9. DeepSeek-R1

Chapter Abstract. R1-Zero showed that rule-based reinforcement learning could elicit long, self-checking reasoning from a base model. DeepSeek-R1 addresses the harder release problem: making that behavior readable, measurable, and usable. One verifiable prompt follows the cold-start and reasoning-RL branch into rejection-sampled data, then through the DeepSeek-V3-Base restart for mixed SFT and final RL. The same prompt then connects the released family to veRL execution, distilled learners, evaluation, and the observation and accounting contracts for reasoning outputs.

9.1. R1 as the Practical Successor to R1-Zero

A model receives a small request:

> Find all positive integer pairs (a, b) such that $a + b = 17$ and $ab = 60$. Return the pairs and a short verification.

This pair prompt is the chapter's running example. It has a deterministic checker—the two ordered pairs must satisfy both equations—while its response format still exposes readability, completeness, and final-answer placement. The same prompt can therefore supply a cold-start target, a rule-rewarded trajectory, a rejection-sampling candidate, and a final policy check without changing the underlying task.

"The answer is 5 and 12" points toward the solution while omitting both ordered pairs and their verification. A longer response may eventually become correct while burying the answer in a poorly formatted trace. A usable reasoning answer must turn the mathematical hunch into the requested objects and checks.

DeepSeek-R1 addresses that release problem. R1-Zero had already shown that rule-based reinforcement learning could push a base model toward long, self-checking behavior on verifiable tasks without an initial supervised fine-tuning stage. The R1 paper then describes a practical successor: cold-start examples shape a more readable actor, reasoning-oriented RL strengthens verifiable behavior, rejection-sampled outputs become supervised data, and a later RL stage broadens the policy beyond narrow reasoning tasks [47]. The repository presents the result as a public family containing R1-Zero, R1, and dense R1-Distill learners [49].

Chapter 7 and Chapter 8 supply the reusable reward, advantage, clipping, KL, and token-level GRPO mechanics. Those mechanics remain established prerequisites. R1 surrounds that update with cold-start supervision, generated-data refinement, the DeepSeek-V3-Base restart, final all-scenario RL, distillation, and release observations.

The policy chain, checker, acceptance gate, and data pools use the local notation in Table 9.1.

Symbol	Meaning in the R1 pipeline
q, o	Running reasoning prompt and candidate output.
$c(o, q)$	Deterministic task-correctness checker for output o on prompt q.
$A(o, q)$	Broader data-acceptance predicate; it may combine $c(o, q)$ with format, length, safety, and deduplication gates.
π_0	DeepSeek-V3-Base before R1-specific refinement.
π_{cs}	Cold-start policy after readable reasoning examples.
π_{rrl}	Policy after reasoning-oriented RL.
$\pi_{\text{sft2}}, \pi_{\text{R1}}$	Policy obtained by restarting from π_0 for mixed supervised refinement, and the final released R1 policy after all-scenario RL.
$D_{\text{reason}}, D_{\text{general}}$	Reasoning and general instruction pools used in the public recipe account.

Table 9.1.: Local notation for the R1 training stages and running prompt.

R1 is therefore both a training recipe and a release family. During training, prompts become supervised rows or rollouts, reward functions score sampled outputs, and optimizers change policies. During inference, an identified checkpoint or hosted route receives a request and returns observable state: final content, optional returned reasoning, usage, and termination. A benchmark captures one such execution under a declared protocol as a measurement rather than a training stage.

The recipe links checkpoints to later data and uses those data for later policy updates through a branched topology. The release identifies the artifacts and interfaces through which the resulting behavior can be observed. The paper and repository ground the public recipe and artifact family, while dated service documentation grounds hosted response fields [47, 49, 66].

The pair prompt will supply readable targets, rule rewards, accepted training rows, and final-answer checks. The policy sequence supplies the stage order; veRL supplies an inspectable third-party execution case; and the artifact and evaluation sections state what the released family can support.

9.2. Running Example: One Verifiable Prompt Across Four Stages

For this prompt, naming 5 and 12 falls short of returning both ordered pairs with a verification.

Return to the pair prompt from the opening. Its exact solution is easy to verify. With the mathematics settled, failures elsewhere are easy to see without hidden model outputs. An answer may omit a pair. A checker may accept too little, a filter may pass the wrong row, or a response schema may return the wrong field.

If a and b are the two roots of $t^2 - 17t + 60 = 0$, then

$$t^2 - 17t + 60 = (t - 5)(t - 12).$$

Thus the unordered values are 5 and 12. Because the prompt asks for ordered positive integer pairs, the solution set is

$$S(q) = \{(5, 12), (12, 5)\}.$$

Table 9.2.: One prompt carried schematically through the R1 stage vocabulary.

Stage witness	Schematic output or row	What the stage teaches
Base response	Names 5 and 12, but may omit ordered pairs or verification	The base policy can contain useful capability without stable answer form
Cold-start-style response	Factors $t^2 - 17t + 60$, names both ordered pairs, and marks the final answer	Supervised examples can make the answer easier to read before RL
Reasoning-RL scored output	Receives reward only when the checker accepts the ordered pairs and verification	Outcome rewards must match the task specification
Rejection-sampling row	An accepted candidate enters reasoning SFT data, while a rejected candidate remains a filter diagnostic	Data construction depends on both the acceptance predicate and the sampler; mixed SFT receives the row while restarting from DeepSeek-V3-Base
Final R1-style answer	Returns the concise pairs and verification, with the response schema deciding whether reasoning is visible	The release problem includes answer format and disclosure rules

Both pairs satisfy the two constraints: $5 + 12 = 17$ and $5 \cdot 12 = 60$. This fixture is too small to be a serious benchmark, but each later mechanism can be checked against the same answer requirement.

Chapter 8 defines the extractor and verifier that make a task rule-verifiable. Applied here, the binary checker returns one exactly when the final answer names both $(5, 12)$ and $(12, 5)$ and verifies the two equations. A weaker checker that accepts only the unordered set $\{5, 12\}$ solves a different task from this prompt's answer specification.

The base response is a baseline for four schematic R1-stage witnesses. It might write "the answer is 5 and 12" and stop, preserving the right numbers while failing the ordered-pair requirement. Cold-start supervision supplies a readable worked target, and reasoning-oriented RL scores whether the complete task rule is met. The resulting RL checkpoint can then generate a candidate that rejection sampling either accepts or rejects. An accepted candidate is a data row for the later mixed SFT run, which the public report initializes again from DeepSeek-V3-Base rather than from the reasoning-RL checkpoint. The final R1-style response should present the pairs and verification clearly, with the response schema controlling whether any reasoning transcript appears in its own field. R1-Zero remains the scientific comparison for direct RL from a base model, outside this four-step R1 example [47].

Table 9.2 keeps the answer requirement fixed while the prompt changes roles across training and release stages.

The same example can also be written as a pipeline row. Chapter 8 supplies the generic rollout-group, reward-tensor, and update fields. Table 9.3 adds the R1-specific lineage needed to distinguish the reasoning-RL sampler from the later mixed-SFT initialization. The values are schematic and local to the book.

Field	Purpose	Running-example value
`prompt_id`	Stable identity across sampling and filtering	Running pair prompt, train or validation split
`sampler_checkpoint`	Policy that generated the candidate	π_{rrl}
`candidate`	Output considered for supervised refinement	Both ordered pairs plus verification, or a logged failure
`checker_version`	Exact correctness rule applied to the candidate	Both pairs and both equations required
`acceptance_predicate`	Versioned format, length, safety, deduplication, and contamination gates	Local $A(o, q)$
`decision`	Accept or reject outcome and reason	Accepted into reasoning data or retained as a filter diagnostic
`destination`	Dataset receiving an accepted row	D_{reason}, a dataset rather than a policy checkpoint
`mixed_sft_init`	Weights used by the later supervised run	π_0 = DeepSeek-V3-Base

Table 9.3.: R1 lifecycle fields for the running prompt.

The example also shows why the chapter avoids copied reasoning traces. The technical question asks which output property is being defined, measured, filtered, or returned to a user. A short, schematic transcript is enough to explain the separation between an internal training signal, a visible reasoning region, and a final answer. Long traces would distract from that separation and risk using style as proof.

9.3. The R1 Training Pipeline as a Sequence of Policies

The R1 paper describes the final model as the result of several post-training stages rather than one optimization step [47]. The stages have a chronological order, but their checkpoint and data edges do not form one linear initialization chain. A source-faithful shorthand is

$$\pi_0 \xrightarrow{\text{SFT}(D_{\text{cs}})} \pi_{\text{cs}} \xrightarrow{\text{reasoning RL}} \pi_{\text{rrl}},$$

$$\pi_{\text{rrl}} \xrightarrow{\text{sample and filter}} D_{\text{reason}},$$

$$\left(\pi_0, D_{\text{reason}} \cup D_{\text{general}}\right) \xrightarrow{\text{two-epoch mixed SFT}} \pi_{\text{sft2}} \xrightarrow{\text{all-scenario RL}} \pi_{\text{R1}}.$$

Here π_0 is DeepSeek-V3-Base. Cold-start SFT produces π_{cs}, and reasoning-oriented RL produces the sampler policy π_{rrl}. That sampler generates candidates for D_{reason}. The public report then states that the mixed SFT run starts from DeepSeek-V3-Base and trains for two epochs on the selected reasoning rows plus general rows. The final all-scenario RL stage transforms π_{sft2} into π_{R1}.

Figure 9.1 makes that branch explicit. A policy edge changes weights; a data edge creates or combines supervised rows.

The reasoning-RL checkpoint supplies generated data to mixed SFT; DeepSeek-V3-Base supplies the mixed-SFT starting weights.

The starting symbol π_0 names DeepSeek-V3-Base before R1-specific post-training. Its pretraining process remains outside this recipe and is covered in Chapter 6.

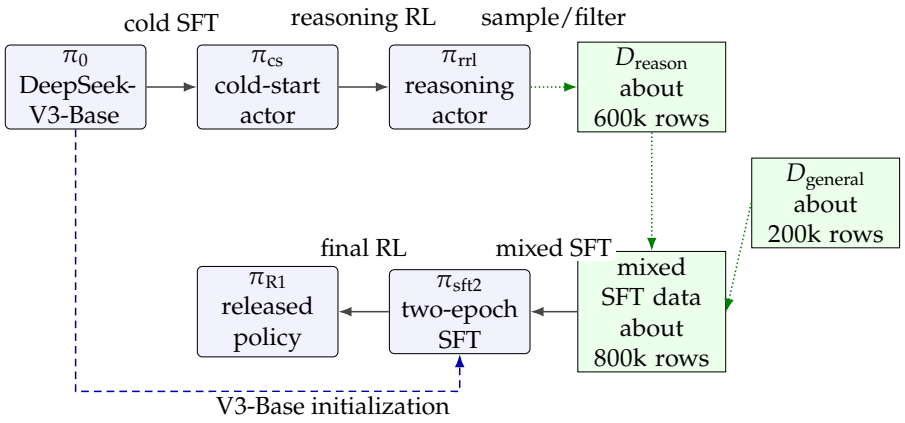

Figure 9.1.: The public R1 training topology. The reasoning-RL checkpoint generates the selected reasoning rows, but the two-epoch mixed SFT run restarts from DeepSeek-V3-Base. Rounded boxes are policies and square boxes are datasets; solid arrows update policies, dotted arrows construct data, and the dashed edge carries the mixed-SFT initialization.

The same base checkpoint supplies two starts inside the R1 recipe: the cold-start actor and the later mixed-SFT actor.

The public paper also gives several quantitative anchors for the recipe [47]. Table 9.4 uses them to show the relative scale and placement of the stages. These source-reported recipe facts provide scale and placement rather than local reproductions; exact private prompts, filters, optimizer settings, and reward coefficients remain unspecified.

Table 9.4.: Source-reported R1 recipe facts and their place in the training sequence.

Reported fact	Role in the R1 recipe
Thousands of cold-start examples	Initialize a more readable actor before reasoning-oriented RL
Two RL stages	One stage targets reasoning-intensive prompts; the later stage covers broader scenarios
Two SFT stages	Cold-start SFT produces the reasoning-RL actor; the later mixed SFT restarts from DeepSeek-V3-Base
About 600k reasoning samples	Selected generated trajectories become reasoning-related supervised data
About 200k non-reasoning samples	Writing, factual QA, self-cognition, translation, and related rows broaden behavior
About 800k mixed samples, two SFT epochs	The 600k reasoning and 200k non-reasoning rows fine-tune DeepSeek-V3-Base into π_{sft2}
The same 800k-sample set for distillation	SFT-only transfer trains the released Qwen and Llama dense learners

Scope 9.3.1 (Match each statement to its object)**.** GRPO is an optimizer-level update mechanism; R1 is a staged training recipe that places such updates around supervised cold-start data, rejection-sampled data construction, mixed SFT, and a final all-scenario RL stage. The reasoning-RL checkpoint samples the selected reasoning rows, whereas DeepSeek-V3-Base initializes mixed SFT. Rejection sampling constructs training rows, mixed SFT restarts from DeepSeek-V3-Base, and final all-scenario RL is a later policy update. Extending a statement from one object to another requires a source that makes the connection [125, 47].

The policy-and-data notation prevents three mistakes. The first is to collapse R1 into "RL after R1-Zero," which misses the supervised cold-start and supervised refinement stages. The second is to assume that π_{rrl} initializes mixed SFT merely because it generates the accepted rows. The third is to treat the public

topology as a complete training recipe, although the sources establish only the checkpoint and data edges and leave the full prompt mixture, filtering code, and optimizer settings undisclosed.

In the running example, π_0 might produce a terse or incomplete answer. After cold-start supervised training, π_{cs} is expected to be more likely to use a readable pattern: set up $t^2 - 17t + 60 = 0$, factor it, and state a concise conclusion. After reasoning-oriented RL, π_{rrl} should be more likely to satisfy the checker on tasks with clear solutions. Its selected outputs enter D_{reason}. A new mixed-SFT run starts from π_0 and uses those rows plus general data to produce π_{sft2}. The final stage then brings reasoning behavior back into a broader distribution of user scenarios.

The R1 paper supports this public stage sequence; repositories and model cards identify the released artifacts, while dated service documentation describes the hosted interface. No one of those sources specifies the entire training system.

Concept-to-code map

The training-stage correspondence uses the pinned DeepSeek-R1 report [47]; its public locator is `https://arxiv.org/abs/2501.12948`. Released checkpoint identities and license terms map separately to `https://github.com/deepseek-ai/DeepSeek-R1`.

- **Policy edges.** Cold-start SFT, reasoning RL, mixed SFT, and all-scenario RL map to the report's public stage sequence.
- **Data edge.** Sampling and filtering map the reasoning-RL policy to selected reasoning rows rather than to the mixed-SFT initialization.
- **Artifact edge.** The repository and model cards identify released R1 and R1-Distill artifacts, not the undisclosed private training system.

Label each edge by the object it carries. Cold-start SFT carries selected prompt–target rows into a new policy. Reasoning RL carries rewards and rollout comparisons into another weight update. Rejection sampling carries generated candidates through filters into a new supervised dataset. The mixed-SFT stage uses that dataset to train a fresh DeepSeek-V3-Base branch, rather than merely continuing the reasoning-RL weights. Final RL again carries reward signals into policy updates, while release edges carry artifacts and evaluation records.

9.4. Cold-Start Data and Readable Reasoning

R1-Zero asks what reinforcement learning can elicit from a base model. DeepSeek-R1 asks how that optimization changes when it begins from a response format that is already easier to read and score. The R1 paper places cold-start data before the main reasoning-oriented RL stage for that purpose [47]. Supervision changes the starting actor, and the later RL stage then optimizes that actor.

Write the supervised cold-start set as

$$D_{cs} = \{(q_i, y_i)\}_{i=1}^{n}$$

. It contains prompts and readable target responses used to produce the actor

$$\pi_{cs} = \mathrm{SFT}(\pi_0, D_{cs}),$$

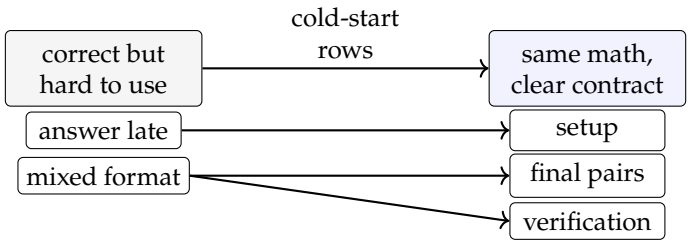

Figure 9.2.: Cold-start examples preserve the pair problem while moving the same mathematical content into a response shape that downstream checks can parse and compare.

π_0 is the base policy, D_{cs} supplies supervised response shape, and π_{cs} starts the later RL stage.

before large-scale reasoning RL begins.

The public report names how DeepSeek explored constructing these rows. The routes included few-shot prompting with a long chain-of-thought example, prompting models to produce detailed answers with reflection and verification, collecting readable-form R1-Zero outputs, and post-processing generated results with human annotators [47]. These source-reported routes describe how rows were constructed while leaving each route's retained count, the full annotation rubric, and the exact acceptance thresholds undisclosed.

For the running prompt, an unreadable but correct answer might list many factor pairs, repeat itself, and eventually mention $(5, 12)$. A cold-start target would instead teach a stable pattern: introduce the variable t, form the polynomial $t^2 - 17t + 60$, factor it, map the roots to ordered pairs, and end with a short verification. Both answers can contain the same mathematical content. Their usefulness differs because a reader, evaluator, or service client can more reliably find the final answer in the second response.

Figure 9.2 contrasts those response shapes without treating readability as a measured RL gain.

Cold-start rows intervene in format and provenance before they can contribute to an RL result. Here readability means stable answer placement, coherent language, a concise summary, and avoidance of chaotic formatting in addition to final-answer correctness.

The R1 paper describes cold-start data as addressing readability problems seen in R1-Zero, including poor readability and language mixing [47]. It also describes a readable pattern that places reasoning and summary in different fields. The description establishes only that DeepSeek used thousands of cold-start examples to initialize the DeepSeek-V3-Base checkpoint before the reasoning-oriented RL stage, and the stated motivation is to improve readability and provide a better starting point for the later recipe.

No local experiment reproduces the numerical effect of a cold start. A small mechanism example is still useful. Suppose four outputs for the running prompt are all mathematically correct, but only two have a clearly marked final answer and verification. If a supervised cold-start set consistently demonstrates the clearer form, the supervised update can increase the probability of that form before any reward is applied. Later RL can then operate on a policy whose outputs are easier to score and easier to use. This prompt and scoring rule are pedagogical examples rather than reported DeepSeek choices.

A practical cold-start recipe describes how those readable rows were made. Table 9.5 gives the minimum recipe fields an outside experiment should keep.

The table derives its controls from the public account rather than private DeepSeek data.

Table 9.5.: A cold-start data recipe captures provenance and filtering as well as the readable examples.

Recipe step	Required field	Risk controlled
Seed source	Prompt origin, license, split, and benchmark-overlap status	Contamination and unclear provenance
Readable target rule	Required final-answer marker, verification style, language policy, and maximum length	Style drift and unscorable answers
Rewrite or annotation rule	Whether rows are human-written, model-generated, rewritten, or filtered	Unsupported attribution of data authorship
Quality filters	Correctness, readability, language consistency, safety, and duplicate checks	Training on attractive but wrong traces
Validation split	Held-out prompts and acceptance criteria kept away from SFT updates	Overfitting to the cold-start examples
Versioning	Dataset digest, tokenizer, prompt template, and reviewer notes	Irreproducible stage comparisons

Cold-start provenance should be read as part of the mechanism. A human-authored example, a model-generated example accepted by a verifier, and a rewritten trace can share the same visible format while carrying different error and contamination risks. Filtering can enforce final-answer correctness, language, format, or readability without proving every intermediate statement. The public count therefore describes the reported scale; an exact reproduction would additionally need source classes, generator versions, acceptance rules, deduplication, and mixture weights.

9.5. Reasoning-Oriented RL After Cold Start

After the actor acquires a more readable starting format, R1 applies reasoning-oriented reinforcement learning. The R1 paper focuses this stage on tasks with clear solutions and correctness rewards. Its examples span coding and mathematics as well as science and logic [47]. This stage preserves the main lesson of R1-Zero: for many reasoning tasks, the training signal can come from outcomes that are verifiable by rules rather than from a human-written reasoning trace for every prompt.

Chapter 8 supplies the GRPO update. The R1-specific facts are that reasoning RL begins after cold-start SFT and that the public reward adds a language-consistency term to task accuracy [47]. The underlying GRPO method comes from DeepSeekMath [125].

Correctness, format, language, and selection occupy their own reward fields until a declared rule combines them.

The R1 report makes one R1-specific reward combination public. It defines a language-consistency reward as the proportion of target-language words in the chain of thought and directly adds that value to reasoning-task accuracy:

$$r_{\text{reason}}(q, o) = r_{\text{acc}}(q, o) + r_{\text{lang}}(o), \qquad r_{\text{lang}}(o) = \frac{N_{\text{target}}(o)}{N_{\text{CoT}}(o)}.$$

The report says this language alignment slightly reduced task performance in its ablation while improving alignment with the desired readable behavior [47]. The exact word-counting implementation, multilingual prompt mixture, and full set of task checkers remain undisclosed.

Accuracy still depends on the declared checker. For the running prompt, a rule that accepts only the values 5 and 12 trains a different task from one that requires both ordered pairs and their verification. The next stage also applies selection gates to decide whether a rewarded output becomes supervised data; those gates should not be silently reported as reward terms.

For an R1-style implementation, reward terms and acceptance gates should be decomposed before either is reduced. Table 9.6 applies the public direct-sum rule to the running prompt and treats later filters independently.

Table 9.6.: An R1 reasoning stage directly sums the reported correctness and language terms while treating local format, length, and safety gates independently.

Component	Running-example rule	Combination role	Public-status note
Correctness	Both ordered pairs and both equations verify	Reward term	Rule reward category and direct addition are public; checker here is local
Language	Target-language words divided by counted CoT words	Reward term	Definition is public; tokenizer and counting details remain unspecified
Format	Final answer is locatable and verification is present	Selection gate	R1-Zero format reward is public; this R1 gate is local
Length	Response stays under the declared maximum token budget	audit gate	Cost and readability control; no DeepSeek coefficient is reported
Safety	No unsafe instruction-following content outside the math task	reject gate	Belongs to broader release fitness beyond narrow math correctness

The language term depends on a counting convention. In a mixed response containing $5 \times 12 = 60$, ordinary words, tokenizer pieces, mathematical symbols, and code-switching can produce different target-language fractions. The reported formula fixes the conceptual ratio, but a reproduction must pin segmentation, language identification, excluded regions, and zero-length handling before the reward becomes executable.

9.6. Rejection Sampling, Supervised Refinement, and Final RL

The next stage in the public R1 recipe turns model-generated candidates into supervised data. After reasoning-oriented RL, DeepSeek reports using the resulting checkpoint to generate trajectories and then retaining selected outputs for supervised fine-tuning [47]. In this use, the model produces candidate training examples that are filtered before they become supervised targets. The resulting reasoning-RL checkpoint is therefore the sampler for this data-construction branch. The report states that the subsequent mixed SFT run starts from DeepSeek-V3-Base and trains for two epochs, a fresh initialization relative to the sampler checkpoint.

Figure 9.3 keeps both outcomes visible: accepted candidates become supervised rows, while rejected candidates reveal sampler and filter failures.

Definition 9.6.1 (Rejection sampling) *For a prompt q, rejection sampling draws K candidate outputs*

$$o_1, \ldots, o_K \sim \pi(\cdot \mid q)$$

from a policy and keeps the candidates satisfying a selection predicate $A(o_i, q) = 1$. The accepted pairs (q, o_i) can then be used as supervised training data.

Definition 9.6.1 makes the selection predicate $A(o, q)$ part of rejection sampling. The R1 report says DeepSeek sampled multiple responses per prompt and retained correct candidates. Some expanded reasoning data used DeepSeek-V3 as a generative judge supplied with the ground truth and model prediction. Reported readability filters removed chain-of-thought rows with mixed languages, long paragraphs, or code blocks [47]. These named checks leave the complete private predicate undisclosed. In the running example, suppose eight candidates are sampled and three pass the acceptance predicate. The accepted fraction is

$$\frac{3}{8} = 0.375.$$

Rejection sampling filters model outputs into SFT targets; the acceptance fraction measures one sampler–filter combination rather than standalone capability.

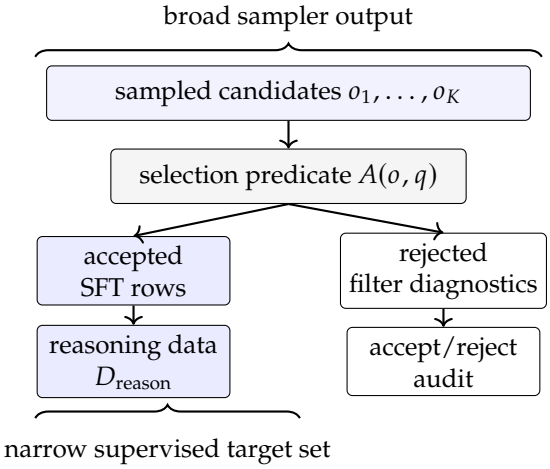

Figure 9.3.: Rejection sampling turns model-generated candidates into supervised targets only after explicit selection stages have been applied.

The 3/8 result measures how many sampled outputs passed a particular filter for a particular prompt set and sampling configuration, rather than the model's general reasoning capability. If the predicate is stricter, the accepted fraction may fall even when many outputs are mathematically correct. If the predicate is too weak, the accepted set may contain poor supervised targets.

The report gives about 600k selected reasoning samples. It also describes about 200k non-reasoning samples for writing, factual QA, self-cognition, translation, and related capabilities. Those general rows reuse portions of the DeepSeek-V3 SFT data; for some tasks, DeepSeek-V3 generates a potential chain of thought before the answer, while simple queries receive no chain of thought [47]. These source facts establish the broad data origins while leaving the full prompt pool, mixture, and private filter stack undisclosed.

The accepted reasoning rows can be combined with general instruction rows from other sources. Write the mixed supervised set as

$$D_{\text{sft}} = D_{\text{reason}} \cup D_{\text{general}},$$

where D_{reason} contains selected reasoning examples and D_{general} contains general assistant examples. The union permits unequal sizes and different sources. The next policy state is

$$\pi_{\text{sft2}} = \text{SFT}^{(2\text{ epochs})}(\pi_0, D_{\text{sft}}), \qquad \pi_0 = \text{DeepSeek-V3-Base}.$$

This initialization edge differs from the data edge $\pi_{\text{rrl}} \to D_{\text{reason}}$.

For the running prompt, D_{reason} could contain accepted solutions that derive $t^2 - 17t + 60 = 0$, return both ordered pairs, and verify the two constraints. D_{general} could contain an instruction such as "summarize the answer in one sentence" or "translate the final answer into another language." The union notation permits joint training even when the examples come from different samplers and need different evaluation metrics. In the public topology, the accepted pair solution teaches a mixed-SFT actor whose optimizer starts afresh from DeepSeek-V3-Base.

This stage introduces two audit risks. The first is selection bias. Accepted outputs reflect both the model and the filter, producing a selected sample of model behavior. The second is contamination risk. If prompts or solutions overlap with public benchmarks, the resulting model may look stronger on those benchmarks without improving the underlying general skill. This is a general risk rather than an accusation about a specific R1 contamination problem. Any rejection-sampled dataset used for supervised refinement has to be audited for it. Table 9.7 lists the fields that make such an audit possible.

Chapter 7 supplies the generic contrast between rule-based and model-based preference signals. The final public R1 stage applies those signal classes over a broader scenario distribution. The R1 paper describes this stage as improving helpfulness and harmlessness while refining reasoning, with different treatment of reasoning and general data [47]. Reasoning prompts in mathematics, code, and logic continue to use the R1-Zero-style rule rewards. General prompts instead use reward models to capture human preferences, with a distribution of preference pairs and training prompts similar to the DeepSeek-V3 pipeline. Helpfulness scoring sees only the final summary, whereas harmlessness scoring sees the whole

Pipeline field	What to store	Why it matters
Sampling fanout	K, temperature, top-p, maximum tokens, seed policy, and checkpoint id	Acceptance rates are meaningless without sampler context
Checker stack	Rule checker, format checker, optional judge, safety filter, and their versions	Accepted rows depend on the whole predicate
Deduplication	Exact-match, normalized-answer, and near-duplicate policy	Prevents repeated easy traces from dominating SFT
Split assignment	Train, validation, and audit splits after filtering	Keeps promotion checks independent from training rows
Acceptance audit	Counts by accept reason and reject reason, sampled examples, and reviewer notes	Shows whether the filter is too weak or too narrow
Contamination audit	Benchmark overlap checks and source exclusions	Protects later evaluation conclusions
Dataset version	Manifest, row count, digest, reward version, and source licenses	Makes the supervised refinement set rerunnable

Table 9.7.: A full rejection-sampling pipeline captures candidates, filters, dedupe, splits, acceptance reasons, and dataset versioning.

Stage object	Mechanism	Failure if collapsed
Rejection sampling	Generated candidates become supervised data only after the predicate $A(o, q)$ accepts them	Reporting an accepted fraction such as 3/8 as if it were a benchmark score
Mixed SFT	Reasoning rows and general assistant rows can share a supervised update from DeepSeek-V3-Base while serving different behavior goals	Treating the reasoning-RL sampler as the mixed-SFT initialization, or scoring a translation row with the math checker
Final all-scenario RL	Rule rewards score reasoning prompts; reward models score general preferences, with different helpfulness and harmlessness views	Optimizing only narrow verifiable answers while missing preference or safety failures in open scenarios

Table 9.8.: Failure checks for the post-RL refinement stages.

response, including the reasoning process and summary. These are public signal assignments; the reward models, exact mixture, and coefficients remain private.

Example 9.6.1 (Two reward routes in final all-scenario RL) The pair prompt can remain on the reasoning route: a rule checks both ordered pairs and their verification. A general request such as rewriting that answer for a novice requires preferences beyond the math rule; a preference reward model can instead compare candidate summaries for helpfulness. The harmlessness check examines each complete response alongside the selected summary. One optimizer stage may consume both routes, but their graders observe different objects.

Algorithm 10 keeps rejected candidates in the audit trail, appends accepted trajectories to D_{reason}, and mixes those rows with general data before the final all-scenario RL stage.

Algorithm 10: Public R1 rejection-sampling refinement workflow, showing public stage order alongside the undisclosed filters and prompt mixtures.

Input: Prompts Q; sampler policy π_{rrl}; base policy π_0; acceptance predicate A; general SFT rows
Output: Mixed supervised refinement set and transition log

foreach *prompt* $q \in Q$ **do**
  sample candidate trajectories $o_1, \ldots, o_K \sim \pi_{\text{rrl}}(\cdot \mid q)$;
  foreach *candidate* o_i **do**
    apply $A(o_i, q)$ and store the accepted or rejected reason;
  append accepted reasoning candidates to D_{reason};
merge D_{reason} with source-bounded general SFT rows;
initialize from DeepSeek-V3-Base and train the mixed supervised policy for two epochs;
list which private mixture and filter details remain unknown;
hand the refined policy to the final all-scenario RL stage;
return *stage log with candidates, accepted rows, and unknown filter details*;

The refinement algorithm first samples the full fanout $o_1, \ldots, o_K$ and retains an accept-or-reject reason for every candidate. Next, only rows satisfying $A(o_i, q)$ enter D_{reason}; rejected rows remain available for diagnosis and never become supervised targets. The accepted reasoning rows are then merged with source-bounded D_{general}, and the two-epoch supervised update starts from π_0 and produces π_{sft2}. Finally, π_{sft2} enters all-scenario RL as a new optimization stage. Candidate construction, supervised optimization, and the final policy update are three different state transitions. The first transition contributes data to the second, whose starting weights come from π_0.

The restart is the easily missed edge. The reasoning-RL checkpoint generates candidates; filters select reasoning rows; general data joins them; but the reported two-epoch supervised refinement trains a DeepSeek-V3-Base restart on that mixed dataset. The generator is therefore an upstream data source, not necessarily the initialization of the mixed-SFT learner. Final RL begins after that supervised branch. Keeping data lineage separate from weight initialization prevents the stage graph from collapsing into a simple chain.

9.7. Executing an R1-Style Loop with veRL

The public R1 recipe specifies policy and data transformations, but an experiment also needs machinery for rollouts, rewards, updates, and checkpoint exchange. veRL provides an inspectable case study: its HybridFlow-based RL library connects FSDP or Megatron training to vLLM or SGLang rollouts [138, 128, 137].

The useful HybridFlow mental model is hierarchical. One controller expresses the inter-model RL dataflow in algorithm order, while worker groups execute the distributed computation inside each model. A controller-level rollout or update call can therefore stand for a multi-GPU program whose workers manage their own collectives. This split keeps the algorithm readable without pretending that rollout, reward inference, and policy optimization are local function calls [128].

Algorithm 11: Conceptual veRL controller iteration. The algorithm delegates GRPO mathematics to the earlier GRPO treatment and exposes the generation-to-training handoff, checkpoint exchange, and scheduling invariant.

Input: Prompt records; logical actor; reward adapter; generation and training layouts; pinned update configuration; stop rule
Output: Updated actor checkpoint and an auditable iteration record

```
pin the framework, recipe, model, tokenizer, reward, and backend revisions;
while the declared training stop rule is false do
    place the logical actor in the generation layout and admit one prompt batch;
    generate the configured rollouts while carrying the established prompt-group identity on
      every response;
    compute reward records and transfer the rollout batch to the training layout;
    reject any scheduling transformation that changes prompt-group membership;
    apply the pinned update contract and checkpoint the resulting actor state;
    preserve rollout, reward, checkpoint, and system diagnostics;
    reshard or publish the updated actor weights back to the generation layout;
return the final recorded checkpoint and iteration history;
```

The actor also serves two systems roles. Rollout favors a generation layout and room for the KV cache. Training favors sharded parameters, gradients, and optimizer state. HybridFlow's 3D-HybridEngine changes the distributed layout between these phases. It reuses resident training shards rather than preserving another full actor copy. Rollout records flow to the update; updated weights return to generation. The backend names are examples of this general transition [128].

Figure 9.4 separates the readable controller flow from the distributed systems that carry it.

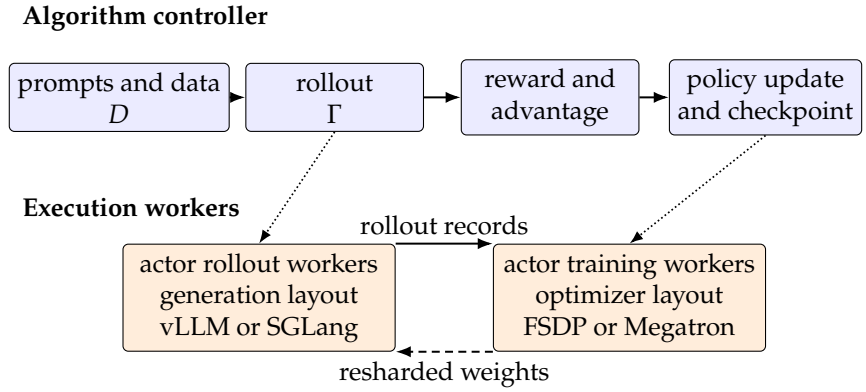

Figure 9.4.: HybridFlow separates centralized RL orchestration from distributed execution. One logical actor uses a generation layout for rollouts and a training layout for updates, so rollout records and re-sharded weights move in opposite directions.

Algorithm 11 turns the controller lane into one source-bounded iteration. Chapter 8 supplies the GRPO update contract; this routine records the systems handoffs needed to execute it.

The algorithm has three ordered phases. First, setup pins the revisions and admits a prompt batch. Then, each loop iteration preserves group identity while rollouts, rewards, and weights cross layouts. Finally, the routine returns the recorded checkpoint and iteration history.

The established prompt-group key remains attached as records move between layouts. Scheduling may change where a response runs, but it cannot change which responses belong to the same update group.

Generation and training layouts may differ, but the prompt-group identity must survive the handoff.

Table 9.9.: Execution surfaces in the veRL case study.

Execution object	veRL surface	Mechanical consequence
Logical actor	Generation and training layouts for one evolving policy	Rollout and optimization need different distributed state
Rollout batch	Records transferred from generation workers to training workers	Rewards and prompt-group identity cross the layout boundary
Updated actor	Checkpointed and resharded weights returned to rollout workers	The next generation phase observes the new policy state
Unequal sequences	Token-aware batching and sequence balancing	Equal example counts need not imply equal worker load
Large-model recipe	Revision-pinned veRL and R1-recipe configuration	Framework support remains public, evolving, and distinct from DeepSeek's private run [136, 135]

Unequal response lengths make the handoff a systems problem. For the running pair prompt, illustrative lengths $(64, 128, 768, 896)$ give one naive worker 192 valid response tokens and another 1,664. Pairing a short response with a long one reduces that imbalance without changing group membership. Token-aware batching and sequence balancing therefore alter placement, not the logical group comparison defined by GRPO.

Worker groups may share a GPU pool or occupy separate pools. Colocation can conserve hardware by time-sharing capacity; separation can permit concurrent stages at the cost of more devices. These are public framework tradeoffs, not evidence that DeepSeek used HybridFlow, these placements, or these backends [136].

The companion example in `chapter_deepseek_r1/` supplies a prompt fixture, a toy reward adapter, and an example veRL configuration. The adapter checks both ordered pairs, the answer marker, a short verification, and a local length gate. The configuration selects four rollouts and a backend such as vLLM. Running the adapter alone tests reward calculation, not training. A training claim must also preserve the pinned framework and recipe revisions, configuration, backend, hardware, command, checkpoints, and monitoring state.

The printed driver exercises that adapter on three authored responses. It awards 1.15 to the correct formatted answer, zero to a parse failure, and 1.05 to the correct pair with a false verification string. The trace checks reward semantics only; it contains no rollout or policy update.

Executable checkpoint: Three reward-adapter outcomes

```python
#!/usr/bin/env python3
"""Exercise the Chapter 9 reward adapter on three authored responses."""

from mini_r1_reward import score_response

responses = {
    "accepted": ("Final answer: (5,12) and (12,5). Verification: 5+12=17 and 5*12=60."),
    "parse_failure": "No parseable ordered pair is present.",
    "false_verification": (
        "Final answer: (5,12) and (12,5). Verification: 15+12=17 and 15*12=60."
    ),
}
scores = {name: score_response(text) for name, text in responses.items()}

assert scores["accepted"]["reward"] == 1.15
assert scores["accepted"]["reasons"] == []
assert scores["parse_failure"]["reward"] == 0.0
assert "parse-failure" in scores["parse_failure"]["reasons"]
assert scores["false_verification"]["reward"] == 1.05
assert scores["false_verification"]["format_ok"] is False

if __name__ == "__main__":
    for name, score in scores.items():
        print(name, "reward=", score["reward"], "reasons=", score["reasons"])
```

Run: `python3 chapter_deepseek_r1/mini_r1_reward_checkpoint.py`

Scope 9.7.1 (Third-party R1-style implementation)**.** A third-party R1-style implementation claim must name the framework revision, recipe revision, configuration, model, reward interface, and experiment record being discussed. It may support claims about executable mechanics, but it does not by itself support claims about DeepSeek's private R1 training run.

The veRL R1 recipe is described as under development and tied to a particular veRL revision [135]. It serves here as an executable R1-style case study, not as a reconstruction of DeepSeek's private pipeline.

Concept-to-code map

The execution case study uses the pinned veRL repository at `https://github.com/verl-project/verl` and the developing R1 recipe at `https://github.com/verl-project/verl-recipe/tree/main/r1`.

- **Controller and worker split.** HybridFlow maps one logical actor to distinct rollout and training layouts [128].
- **Group-relative update.** The public GRPO interface maps sampled prompt groups and reward records to the configured policy update [137].

► **Local semantic checkpoint.** The reward adapter and authored fixtures map to `chapter_deepseek_r1/mini_r1_reward_checkpoint.py`; they test reward semantics without claiming to reproduce training.

The mapping remains conceptual rather than an exact R1 reconstruction. veRL can own rollout workers, reward calls, reference-policy scoring, advantage construction, actor updates, and checkpoint publication, but DeepSeek's private prompt mixture, reward implementations, topology, coefficients, and recovery policy are not supplied by naming those surfaces. A faithful R1-style study must populate and version each owner under its own declared configuration.

9.8. R1-Distill Dense Learners

A distillation comparison needs the teacher, learner artifact, evaluation setup, and transfer limit.

The R1 release contains more than the large R1 and R1-Zero artifacts. The repository and paper also describe dense distilled models based on Qwen and Llama families [47, 49]. Their public artifacts support statements about teacher identity, learner families, and reported scores while leaving the private distillation-data pipeline undisclosed.

Chapter 7's distillation definition supplies the generic teacher-to-learner mechanism. The R1-specific question is which teacher-generated dataset trained which released dense learners and whether those learners also received reinforcement learning. A defensible artifact comparison names the teacher, learner base, model size, and artifact source. It also states the evaluation setup and limitation; artifact identity alone leaves behavioral equivalence unproven.

The source-backed relation is narrower than behavioral equivalence: the same curated 800k-sample mixture used in R1's second SFT stage fine-tuned dense learners that were released as R1-Distill artifacts. The public sources leave the complete sampler, filter, mixture, and rejection rules undisclosed for that data. The R1 report makes the optimizer split explicit: the released distilled learners received only supervised fine-tuning on the curated 800k-sample set; they did not receive an RL stage [47]. Their "R1" name identifies the teacher-derived data lineage rather than learner-side repetition of the R1 RL recipe.

The released Qwen and Llama learners each have model cards [44, 46, 42, 43, 45, 41].

Table 9.10.: R1 distilled artifact lines treated as release context.

Released distill line	Base family named by artifact	Treatment here
DeepSeek-R1-Distill-Qwen-1.5B, 7B, 14B, 32B	Qwen2.5 or Qwen2.5-Math bases listed by the README	Artifact map and headline context with model-card checks
DeepSeek-R1-Distill-Llama-8B, 70B	Llama-3.1 or Llama-3.3 bases listed by the README	Artifact map and headline context with model-card checks

Reported construction	AIME pass@1	AIME cons@64	MATH-500 pass@1	LCB pass@1
DeepSeek-R1-Zero-Qwen-32B, direct RL on Qwen-32B-Base	47.0	60.0	91.6	40.2
DeepSeek-R1-Distill-Qwen-32B, SFT on R1-generated data	72.6	83.3	94.3	57.2

Table 9.11.: The R1 paper's reported 32B comparison treats distillation as a source-reported transfer result rather than smaller-model packaging.

The R1 paper's strongest R1-Distill lesson is comparative rather than merely compressive. In its 32B comparison, the learner trained on R1-generated data outscores a direct large-scale reinforcement-learning run on a Qwen-32B base [47]. Table 9.11 gives the full model identities and scores as source-reported results rather than a local reproduction.

The running example gives a small interpretive guard. Suppose the teacher produces a long derivation for the pair problem and the learner produces a similarly long derivation. If the learner still misses the ordered-pair requirement, then the release artifact has not demonstrated preservation of the relevant task behavior on this prompt. Conversely, a concise learner answer that gives $(5, 12)$ and $(12, 5)$ with correct verification may satisfy the task even without teacher-like wording. The DeepSeek-specific conclusion should therefore stay at the artifact and reported-result level unless a local run names the learner checkpoint, prompt set, template, decoding settings, metric, and raw outputs.

Section 9.9 tests released learner artifacts and their deployment admission. Those measurements reproduce neither the distillation process nor a learner-to-R1-teacher comparison.

The reported 32B rows do not isolate distillation as the only changed variable: the training data, optimization path, and potentially other recipe coordinates differ between the distilled learner and the direct-RL construction. The table supports the source-reported result that the R1-generated-data learner scored higher under those protocols. It does not establish a universal causal rule that SFT distillation outperforms RL for every learner, task mixture, or compute budget.

9.9. R1 Release Observation and Evaluation

Definition 2.10.2 maps final answers, returned work, tool actions, termination, and usage to their own fields. The same chapter supplies the canonical sampling, aggregation, and cost rules; the Appendix A turns them into harness components and adds population-aware uncertainty controls. This section applies that schema to returned reasoning and final answers, then uses the shared protocol for released R1-Distill learners.

Returned reasoning and final answers. A returned transcript belongs in the observation's work field; the answer intended for the user or scorer belongs in its final-answer field. Unreturned intermediate computation remains

outside the observation. Evaluation, accounting, and disclosure rules can therefore act on different objects without treating a current response field as the underlying concept.

Example 9.9.1 (Trace fields on the running prompt) For the running prompt, a visible transcript might show the derivation $t^2 - 17t + 60 = (t - 5)(t - 12)$, then a final answer might state the two ordered pairs and verification. In an answer-only response schema, the user may see only the pairs and verification. A system may also perform unreturned intermediate computation. These observations answer the same mathematical question but expose different response components.

For an observed route, a dated adapter maps returned reasoning, final answer, tool state, termination, and usage into the normalized schema. Exact hosted field names, controls, aliases, and output limits belong to Appendix C.4; counter nesting and unavailable-field handling belong to the schema adapter in Chapter A. A valid measurement retains the model or route, schema date, returned usage fields, and explicit latency timing while leaving unavailable transcripts and token counts unset.

Returned reasoning, final answer, and usage counters measure different parts of a response; substituting one for another corrupts the evaluation.

The R1 paper sometimes scores the final summary for open-ended generation to avoid length bias [47]. That procedure judges the answer rather than the transcript. An evaluation must name the object that its scorer observes.

R1-family diagnostics. A reported R1 score belongs to its model, item set, prompt, decoding policy, sample count, extractor, scorer, and aggregation rule. The local results below retain those protocol fields and test only released distill learners, outside a reproduction of the R1 teacher evaluation.

The running ordered-pair task accepts a response only when it extracts $\{(5, 12), (12, 5)\}$ and verifies both equations. Increasing the 1.5B learner's cap from 512 to 2,048 tokens changes acceptance from 11/64 to 44/64 while reducing cap hits from 64/64 to 16/64. The shorter run therefore samples a narrower response surface than the longer run.

With the same prompt, 2,048-token cap, temperature, top-p, and seeds, the 7B and 1.5B learners accept 50/64 and 43/64 samples. An exact rerun reproduces those counts, while a new seed block yields 55/64 and 41/64. The descriptive difference is seed-sensitive even on one fixed task.

Table 9.12 consolidates the retained local diagnostics without pooling runs that use different sampling policies.

Another local ladder asks a deployment-admission question: which learner paths reach explicit CUDA generation under the same deterministic 32-token policy? Table 9.13 reports admission status rather than benchmark quality.

All three artifacts reached CUDA generation, but none of their capped outputs contained a parseable final answer. The first 70B attempt stopped before weight loading because `auto-awq` was absent. A follow-up installed AutoAWQ 0.2.9 in an isolated environment, pinned Transformers 4.51.3, and used float16 for the non-quantized tensors; the artifact then fit wholly on `cuda:0` and completed 10/10 generations. The ladder resolves deployment admission only. Because the

Local run	Retained observation	Interpretation
1.5B cap control	11/64 accepted at 512 tokens; 44/64 at 2,048	Output budget and censoring are part of the score.
Matched 7B/1.5B seed blocks	50/64 versus 43/64, reproduced exactly; new seeds give 55/64 versus 41/64	One prompt supports a reproducible artifact control; a broad size ranking requires more tasks.
7B protocol matrix	Mixed harness matches move from 18/32 to 15/32; most rows reach the cap and no strict render or structured tool call passes	Parser, cap, and scorer sensitivity dominate the aggregate.
Jetson Thor terminal-format diagnostic	7B: 0/40 strict but 38/40 correct boxed integers; 32B: 2/10 strict and 10/10 boxed	Correct content and the required response format define different evaluation outcomes.

Table 9.12.: Consolidated R1-family protocol diagnostics. The NVIDIA execution chapter carries detailed campaign provenance; each sampling policy is analyzed on its own.

Artifact and revision	Gate status	Runs	Gen. tok/s	Peak CUDA allocated (GiB)	Limitation
R1-Distill-Qwen-7B, `916b56a44061`	bfloat16 on `cuda:0`	10/10	9.11	14.25	All ten reached the 32-token cap
R1-Distill-Qwen-32B, `711ad2ea6aa4`	bfloat16 on `cuda:0`	10/10	2.17	61.06	All ten reached the 32-token cap
Third-party R1-Distill-Llama-70B-AWQ, `a1ab7653aae7`	float16 on `cuda:0`	10/10	1.61	37.08	All ten reached the 32-token cap

Table 9.13.: Single-device direct-Transformers admission ladder on one Jetson Thor. Generation rate is end-to-end `model.generate` token rate, not serving throughput, time to first token (TTFT), or inter-token latency (ITL); peak allocation is the recorded whole-process CUDA maximum.

70B follow-up changes quantization, tensor dtype, and dependency path, the reported rates are not a controlled learner-size comparison; they also say neither which learner reasons better nor how any learner compares with the R1 teacher.

The R1 report also names function calling, multi-turn use, role play, JSON output, language mixing, prompt sensitivity, few-shot degradation, and the scale of software-engineering RL as separate limitations [47]. Mathematics or reasoning averages leave those surfaces unresolved.

No local row is an R1-teacher-to-learner comparison. The official 671B teacher exceeds the verified four-H200 capacity, so learner-only controls remain diagnostics. The companion configuration `chapter_deepseek_r1/mini_r1_verl_record.example.json` specifies the intended data, reward, rollout, command, and checkpoint fields; measurement remains future work. Deployment therefore needs a bounded learner evaluation: identify the artifact and workload, measure quality and latency thresholds, and state the fallback route.

Deployment admission has three gates. An artifact gate verifies checkpoint, tokenizer, configuration, memory fit, and supported dtype. An execution gate verifies that the rendered request completes under the selected runtime and returns the expected reasoning/final-answer schema. A capability gate measures quality, latency, cost, and policy thresholds on the declared workload. Passing the first two proves loadability and operability, not acceptable reasoning performance.

9.10. R1 Versus R1-Zero Synthesis

R1-Zero and R1 serve different purposes. R1-Zero isolates the scientific result that rule-reward RL can elicit reasoning behavior from a base model. R1 turns that result into a public recipe and artifact family. Cold-start data and supervised refinement shape readable answers; broader alignment, distillation, and release evaluation complete the release account [47].

The running factor-pair example makes the release difference concrete. A R1-Zero-style sample may explore and self-correct; an R1-style release must also expose the answer through a readable, stable schema. The mathematics stayed fixed; the release obligation changed.

The public sources support the stage order, artifact family, comparisons, and dated interface observations. Private prompts, rewards, filters, and hosted implementation remain undisclosed. Stage order must still preserve edge type: the reasoning-RL checkpoint generates later SFT rows, while DeepSeek-V3-Base supplies the mixed-SFT starting weights.

Comparison axis	R1-Zero	R1
Starting point	Base model with pure RL emphasis	Base model first shaped with cold-start supervised data
Supervised data before RL	None in the R1-Zero setup described by the paper	Thousands of cold-start examples reported by the R1 paper
RL role	Demonstrates emergence of reasoning behavior from rule-reward RL	Improves reasoning after readable initialization and again in broader scenarios
Readability	Public sources report poor readability and language mixing as issues	Cold start and later stages target readable, usable behavior
Supervised refinement	Outside the main R1-Zero account	Rejection-sampled reasoning data plus general data fine-tune a DeepSeek-V3-Base restart for two epochs
Release target	Pedagogical and scientific demonstration of pure RL reasoning	Practical reasoning-model release and training recipe
Hosted-interface implication	Outside the central product-facing release	Reasoning-output schemas and token accounting become deployment concerns
Main risk in reading	Treating emergent traces as magic or as complete internal computation	Treating the public recipe as if it exposed every private training detail

Table 9.14.: Public R1-Zero and R1 contrast used for the chapter synthesis.

9.11. Summary

DeepSeek-R1 turns the R1-Zero scientific result into a staged recipe with cold-start supervision, reasoning RL, supervised refinement from accepted trajectories, and broader later RL. The topology is branched: the reasoning-RL checkpoint generates the selected reasoning rows, while a DeepSeek-V3-Base restart receives those rows plus general data for two SFT epochs before final RL. R1-Distill applies SFT on the curated 800k-sample set to dense learners without learner-side RL. The synthesis table in Section 9.10 is the canonical comparison of their roles.

The chapter's local learner rows test output-cap effects, fixed-prompt model-size differences, scorer semantics, and rendering failures. Section 9.9 consolidates those results and their limits. Each is a diagnostic rather than a standard benchmark, an R1-teacher transfer measurement, or a view into private training details.

The companion mini-R1 configuration supplies a rerunnable follow-up surface. Its logs, restart behavior, and evaluation protocol remain part of any result, while Scope 9.3.1 and Scope 9.7.1 distinguish public recipe statements from third-party execution. The verification ladder continues in Chapter 10.

9.12. Exercises

These exercises test the R1 stage sequence, post-cold-start reward boundary, execution handoffs, distillation artifacts, and release evidence.

1. Applied: Define cold-start data formally and explain, using the running example, why it can improve readability without removing the later reinforcement learning contribution.
2. Applied: For four sampled answers to the running example, define separate records for R1's reported accuracy and language-consistency reward terms, then define local format and length acceptance gates. Explain why an acceptance gate should not be reported as a reward term.
3. Research: Capstone lab: using `chapter_deepseek_r1/mini_r1_prompts.jsonl`, `chapter_deepseek_r1/mini_r1_reward.py`, `chapter_deepseek_r1/mini_r1_reward_checkpoint.py`, and `chapter_deepseek_r1/mini_r1_verl_record.example.json`, draw and explain the policy-and-data topology: mark the cold-start and reasoning-RL checkpoint edges, the $\pi_{\text{rrl}} \to D_{\text{reason}}$ data edge, and the $\pi_0 \to \pi_{\text{sft2}}$ mixed-SFT initialization edge. Then validate the three printed reward outcomes, prepare the data paths, and run or dry-run a one-epoch veRL GRPO job under the pinned framework revision. Inspect reward distributions and parse-failure counts, evaluate the checkpoint on the validation row, resume from the saved checkpoint, and summarize the validation and resume behavior from the logged outputs.
4. Core: Explain why a HybridFlow-style logical actor uses a generation layout for rollouts and a training layout for policy updates. Name what crosses the handoff in each direction, and state why this framework example does not establish DeepSeek's private R1 infrastructure.
5. Core: Explain rejection sampling as a supervised-data construction method. Then distinguish mixed SFT from the final all-scenario RL stage, and give one way the transition can introduce selection bias, benchmark contamination risk, or a too-narrow evaluation surface.
6. Core: Compare visible reasoning, hidden reasoning, and final-answer-only output. For each, state what the client can log, what the scorer can evaluate, and which internal policy state remains hidden.
7. Research: Build a benchmark observation record for one R1 score reported in the R1 paper. Include model identifier, benchmark, metric, prompt or harness, decoding settings, sample count, source, and limitation.
8. Research: Choose two DeepSeek-R1-Distill checkpoints with different learner base models or parameter sizes. Compare their teacher, learner family, parameter count, expected memory footprint, and likely throughput tradeoff under the same decoding length.
9. Applied: Fill in an R1 versus R1-Zero comparison table with at least six rows: starting point, supervised data, reinforcement-learning role, readability, failure modes, and release target.
10. Challenge: Compare two reasoning-service rows when one exposes a reasoning-token count and the other returns only total and final tokens. Specify which cost-normalized claims remain valid, which become unidentifiable, and what request-schema evidence would close the gap.