The DeepSeek training pipelines

CIS 6200 October 29, 2026

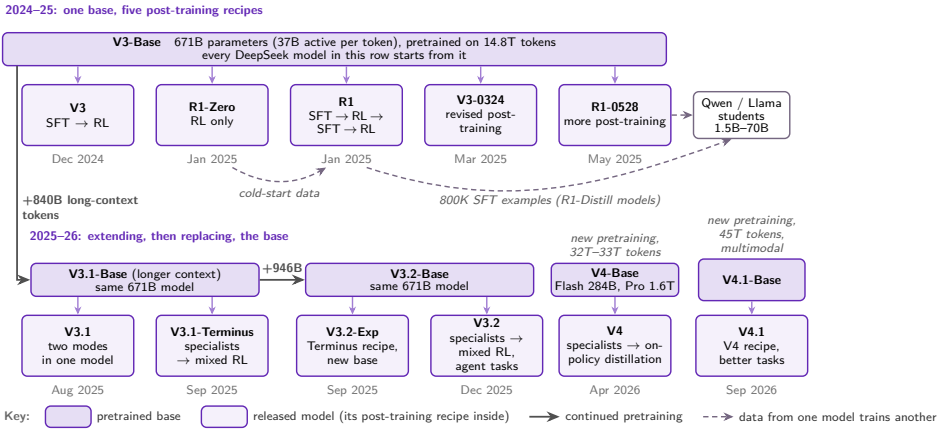
The question for today

We have learned a lot about post-training, but a frontier model combines many of these ideas.
Today: How DeepSeek chains them, release by release. For every stage:

1. **Start**: which base reference model do we start with?
2. **Data**: which prompts or responses, generated by models or humans?
3. **Feedback**: what is the reward signal or supervision?
4. **Evidence**: what comparison shows the stage mattered?

The reports answer 1–3 in detail, 4 rarely. Evidence slides carry a tag naming the kind of comparison.

The family tree



The top-row models share one base, so comparing them holds pretraining fixed. "671B, 37B active" means a mixture-of-experts model that uses 37B of its 671B parameters for each token.

The benchmarks we will track

Benchmark	What it tests	Score
AIME 2024 / 2025	competition math, integer answers	% correct
LiveCodeBench	recent contest programming	% passing tests
Aider-Polyglot	editing code to pass unit tests	% solved
IF-Eval	checkable instructions (“under 100 words”)	% followed
AlpacaEval 2.0	open-ended requests, judged by a model	% wins vs. a reference

pass@1 is the accuracy of one sampled answer, averaged over several samples. A **majority vote** takes the most common answer among 64 samples. Other benchmarks are described in the appendix.

1. The starting point: V3-Base

Fixed pretrained model. Every model today starts from it, or continues its pretraining.

What pretraining provides

- **Mixture of experts.** The model has 671B parameters but uses only 37B of them for each token.
- **Objective.** It predicts the next token over 14.8T tokens, with a small extra module that predicts one token further.
- **Context.** Sequences are 4K tokens long, later extended to 128K.
- **Cost.** Pretraining took 2.66M GPU hours; everything after took about 0.1M.
- **What it can do.** With a few worked examples, it solves 61.6% of MATH; with R1's template, 15.6% of AIME.
- **What it cannot do.** It continues text, but it does not chat or follow instructions.

QUESTION. Post-training is under 4% of the compute. Why so much effort on post-training?

ANSWER. Post-training decides what the base becomes. From the same base, DeepSeek's chat model V3 scores 39.2 on AIME 2024, while the reasoning model R1 scores 79.8.

Source: DeepSeek-V3 report §1–2, Table 3 (via the V3 README); AIME: R1 report.

2. R1-Zero: RL directly on the base

One RL stage with rule-checked rewards and no demonstrations

In the report: the claim

Post-Training: Large-Scale Reinforcement Learning on the Base Model

- We directly apply RL to the base model without relying on supervised fine-tuning (SFT) as a preliminary step. This approach allows the model to explore chain-of-thought (CoT) for solving complex problems, resulting in the development of DeepSeek-R1-Zero. DeepSeek-R1-Zero demonstrates capabilities such as self-verification, reflection, and generating long CoTs, marking a significant milestone for the research community. Notably, it is the first open research to validate that reasoning capabilities of LLMs can be incentivized purely through RL, without the need for SFT. This breakthrough paves the way for future advancements in this area.

"Zero" means zero supervised examples, not an untrained model: training starts from V3-Base.

Source: DeepSeek-R1 report (original), §1.1, p. 4.

GRPO: the RL algorithm in every DeepSeek release

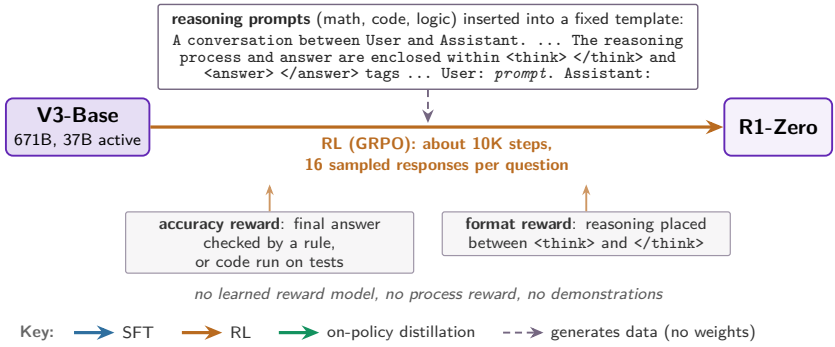
For each question q:

1. Sample a group of G responses from the current policy (R1-Zero uses $G = 16$).
2. Score each response; for R1-Zero, the reward checks the final answer and the format.
3. Compute each response's advantage $\hat{A}_i = (r_i - \text{mean})/\text{std}$ within its group.
4. Raise the probability of tokens in above-average responses, and lower it for the rest.

$$J(\theta) = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\min\Big(\rho_{i,t}\hat{A}_i,\ \text{clip}(\rho_{i,t}, 1-\varepsilon, 1+\varepsilon)\,\hat{A}_i\Big) \ - \ \beta\,\text{KL}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big)$$

- $\rho_{i,t}$ is the new-to-old probability ratio of token t; clipping limits how far one update moves it.
- The KL term keeps the policy close to a reference model ($\beta = 0.001$).
- There is no value network: the group mean serves as the baseline.

R1-Zero: one RL stage



Reading the pipeline figures. Each box is a saved model. A solid arrow is a training stage that changes the weights, and its color names the kind of feedback (**orange = RL**).

Each step uses 32 questions with 16 responses each; the run takes about 10K steps and 101K GPU hours.

Source: DeepSeek-R1 report, §2.2 (original); hyperparameters and cost from the expanded report, §2.1 and App. B.4.

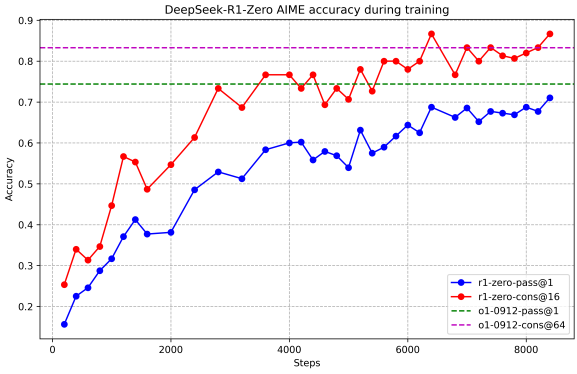
In the report: the rewards, and what is left out

- **Accuracy rewards:** The accuracy reward model evaluates whether the response is correct. For example, in the case of math problems with deterministic results, the model is required to provide the final answer in a specified format (e.g., within a box), enabling reliable rule-based verification of correctness. Similarly, for LeetCode problems, a compiler can be used to generate feedback based on predefined test cases.
- **Format rewards:** In addition to the accuracy reward model, we employ a format reward model that enforces the model to put its thinking process between '<think>' and '</think>' tags.

There is no learned reward model, because it "may suffer from reward hacking" and retraining it "complicates the whole training pipeline."

Source: DeepSeek-R1 report (original), §2.2.2.

What the run shows



Original report, Figure 2. Blue: pass@1. Red: majority vote over 16 samples.

Dashed: OpenAI o1-0912.

Source: DeepSeek-R1 report, Figure 2 and §2.2.4 (original); App. C (expanded).

- Single-sample accuracy on AIME rises from 15.6% to **71.0%**.
- A majority vote over 64 samples reaches 86.7%.
- Responses grow from hundreds to thousands of tokens.
- Words like “wait” and “verify” become 5–7$\times$ more frequent.
- The outputs are hard to read and mix languages.

In the report: R1-Zero checks its own work

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a+x}} = x$ is equal to

Response: <think>
To solve the equation $\sqrt{a - \sqrt{a+x}} = x$, let's start by squaring both $\cdots$
$\left(\sqrt{a - \sqrt{a+x}}\right)^2 = x^2 \implies a - \sqrt{a+x} = x^2$.
Rearrange to isolate the inner square root term:
$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$
$\ldots$
Wait, wait. Wait. That's an aha moment I can flag here.
Let's reevaluate this step-by-step to identify if the correct sum can be $\cdots$
We started with the equation:
$\sqrt{a - \sqrt{a+x}} = x$
First, let's square both sides:
$a - \sqrt{a+x} = x^2 \implies \sqrt{a+x} = a - x^2$
Next, I could square both sides again, treating the equation: $\cdots$
$\ldots$

Original report, Table 3: an intermediate R1-Zero checkpoint (red in the original).

Source: DeepSeek-R1 report (original), §2.2.4 and Table 3, p. 9.

- The model squares both sides, reaches a quartic, then stops ("Wait, wait. Wait.") and starts over.
- Nothing in the reward asks for this: only the final answer and the format are scored.

Open question 1: did RL create the self-checking?

EXTERNAL

EXTERNAL: a study by another group, on a question the DeepSeek reports do not address.

- Liu et al. sample V3-Base *before any RL* and find the same kind of self-checks: "DeepSeek-V3-Base already exhibit 'Aha moment'."
- So "wait" was already in the base model's distribution; it was just rare.

QUESTION. What did RL change, then? Use the outcome-reward update from the policy-gradient lecture, with y sampled from the current model: $R \nabla_\theta \log q_\theta(y \mid x) = R \sum_t \nabla_\theta \log q_\theta(y_t \mid x, y_{<t})$.

ANSWER. How *often* the model checks. Every token of a correct response is pushed up, "wait" included, so if checking leads to correct answers, it becomes more common. RL only reweights what the model already samples: with no correct sample, there is no update. DeepSeekMath saw the same pattern: majority vote rose, pass@K did not.

Source: Liu et al. (2025), §2.3; Shao et al., DeepSeekMath (2024), §5.2.2.

Open question 2: does RL from a base work at 32B?

CONTROLLED

CONTROLLED: one ingredient is changed while everything else is held fixed. This is the strongest kind of evidence, and it is rare.

PREDICT. Start from the Qwen2.5-32B base. Model A is trained with R1-Zero's RL recipe; model B is fine-tuned on R1's 800K examples. Which scores higher on AIME, and by how much?

Qwen2.5-32B, trained by	AIME 2024	MATH-500	LiveCodeBench
A: RL on the base	47.0	91.6	40.2
B: SFT on the 800K examples	**72.6**	**94.3**	**57.2**

ANSWER. B, by 25.6 points. At 32B, learning from a stronger model's solutions beats discovering them through RL.

Source: DeepSeek-R1 report (original), §4.1, Table 6.

3. R1: supervised data around RL

Four stages, with two restarts from V3-Base

In the report: why add supervision?

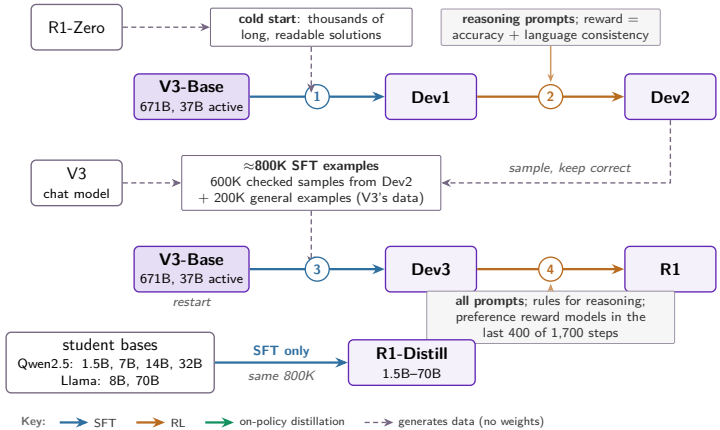
However, DeepSeek-R1-Zero encounters challenges such as poor readability, and language mixing. To address these issues and further enhance reasoning performance, we introduce DeepSeek-R1, which incorporates a small amount of cold-start data and a multi-stage training pipeline.

2.3.1. Cold Start

Unlike DeepSeek-R1-Zero, to prevent the early unstable cold start phase of RL training from the base model, for DeepSeek-R1 we construct and collect a small amount of long CoT data to fine-tune the model as the initial RL actor. To collect such data, we have explored several approaches: using few-shot prompting with a long CoT as an example, directly prompting models to generate detailed answers with reflection and verification, gathering DeepSeek-R1-Zero outputs in a readable format, and refining the results through post-processing by human annotators.

Source: DeepSeek-R1 report (original), §1 and §2.3.1.

R1, one stage at a time



Blue = SFT, orange = RL, dashed = data. Stage 3 restarts from V3-Base, using 600K checked solutions from Dev2 and 200K examples from V3.

In the report: building the 800K dataset

Additionally, because the model output is sometimes chaotic and difficult to read, we have filtered out chain-of-thought with mixed languages, long parapraphs, and code blocks. For each prompt, we sample multiple responses and retain only the correct ones. In total, we collect about 600k reasoning related training samples.

Non-Reasoning data For non-reasoning data, such as writing, factual QA, self-cognition, and translation, we adopt the DeepSeek-V3 pipeline and reuse portions of the SFT dataset of DeepSeek-V3. For certain non-reasoning tasks, we call DeepSeek-V3 to generate a potential chain-of-thought before answering the question by prompting. However, for simpler queries, such as "hello" we do not provide a CoT in response. In the end, we collected a total of approximately 200k training samples that are unrelated to reasoning.

We fine-tune DeepSeek-V3-Base for two epochs using the above curated dataset of about 800k samples.

Source: DeepSeek-R1 report (original), §2.3.3, pp. 10–11.

Predict before we look

PREDICT. Stage 1 fine-tunes V3-Base on a few thousand readable solutions, giving Dev1. Compared with R1-Zero, will Dev1 do better or worse on **AIME**, **IF-Eval**, and **AlpacaEval**?

PREDICT. Which stage will raise AlpacaEval the most?

The answer is a STAGE TABLE: the same benchmarks measured after each stage. Since each stage changes the data, the feedback, and the starting weights at once, the table shows *where* scores moved, but not exactly *why*.

What each stage changed: four benchmarks

STAGE TABLE

	V3 chat model	R1-Zero RL only	Dev1 1: SFT, thousands of long solutions	Dev2 2: RL, reasoning prompts only	Dev3 3: SFT on 800K, from V3-Base	R1 4: RL, all prompts; pref. reward last
AIME 2024 competition math	39.2	77.9	59.0 −18.9	74.0 +15.0	78.1 +4.1	79.8 +1.7
Aider-Polyglot code editing	49.6	12.2	6.7 −5.5	25.6 +18.9	44.8 +19.2	53.3 +8.5
IF-Eval format instructions	86.1	46.6	71.7 +25.1	72.0 +0.3	78.1 +6.1	83.3 +5.2
AlpacaEval 2.0 judged helpfulness	70.0	24.7	50.1 +25.4	55.8 +5.7	62.1 +6.3	87.6 +25.5

Small numbers show the change from the column to the left. Green is a gain and red a loss; darker means 10 points or more. Gray V3 is DeepSeek's chat model from the same base. Sources: expanded R1 report; V3 from the original report, Table 4.

Stage 1 trades reasoning for format (AIME −19, IF-Eval and AlpacaEval +25). **Stage 2** brings reasoning back (AIME +15, Aider +19). **Stage 3** broadens the model (Aider +19). **Stage 4** targets preferences (AlpacaEval +25.5). R1 still trails R1-Zero on GPQA (71.5 vs. 75.8).

Two rewards and their costs

Language consistency

CONTROLLED

- The reward is the share of the reasoning written in the user's language.
- In an ablation, it makes reasoning readable at the cost of a "slight degradation" in accuracy.

Preference reward

REPORTED FAILURE

- **Reward hacking**: the helpfulness reward keeps rising while the Codeforces rating falls.
- The fix is to use preference rewards only in the last 400 of 1,700 steps.

REPORTED FAILURE: something the authors tried that went wrong.

QUESTION. Why not simply raise the KL penalty instead?

ANSWER. A larger KL penalty slows *every* update, including the gains from the rule-based rewards. Limiting the preference reward to 400 steps only caps exposure to the reward that can be hacked.

Source: DeepSeek-R1, original §2.3.2; expanded §3.2.2, App. B.5–B.6.

What did not work

REPORTED FAILURE

Process reward model (PRM)

A learned model that scores each intermediate step.

- A "step" is hard to define in general reasoning.
- Labeling steps does not scale.
- A learned process reward "inevitably leads to reward hacking."

It is still useful for reranking and guided search.

Monte Carlo tree search (MCTS)

A value model chooses which partial solutions to expand.

- The space of token sequences is enormous; limiting it leads to local optima.
- The value model is hard to train.

It can still help at inference, with a pretrained value model.

Source: DeepSeek-R1 report (original), §4.2 "Unsuccessful Attempts".

Students, and the cost of the pipeline

- **Distillation.** Six open models (1.5B to 70B) are fine-tuned on the 800K examples, with SFT only.
- Each student learns at the teacher's prefixes and never sees what to do after its *own* mistakes. RL trains on its own attempts, but at 32B it lost to distillation. V3.2 and V4 combine a strong teacher with the student's own attempts.
- **Cost.** R1-Zero and R1 together took 147K GPU hours (about $294K), compared with 2.66M to pretrain the base.

QUESTION. At 32B, RL from the base reached 47.0 on AIME, while SFT on the 800K examples reached 72.6. What do the teacher's solutions provide?

ANSWER. Correct solutions to hard problems. RL can only reinforce what the model samples, and a 32B base rarely samples a correct solution, so its reward is almost always zero.

Source: DeepSeek-R1, original §2.4; expanded App. B.4.4, Table 7; V3 report Table 1.

4. From R1 to V3.2

Releases that change one or two things at a time.

Most evidence here is **RELEASE VS. RELEASE**: two released models, whose base, data, compute, and evaluation may all differ.

R1-0528: more post-training

	R1	R1-0528
AIME 2025	70.0	87.5
tokens per question	12K	23K

Both start from the same base. R1-0528 had more post-training; other changes are not stated.

QUESTION. R1-0528 scores 17.5 points higher, but it also thinks twice as long. Did training improve, or does it only think more?

ANSWER. Compare both at the same token budget. If R1-0528 still wins at 12K tokens, training improved. The model card reports no such comparison; it attributes the gain to "enhanced thinking depth."

Source: DeepSeek-R1-0528 model card.

V3.1: extending the context to 128K tokens

V3-Base was pretrained on 4K-token sequences, so it never saw a position beyond 4K. Long reasoning and agent transcripts need much more.

Context extension happens after pretraining. The position encoding is rescaled so that long positions resemble familiar ones (V3 uses a method called YaRN), and training continues on long documents at 32K tokens, then at 128K.

tokens in the	V3-Base	V3.1-Base
32K phase	≈63B	630B (10×)
128K phase	≈63B	209B (3.3×)

V3.1 collected more long documents and trained much longer in both phases: 840B extra tokens.

QUESTION. Why not pretrain on 128K-token sequences from the start?

ANSWER. Attention's cost grows with sequence length, and few documents are that long. A short extension phase is much cheaper.

Source: DeepSeek-V3 report, §4.3; DeepSeek-V3.1-Base model card.

V3.1: thinking and non-thinking modes

The prompt sets the mode. V3.1's chat template ends the prompt in one of two ways:

```
thinking:      <|begin_of_sentence|>{system}<|User|>{query}<|Assistant|><think>
non-thinking:  <|begin_of_sentence|>{system}<|User|>{query}<|Assistant|></think>
```

- After <think>, the model writes its reasoning, </think>, and the answer. After </think>, it answers directly.
- Earlier turns keep only their answers: <|Assistant|></think>{answer}<|end_of_sentence|>.
- **Training:** the SFT data has both kinds of responses, from separate generator models. **Inference:** the application picks the mode.

Source: DeepSeek-V3.1 model card (chat template); V3.2-Exp report §2.2; V3.2 report §3; V4 report §5.1.1.

5. V3.2: specialists, then one model

Train a specialist for each domain, then combine them into one model

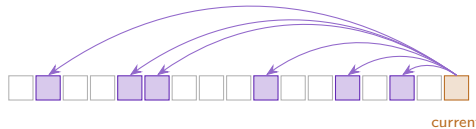
V3.2-Base: the same model with cheaper attention

Start: V3.1-Base, already extended to 128K tokens.

Change: *sparse attention*. A small, cheap *indexer* scores all earlier tokens, and each token attends only to the 2,048 highest-scoring ones instead of to all of them.

Training: continued pretraining on the same 128K-token data:

1. 2.1B tokens that train only the indexer, to match where full attention looks;
2. 943.7B tokens that train the whole model with sparse attention.



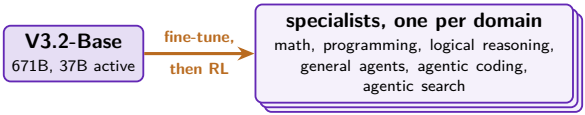
earlier tokens; the indexer picks a few (purple) for attention

Attention's cost per token no longer grows with the full context, which matters for long agent transcripts.

V3.2-Exp, a test release, applied V3.1-Terminus's post-training to this new base and scored about the same as V3.1-Terminus on most benchmarks.

Source: DeepSeek-V3.2-Exp report, §2.1 and Table 1.

The V3.2 pipeline



1. **Specialists.** From one base, train a separate model for each domain: math, programming, logical reasoning, general agent tasks, agentic coding, and agentic search. Each is fine-tuned, then trained with RL.
2. **Distillation.** Fine-tune one model on all the specialists' responses.
3. **Mixed RL.** Train that one model with RL on all domains at once.

We start with step 1, the tasks each specialist is trained on.

Source: DeepSeek-V3.2 report, §3.

Training tasks for each specialist

Domain	One task	Reward
math, logical reasoning	a problem with a known answer	1 if the final answer is right, as in R1
programming	a problem with test cases	1 if the program passes the tests
agents: search, coding, general	**a goal that takes many steps with tools**	**1 if a check of the result passes**

V3.2 adds a length penalty and a language-consistency reward to each of these.

Agent tasks are new. In R1 the model only wrote an answer; an agent must also act, for example by looking things up, running code, or changing files. The next slide shows how a model that outputs only text can do this.

Source: DeepSeek-V3.2 report, §3 (rule-based outcome reward for reasoning and agent tasks).

Tool use

```
[system] You are a coding agent. Tools:
         run(cmd): run a shell command
         edit(path, old, new): replace text in a file
[user]   Fix the failing test for dates.py.
[model]  <think>Find the failing test first.</think>
         run("pytest tests/test_dates.py -x")
[tool]   FAILED test_parse: expected 2026-11-02,
         got 2026-02-11
[model]  <think>Day and month are swapped.</think>
         edit("dates.py", "%Y-%d-%m", "%Y-%m-%d")
[tool]   ok
[model]  run("pytest tests/test_dates.py")
[tool]   1 passed
[model]  Fixed: the parser swapped day and month.
```

A **tool** is a function the model may call: run a command, search the web, read a database.

- The model writes the call as ordinary text.
- A program around the model, the **harness**, runs the call and pastes the result into the context.
- The model reads the result and continues.

The weights do not change during an attempt; only the context grows.

What a tool looks like

The model sees a list of tools in its system prompt:

```
search_books(author: str)
  "Return the books in the catalog by this author."
submit_answer(text: str)
  "Submit your final answer."
```

One call and its result:

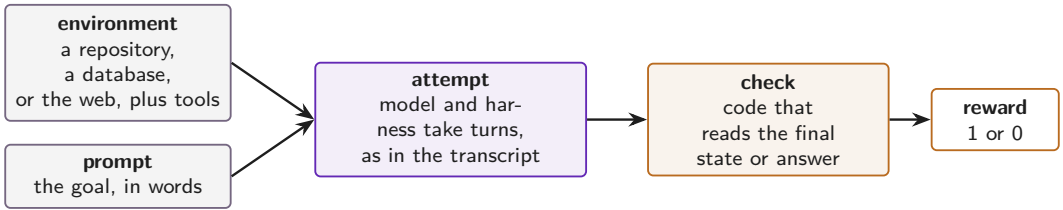
```
[model] search_books("Toni Morrison")
[tool]  [{"id": 17, "title": "Beloved", "price": 18},
         {"id": 42, "title": "Sula", "price": 15}]
```

The environment's designer writes each tool as a function over the environment's data:

```
def search_books(author):
    return [b for b in db.books
            if b.author == author]
```

The harness runs the function and pastes its result into the context. The model ends by calling `submit_answer`, and the check reads what it submitted.

RL tasks with tools



RL is the same as in R1: run a group of attempts per task, score each one, and reinforce the attempts that score above the group's average. An attempt is now a whole transcript, and only the model's own tokens are its actions.

V3.2's agent tasks	count	environment	prompt
code agent	24,667	real	extracted
search agent	50,275	real	synthesized
general agent	4,417	synthesized	synthesized
code interpreter	5,908	real	extracted

Source: DeepSeek-V3.2 report, §3.2.3 and Table 1.

"Real" means real tools: a web-search API, coding tools, or a Jupyter notebook. Next: a real environment (code). After the breakout: a synthesized one.

Code tasks from GitHub (SWE-bench)

Building one task

1. Find a GitHub issue (a bug report) and the pull request that fixed it, with its tests.
2. Set up the repository as it was before the fix, and run the tests.
3. Keep the task only if the known fix makes a failing test pass and breaks no test that passed before.
4. During RL, the model gets only the issue text and the repository, and writes its own fix.

Environment: the repository and a shell.
Prompt: the issue text.
Check: run the tests. A **fail-to-pass** test failed before the known fix and passes after it.

SWE-bench (Jimenez et al., ICLR 2024) introduced this recipe as a benchmark: 2,294 issues from 12 Python repositories. It is the standard test of coding agents.

Source: DeepSeek-V3.2 report, §3.2.3; Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" (ICLR 2024).

A check that rewards a wrong program

QUESTION. A task asks for a function that sorts a list. Its tests only check that the output is in increasing order. Write a wrong program that passes.

```
def sort(xs):
    return []
```

ANSWER. It passes. Add a test that the output has the same elements as the input.

RL would reinforce `return []`, because the check gives it full reward. The check must test what the prompt asks for.

Breakout: design an environment for trip planning

Skill: plan multi-day trips that meet a traveler's constraints, using tools. No real environment exists, so you will design one.

Example request. Three days from Boston, a different city each night, under $900 in total, at least one museum, and only restaurants rated 4 or higher.

Instructions

1. Join your breakout room (3–4 people).
2. Write in your room's table in the shared doc.
3. Spend 6 minutes on the four questions.
4. Then we compare designs as a class.

Environment	What data exists, and which tools can the model call? Write two tool signatures.
Reward	Many plans are valid. What does your code check to give 1 or 0?
Hacks	How could a model earn reward without a valid plan? How would you patch it?
Many tasks	How do you get thousands of such tasks, from easy to hard, without people writing each one?

V3.2's trip-planning environment

An Example of Synthesized Task: Trip Planning

I'm planning a three-day trip starting from Hangzhou, and I need help creating an itinerary from October 1st to October 3rd, 2025. A few important requirements: I don't want to repeat any cities, hotels, attractions, or restaurants during the entire trip. Also, please make sure that every hotel, restaurant, and attraction you recommend is actually located in the city where I'll be staying that day. One more thing about the second day - I'm trying to be smart about my budget. If I end up booking a luxury hotel that costs 800 CNY or more per night, then I need to be more careful with other expenses: my total spending on both restaurants (lunch and dinner) should stay under 350 CNY, both restaurants should be rated at least 4.0 stars, and the afternoon attraction ticket needs to be less than 120 CNY. If the hotel on day 2 is in the mid-to-high range (500-800 CNY), then I have a bit more flexibility - I just need to make sure at least one of my restaurant choices is rated 4.0 or higher, and the attraction ticket should be below 180 CNY. For more affordable hotels (200-500 CNY range), I only need to ensure that at least one restaurant has a rating of 3.2 or above. Can you help me put together this itinerary?

Source: DeepSeek-V3.2 report, p. 12.

V3.2 faced the same problem for its *general-agent* tasks, and synthesized the environment as well as the task. This trip-planning task is from the report.

The tools read a generated database of cities, hotels, restaurants, and attractions:

```
get_all_cities()
get_all_hotels_by_city(city)
get_all_restaurants_by_city(city)
get_infos_by_hotel(keys, hotel)
...
submit_result(answer_text)
```

A verifier for the trip-planning task

```
def verify(plan):              # True gives reward 1
  days = plan.days  # each: city, hotel, lunch,
                    #       attraction, dinner
  # no repeated cities, hotels, attractions, restaurants
  ok = distinct([d.city for d in days])
  ok &= distinct([p for d in days for p in d.places])
  # every place is in that day's city
  ok &= all(city(p) == d.city
            for d in days for p in d.places)
  # day 2, with a hotel of 800 CNY or more
  d = days[1]
  if price(d.hotel) >= 800:
    ok &= price(d.lunch) + price(d.dinner) < 350
    ok &= min(rating(d.lunch), rating(d.dinner)) >= 4.0
    ok &= price(d.attraction) < 120
  # ... the other two hotel tiers are similar
  return ok
```

`city`, `price`, and `rating` read the database.

Each sentence of the request becomes a few lines. `city(p)` also rejects places the model made up, since they are not in the database.

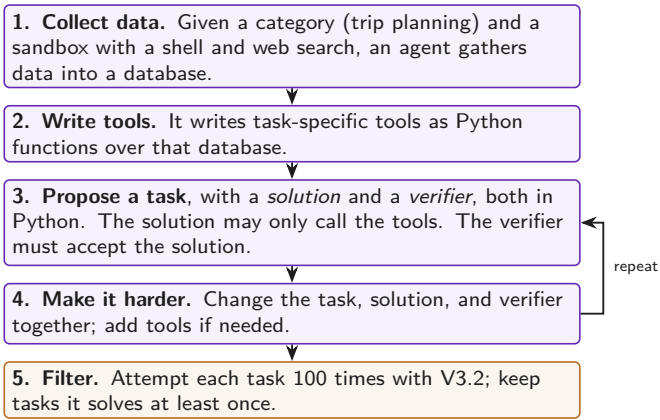
QUESTION. Why not compare the model's plan with one correct plan?

ANSWER. Many plans satisfy the request. Matching one plan would reject the others.

Finding a plan takes many tool calls. Checking one takes a few lookups.

Source: Our sketch of a verifier for the task on the previous slide; the report does not show its verifiers' code.

How V3.2 builds thousands of tasks



QUESTION. Why write a solution at all, if the model will solve the task itself?

ANSWER. It proves the task can be solved with the tools, and that the verifier accepts a correct answer.

Step 5 keeps only tasks with some successes, so GRPO has signal. This produced 4,417 tasks in 1,827 environments, described as "hard to solve but easy to verify."

Source: DeepSeek-V3.2 report, §3.2.3.

From synthesized tasks to real tools

CONTROLLED

CONTROLLED: one ingredient is changed while everything else is held fixed.

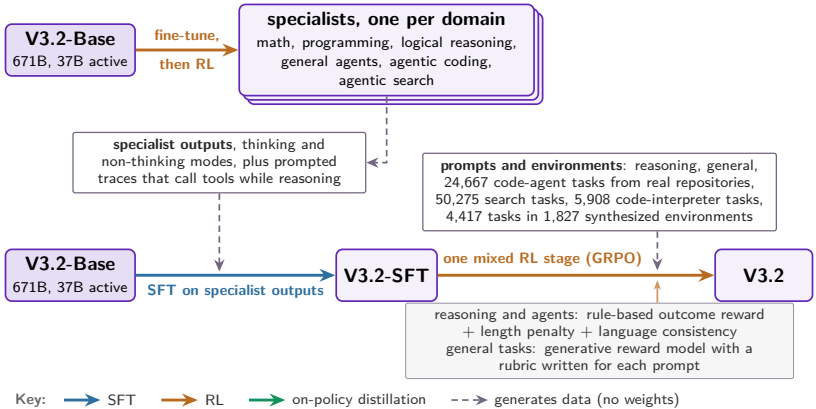
PREDICT. Start from V3.2's SFT checkpoint. Run RL on *only* the synthesized general-agent tasks. Then test on three tool-use benchmarks it never trained on, such as τ^2-Bench: customer-service conversations in which the model uses an airline's or a store's tools. Do the scores go up?

ANSWER. Yes, on all three. Training on synthesized environments improved scores with real tools.

Not shown: which properties of the tasks matter. The report does not vary the number or kind of environments.

Source: DeepSeek-V3.2 report, §3.2.3 and Figure 5. The other two benchmarks are MCP-Mark and MCP-Universe.

Combining the specialists



Step 2, distillation. One model is fine-tuned on all the specialists' responses, as in R1's distillation but with six teachers. **Step 3, mixed RL.** One GRPO stage over all domains, with the same checks; tasks without a check are scored by a judge model (next slide).

Generative reward models

In the RLHF lecture, a **reward model** $r_\phi(x, y)$ was a network that outputs one number, fit to human comparisons.

A **generative reward model** is a language model used as a judge. It reads the prompt, a rubric, and a response, writes an evaluation, and ends with a score that a program reads.

A **rubric** is a short checklist written for one prompt: what a good response must do.

V3.2's mixed RL uses one for tasks that no rule can check, such as writing.

Prompt. Email my landlord that the heating has been broken since Monday.

Rubric. (1) Says what is broken and since when. (2) Asks for a repair date. (3) Polite, under 120 words.

Response. *(the policy's email)*

The judge writes. (1) Yes: the heating, since Monday. (2) No: it never asks when the repair will happen. (3) Yes: polite, 64 words.
Score: 2/3.

Source: DeepSeek-V3.2 report, §3; DeepSeek-V4 report, §5.1.1.

RL after distillation

> Once the specialist models are prepared, they are used to produce the domain-specific data for the final checkpoint. Experimental results demonstrate that models trained on the distilled data achieve performance levels only marginally below those of domain-specific specialists, with the performance gap being effectively eliminated through subsequent RL training.

QUESTION. The distilled model is already close to the specialists. What can RL on its own attempts fix that SFT cannot?

ANSWER. Its own mistakes. SFT only shows prefixes the specialists wrote. RL samples the model's own attempts, mistakes included, and rewards the ones that work.

This is the first way to combine strong teachers with the student's own attempts: SFT on the teachers, then RL. V4 does it differently.

Source: DeepSeek-V3.2 report, §3.

6. V4: on-policy distillation replaces mixed RL

Same outline as V3.2; a new way to combine the specialists

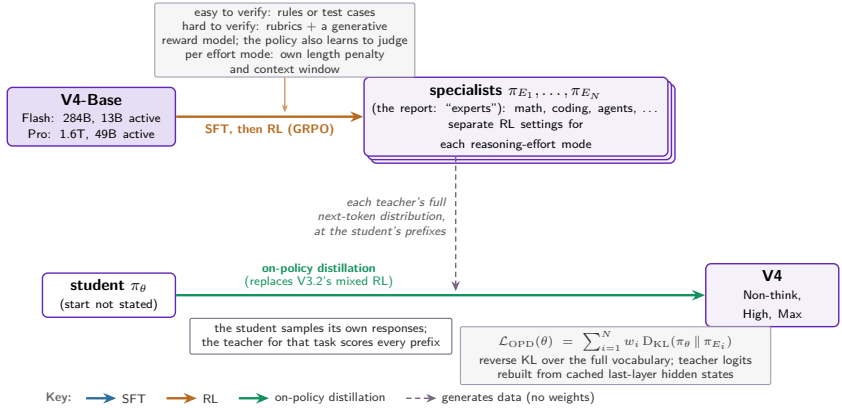
In the report: one substitution

> Following pre-training, we conducted a post-training phase to yield the final models of DeepSeek-V4 series. Although the training pipeline largely mirrored that of DeepSeek-V3.2, a critical methodological substitution was made: the mixed Reinforcement Learning (RL) stage was entirely replaced by On-Policy Distillation (OPD).

> After training multiple domain-specific experts via specialized fine-tuning and reinforcement learning, we employ multi-teacher On-Policy Distillation (OPD) as the primary technique for merging expert capabilities into the final model. OPD has emerged as an effective post-training paradigm for efficiently transferring the knowledge and capabilities of domain experts to a single, unified model. This is achieved by having the student learn from the output distributions of teacher models on its own generated trajectories.

Source: DeepSeek-V4 report, §5.1 and §5.1.2, pp. 29 and 32.

V4: specialists → on-policy distillation



One student then learns from all the specialists by **on-policy distillation** (**green**), which replaces V3.2's mixed RL. We first look at one change to how the specialists are trained, the judge, and then at on-policy distillation.

V4 trains the policy to be the judge

STATED RATIONALE: a reason given, without a reported measurement.

> **Generative Reward Model.** Typically, easy-to-verify tasks can be effectively optimized using simple rule-based verifiers or test cases. In contrast, hard-to-verify tasks traditionally rely on Reinforcement Learning from Human Feedback (RLHF), which necessitates extensive human annotation to train a scalar reward model. In the post-training phase of DeepSeek-V4 series, however, we dispense with these conventional scalar-based reward models. Instead, to address hard-to-verify tasks, we curate rubric-guided RL data and employ a Generative Reward Model (GRM) to evaluate policy trajectories. Crucially, we apply RL optimization directly to the GRM itself. In this paradigm, the actor network natively functions as the GRM, enabling the joint optimization of the model's evaluative (judging) proficiency alongside its standard generative capabilities.

V4 keeps V3.2's generative reward model, but the judge is now the policy itself: the same network writes answers and evaluations. The report says the judge is also trained with RL but gives no details, and no comparison with a scalar reward model. For a related method, see DeepSeek-GRM.

QUESTION. The judge is the policy itself. What could go wrong?

ANSWER. It may miss mistakes that it also makes. RL then rewards those mistakes.

Source: DeepSeek-V4 report, §5.1.1, pp. 29–30; Liu et al., "Inference-Time Scaling for Generalist Reward Modeling" (DeepSeek-GRM), 2025.

On-policy distillation, step by step

The **student** q_θ is the model being trained; the **teacher** q_{T} is the specialist for the prompt's domain.

1. The student writes its own response $y_1 \dots y_H$.
2. At every prefix $y_{<h}$, the teacher gives its next-token distribution.
3. The student's distribution moves toward the teacher's, by decreasing $\sum_h \mathrm{KL}(q_\theta(\cdot \mid x, y_{<h}) \,\|\, q_{\mathrm{T}}(\cdot \mid x, y_{<h}))$.
4. Repeat with fresh responses.

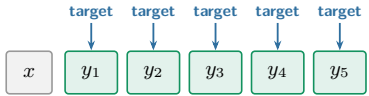
Example. The student writes "… so $3 \times 5 = 12$ …". After "$3 \times 5 =$", the teacher puts its probability on "15", so the student is corrected at the token where it went wrong.

It is **on-policy** because the prefixes are the student's own, as in RL; it is **distillation** because the feedback is a full distribution at every token.

Source: Not new in V4: GKD (Agarwal et al.) and MiniLLM (Gu et al.), ICLR 2024; DAgger (Ross et al., 2011).

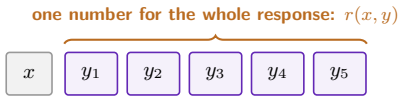
SFT, RL, and on-policy distillation compared

SFT on a teacher's responses
R1-Distill;
V3.2 specialist data



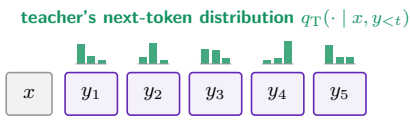
prefixes from the **teacher**;
one target token per position

RL with an outcome reward
R1-Zero; V3.2 mixed RL



prefixes from the **student**;
one score for the whole response

On-policy distillation
V4, V4.1



prefixes from the **student**;
the teacher's whole distribution
at every position

On-policy distillation uses the student's own prefixes, like RL, and gives feedback at every token, like SFT, but as a whole distribution.

In the report: the objective

$\{\pi_{E_1}, \pi_{E_2}, \ldots, \pi_{E_N}\}$, the OPD objective function is defined as:

$$\mathcal{L}_{\text{OPD}}(\theta) = \sum_{i=1}^{N} w_i \cdot \mathrm{D}_{\text{KL}}\left(\pi_\theta \,\|\, \pi_{E_i}\right). \tag{29}$$

In this formulation, w_i represents the assigned weight for each expert, typically determined by the relative importance of the expert. Computing the reverse KL loss $\mathrm{D}_{\text{KL}}\left(\pi_\theta \,\|\, \pi_{E_i}\right)$ requires sampling training trajectories from the student π_θ to maintain on-policy learning. The underlying logic ensures that the unified policy π_θ selectively learns from the specialized expert relevant to the current task context (e.g., aligning with the mathematics expert for math reasoning tasks and the coding expert for programming tasks).

- The report writes π_θ for the student and π_{E_i} for teacher i. There is one term per teacher, computed on prompts from its domain and weighted by w_i.
- The KL is **reverse**: the student's distribution π_θ comes first. The next slide shows what that rewards.

Source: DeepSeek-V4 report, §5.1.2, Eq. (29), p. 33.

What the reverse KL rewards

At one prefix, $\mathrm{KL}(q_\theta \| q_{\mathrm{T}}) = \sum_v q_\theta(v) \log \frac{q_\theta(v)}{q_{\mathrm{T}}(v)}$. It averages over the tokens the *student* would pick, and asks how much more the student likes each one than the teacher does.

after "$3 \times 5 =$"	teacher	A	B
`15`	0.60	1.00	0.50
`fifteen`	0.399	0	0
`12`	0.001	0	0.50
reverse KL		**0.51**	**3.02**

- Student A ignores `fifteen`, which the teacher also allows, and pays little.
- Student B gives half its probability to 12, which the teacher almost never says, and pays a lot.

QUESTION. Why use the reverse KL, $\mathrm{KL}(q_\theta \| q_{\mathrm{T}})$, rather than $\mathrm{KL}(q_{\mathrm{T}} \| q_\theta)$?

ANSWER. It mainly stops the student from saying what the teacher would not, and lets it commit to one good option instead of covering all of them. It is also an average over the student's own tokens, which is what on-policy sampling produces.

The full-vocabulary KL

STATED RATIONALE

> In handling the above OPD objective, prior works usually simplify the full-vocabulary KL loss into a token-level KL estimate at each token position, and reuse RL framework by replacing $\mathrm{sg}\left[\log \frac{\pi_{E_i}(y_t|x,y_{<t})}{\pi_\theta(y_t|x,y_{<t})}\right]$ (sg represents the stop gradient operation) as the per-token advantage estimate in the policy loss calculation. Although this approach is resource-efficient, it leads to high variance in gradient estimation and often causes training instability. Therefore, we adopt full-vocabulary logit distillation in our OPD. Preserving the complete logit distribution in calculating reverse KL loss yields more stable gradient estimates and ensures faithful distillation of the teachers' knowledge. In the following subsection, we describe the engineering efforts that make full-vocabulary OPD feasible at scale.

QUESTION. At one prefix, the KL can be estimated from the one token the student sampled, or summed exactly over all >100K tokens. Why is the sum less noisy?

ANSWER. It does not depend on which token happened to be sampled. Only the response itself is still random.

Storing >100K teacher scores per token is too expensive, so V4 stores each teacher's last hidden state and recomputes the scores during training.

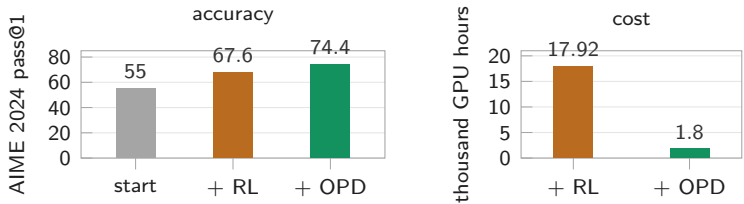
Source: DeepSeek-V4 report, §5.1.2, p. 33, and §5.2.2.

On-policy distillation versus RL in Qwen3

EXTERNAL

EXTERNAL: a study by another group, on a question the DeepSeek reports do not address.

PREDICT. Qwen3-8B starts from one checkpoint (55.0 on AIME 2024). It continues either with RL or with on-policy distillation from a larger Qwen3 teacher. Which scores higher, and which costs more?



ANSWER. On-policy distillation: 74.4 vs. 67.6, with a tenth of the GPU hours. The teacher's own training is not counted.

Source: Qwen Team, Qwen3 Technical Report (2025), §4.7, Table 21.

7. V4.1: same algorithms, better tasks

"What the model is trained on rather than how it is optimized"

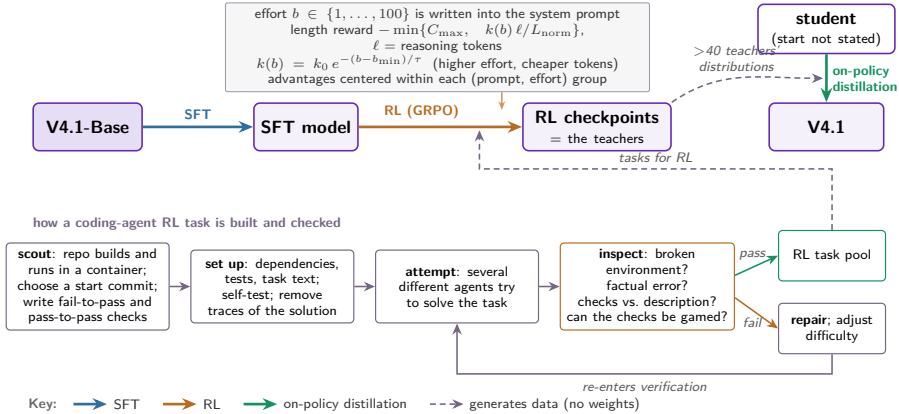
In the report: no new algorithms

STATED RATIONALE

> In this release, we refrain from introducing novel post-training algorithms. The overall recipe follows the standard paradigm of supervised fine-tuning (SFT) followed by reinforcement learning (RL) and on-policy distillation (OPD; Gu et al., 2024; Lu and Lab, 2025), without algorithmic modifications beyond well-established practices. Instead, our efforts are concentrated almost entirely on what the model is trained on rather than how it is optimized: we invest in large-scale, automated pipelines for data synthesis and environment construction. Concretely, the pipeline (i) synthesizes diverse, verifiable training tasks together with their reference solutions and reward signals, (ii) procedurally constructs and scales interactive agent environments in which trajectories can be collected and evaluated at low cost, and (iii) applies rigorous filtering, deduplication, and difficulty calibration to ensure data quality and curriculum balance. We find that, under a fixed and unremarkable optimization procedure, systematic improvements in the scale, diversity, and verifiability of synthesized data and environments account for essentially all of the observed gains. This observation echoes a broader lesson: at the current stage, the marginal return of engineering the data and environment pipeline substantially exceeds that of algorithmic novelty in post-training.

Source: DeepSeek-V4.1 report, §5.1, p. 25.

V4.1: the recipe and task construction



Same recipe as V4, ending with on-policy distillation from over 40 RL checkpoints. Next: the coding tasks (bottom), then the effort level (top). A **fail-to-pass** check fails before the fix and passes after it; a **pass-to-pass** check passes both times.

In the report: attempt, inspect, repair

> Then, multiple distinct agents attempt the task, and an independent quality-inspection agent reviews the environment together with the solving agents' trajectories, checking for environment issues, factual errors, mismatches between evaluation points and task descriptions, and hackability risks. If the inspection does not pass, a repair agent fixes all identified errors, adjusts evaluation points that are too easy or too difficult, and the task re-enters verification.

QUESTION. As with your trip verifier: a task comes from a real repository and its tests. Name two ways "all tests pass" could be a bad reward.

ANSWER. The environment is broken, so nothing ever passes. The tests miss part of the task. The model can edit the tests or look up the answer. The inspection agent checks for each of these.

Source: DeepSeek-V4.1 report, §5.1.1 and §5.1.3, pp. 26–28.

V4.1: an effort level from 1 to 100

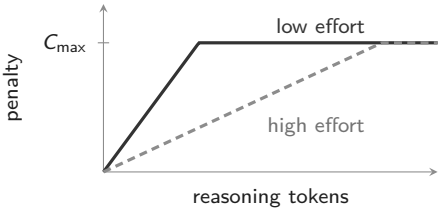
Long reasoning costs time and money, so V4.1 lets the user choose how much to spend. The system prompt carries an effort level b from 1 to 100:

```
Reasoning Effort:  {effort} (range 1-100; higher values request more
                   thorough reasoning)
```

During RL, the reward is correctness minus a **price per reasoning token**, up to a cap $C_{\max}$. The price falls as the effort rises:

$$k(b) = k_0\, e^{-(b-b_{\min})/\tau}.$$

k_0: the price at the lowest effort. τ: how many effort levels divide the price by e.



- The cap keeps a long correct answer worth more than a wrong one.
- GRPO compares each response only with others at the same effort, so it is judged against responses with the same price.

Source: DeepSeek-V4.1 report, §5.1.4, Eqs. (8)–(10), Table 2 (product tiers: low = 50, high = 75, max = 100). The penalty is $\min\{C_{\max},\ k(b)\,\ell/L_{\mathrm{norm}}\}$ for ℓ reasoning tokens and a fixed length L_{norm}.

Why an exponential price?

During RL, the model learns to keep thinking only while the next 4K tokens raise its accuracy by more than their price. **Diminishing returns**, where each 4K tokens helps less than the ones before, mean that thinking stops somewhere. A lower price makes it stop later.

effort level		25	50	75	100
price of 4K tokens		10	5	2.5	1.25
	gain	gain larger than the price?			
1st 4K tokens	+16	✓	✓	✓	✓
2nd 4K tokens	+8	–	✓	✓	✓
3rd 4K tokens	+4	–	–	✓	✓
4th 4K tokens	+2	–	–	–	✓
5th 4K tokens	+1	–	–	–	–
thinking length		**4K**	**8K**	**12K**	**16K**

Source: Illustrative numbers. V4.1's price falls by a factor of e every τ effort levels.

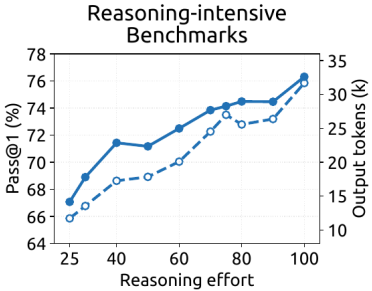
Each 4K tokens is worth half the ones before, so **halving the price buys one more 4K**.

A price that halves at each effort step, an exponential, makes each step add the same length.

A price that falls by equal amounts (10, 7.5, 5, 2.5) gives 4K, 8K, 8K, 12K.

Effort level, length, and accuracy

PREDICT. With the weights fixed, raising the effort from 25 to 100 makes the average response grow from about 12K to 32K tokens, 2.7 times longer. How much does accuracy grow?



ANSWER. From about 67% to 76%. Most problems are already solved with short reasoning; the extra tokens help only on the rest. Length grows roughly linearly in the effort, as the model on the last slide predicts.

Source: DeepSeek-V4.1 report, Figure 9 (left): V4.1-Flash on reasoning benchmarks; solid line pass@1, dashed line output tokens.

What to take away

Lesson	Where we saw it
Checkable rewards go far. RL with rule-checked rewards on a strong base taught long reasoning; every later release keeps GRPO.	R1-Zero; R1 to V4.1
The check is the reward. RL reinforces whatever the verifier accepts, so it must test what the prompt asks and resist hacks.	sorting test; trip verifier; V4.1's inspection
Environments are a main lever. Tools, data, and a verifier; useful tasks are hard to solve, easy to check, and solved sometimes.	V3.2's synthesized tasks; V4.1
Train on the student's own attempts. SFT on teachers' outputs never shows the student its own mistakes.	V3.2: SFT, then RL; V4: on-policy distillation
Not every task has a rule. A judge model scores responses against a rubric written for each prompt.	V3.2 and V4
Length can be trained. A price per reasoning token sets how long the model thinks; returns diminish.	V4.1's effort level
Read the evidence. Most claims compare releases that differ in many ways; few are controlled.	R1-0528 (release vs. release); synthesized tasks (controlled)