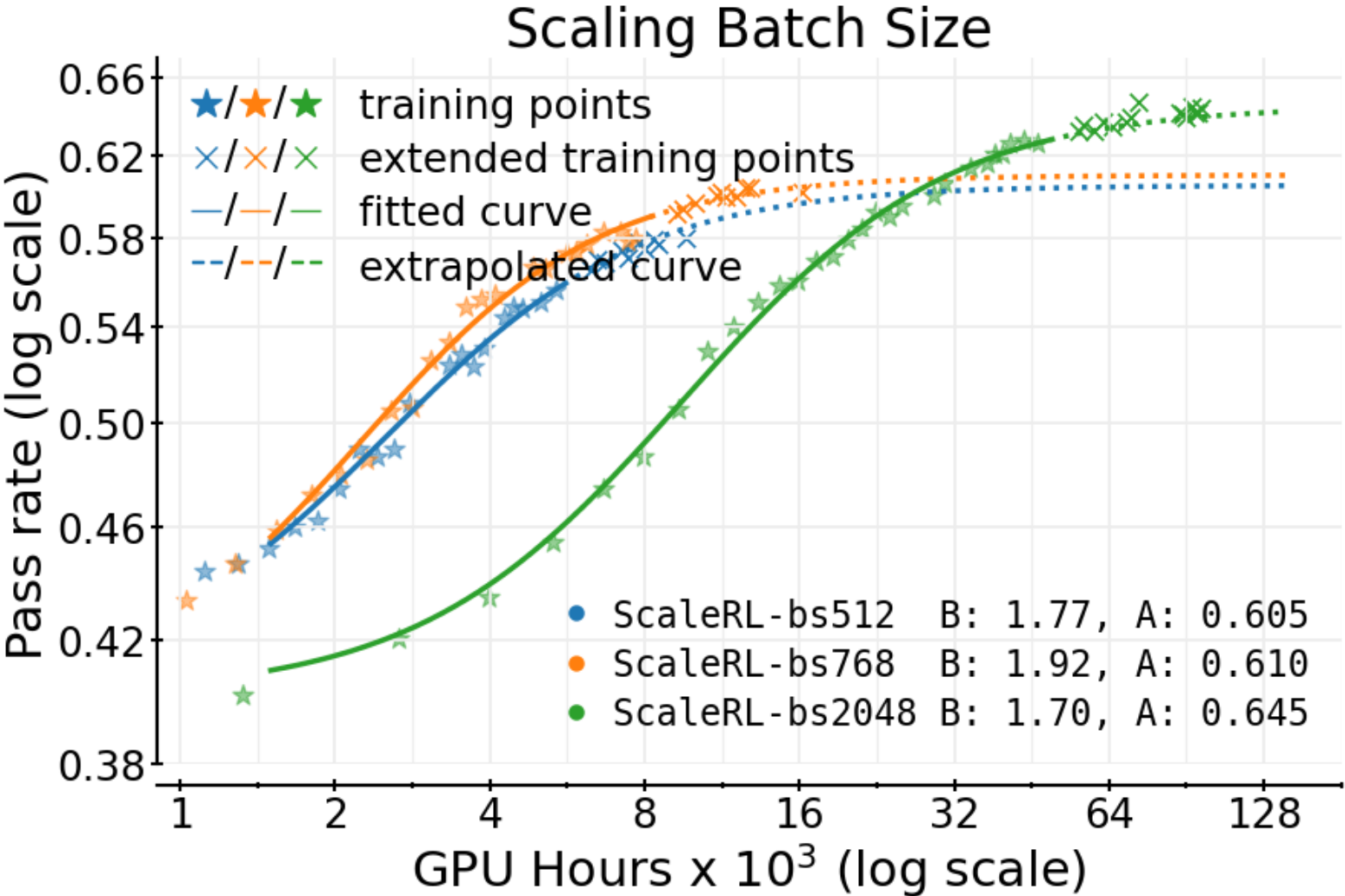
In RL training, batch size = number of prompts per update:

- Small batch (512): faster updates, noisier gradients
- Large batch (2048): slower updates, more stable gradients

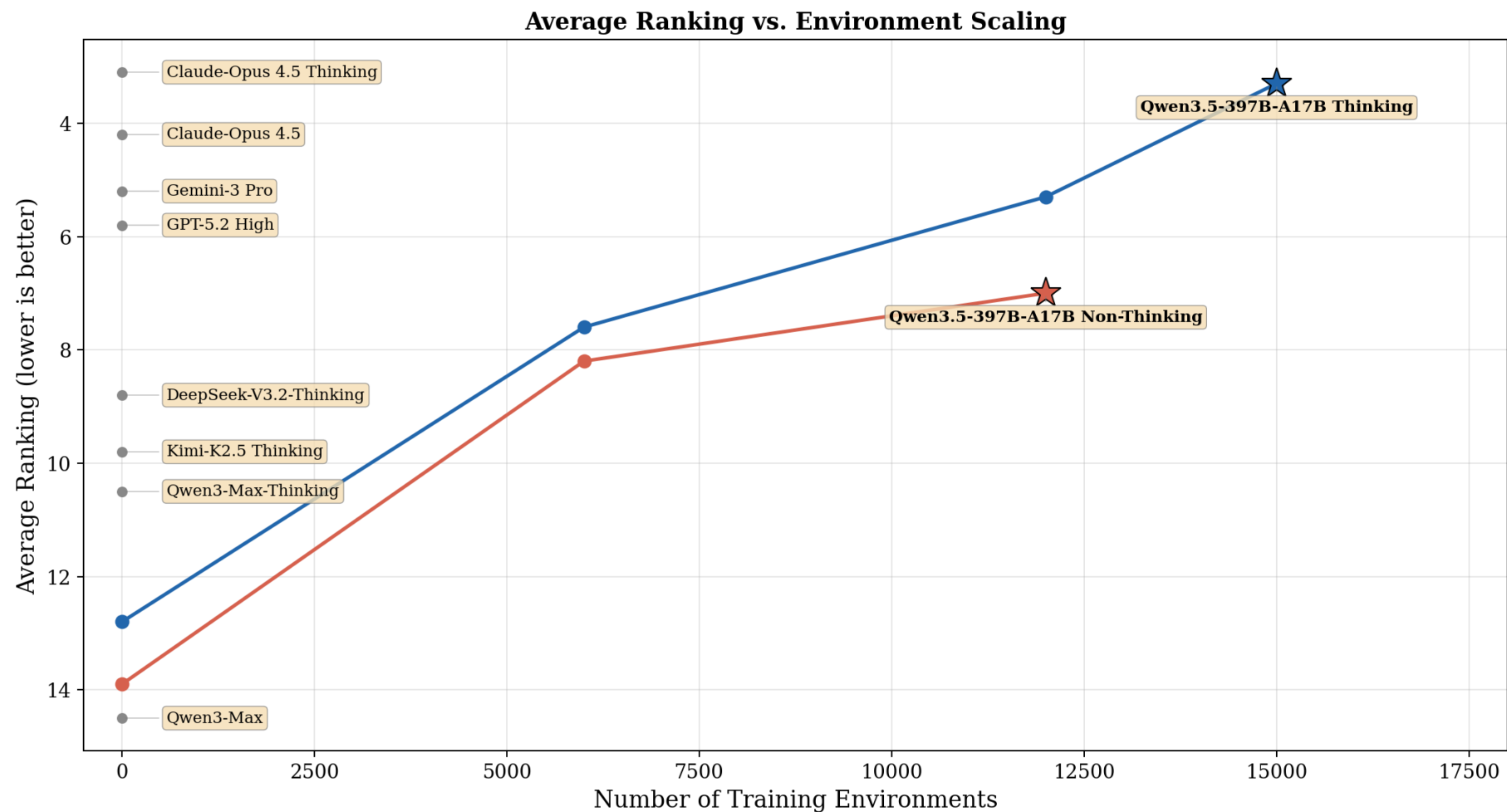
Unlike supervised learning, RL gradient quality depends on diversity of problems seen per update:

- Better coverage of the problem difficulty spectrum

Scaling Batch size



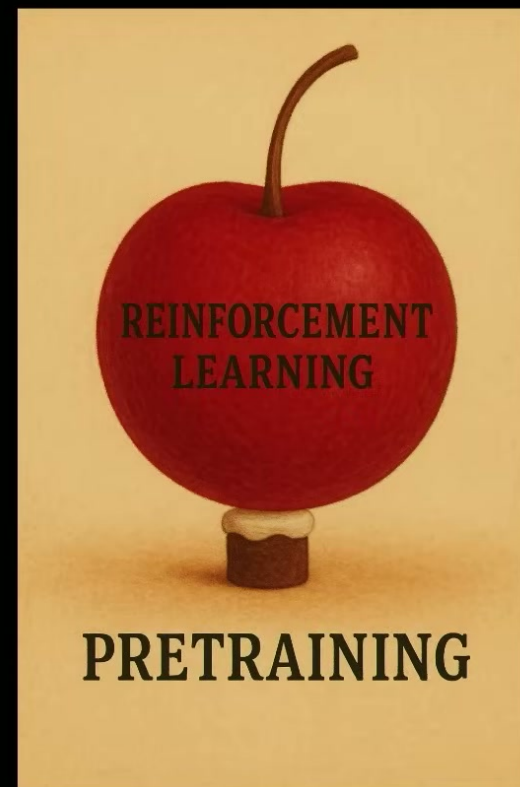
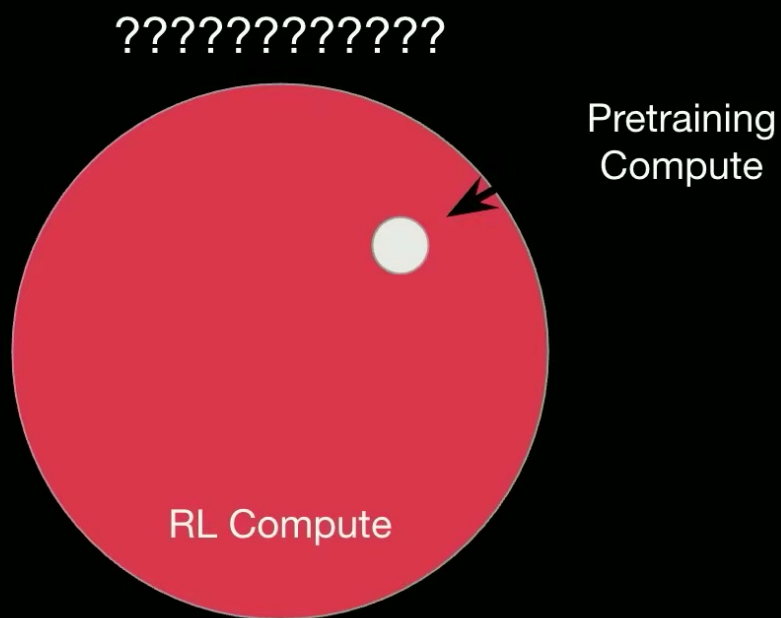
Scaling RL environments (not compute matched!)



The overall performance is calculated by averaging the ranking of each model on the following benchmarks: BFCL-V4, VITA-Bench, DeepPlanning, Tool-Decathlon, and MCP-Mark.

Now you know how to get here!

Scaling Reinforcement Learning



Thank you!

Questions?

Paper: arxiv.org/abs/2510.13786

Curve-fitting code released with the paper