

# **Advanced Post-Training Methodologies for Reasoning-Oriented Large Language Models: A Comprehensive Analysis of Algorithmic Architectures and Hyperparameter Optimization Strategies for the Google Tunix Hackathon**

The evolution of generative artificial intelligence has transitioned from basic token prediction toward the elicitation of complex reasoning capabilities, specifically through the incentivization of internal "thinking" traces. For practitioners engaged in the Google Tunix Hackathon, the objective is not merely to produce a correct final answer but to engineer a model that demonstrates a coherent, logical progression of thought within a constrained computational environment. This challenge necessitates a sophisticated synthesis of JAX-native post-training techniques, particularly those centered around reinforcement learning from verifiable and structured rewards. The technical landscape is currently defined by a shift from purely supervised paradigms to multi-stage reinforcement learning (RL) frameworks that prioritize emergent behaviors like self-correction, reflection, and multi-step verification. The following analysis dissects the architectural breakthroughs of seminal research papers, focusing on their training data compositions, hyperparameter sensitivities, and strategic implementations, while providing actionable insights for optimizing Gemma-based models on TPU infrastructure.

## **DeepSeek-R1: Architecting Emergent Reasoning Through Multi-Stage Reinforcement Learning**

The DeepSeek-R1 series represents a paradigm shift in the development of reasoning models by demonstrating that large-scale reinforcement learning can independently unlock advanced Chain-of-Thought (CoT) capabilities. The research distinguishes between DeepSeek-R1-Zero, which relies on pure RL from a base model, and DeepSeek-R1, which utilizes a sophisticated multi-stage pipeline to address the limitations of the former.<sup>1</sup>

### **DeepSeek-R1-Zero: The Emergence of Pure Reasoning**

The fundamental contribution of DeepSeek-R1-Zero is the proof that a base model, when

subjected to Group Relative Policy Optimization (GRPO) without any prior supervised fine-tuning (SFT), can autonomously develop reasoning strategies. The model utilizes a reward system based exclusively on rule-based verifiers for mathematics and programming.<sup>3</sup> This approach avoids the high cost of manual annotation and prevents the model from being biased by human-written reasoning traces that might not be optimal for the model's internal latent space.

The primary strategy involves maximizing the expected reward where the verifier assigns a binary score based on the final answer's correctness. During training, researchers observed the spontaneous emergence of reflection and self-verification, behaviors that were not explicitly programmed but arose as optimal strategies for maximizing accuracy.<sup>1</sup> However, this "Zero-SFT" variant suffered from "language mixing" and repetitive outputs, indicating that while logic can be elicited through pure RL, the presentation and readability of that logic require human-centric grounding.<sup>2</sup>

### DeepSeek-R1: Stabilization via Cold Start and Multi-Stage RL

To rectify the readability and instability issues of the Zero variant, DeepSeek-R1 introduces a "Cold Start" phase. This phase utilizes a curated dataset of several thousand long CoT reasoning samples to fine-tune the base model before the RL phase begins.<sup>2</sup> The data curation strategy for this cold start is critical; it includes few-shot prompting with detailed examples, direct prompting for reflection, and the post-processing of DeepSeek-R1-Zero outputs by humans to ensure readability and formatting standards.<sup>2</sup>

The multi-stage pipeline follows a specific progression:

1. **Cold Start:** SFT on a small, high-quality reasoning dataset (~thousands of samples) to establish a readable pattern.<sup>2</sup>
2. **Reasoning-Oriented RL:** Applying large-scale GRPO to enhance logical capabilities in math and coding.<sup>2</sup>
3. **Rejection Sampling and SFT:** Near the point of RL convergence, the model generates its own labeled data (synthetic data). Top-tier examples are selected via rejection sampling and combined with general-domain SFT data (writing, factual QA) for a second round of fine-tuning.<sup>8</sup>
4. **Final RL Stage:** A final alignment phase using RL across diverse prompts to ensure the model's reasoning generalizes to non-verifiable tasks and aligns with human preferences.<sup>5</sup>

Hyperparameter Category	Configuration	Impact on Results
-------------------------	---------------	-------------------

<b>Group Size (\$G\$)</b>	16	Stabilizes advantage estimation by providing a relative baseline without the memory overhead of a critic model. <sup>5</sup>
<b>KL Penalty (\$\beta\$)</b>	0.04	Regularizes the policy to prevent it from drifting too far from the reference model, preserving general linguistic capabilities. <sup>6</sup>
<b>Learning Rate</b>	$1 \times 10^{-5}$	Ensures stable convergence during the high-variance RL phase; higher rates often lead to catastrophic forgetting of base knowledge. <sup>10</sup>
<b>Temperature</b>	0.6	Balances exploration with coherence; 0.6 is recommended to avoid the incoherent outputs seen at higher temperatures. <sup>12</sup>
<b>Adam Epsilon</b>	$1 \times 10^{-8}$	Standard setting, though increasing this can mitigate numerical instability in JAX-based TPUs. <sup>11</sup>

The hyperparameter selection in GRPO is particularly vital for the hackathon's TPU environment. By using a group size of 16, the model can compute the mean and standard deviation of rewards within a batch, effectively normalizing the advantage signal.<sup>1</sup> This normalization mitigates the noise inherent in absolute reward scales and eliminates the requirement for a separate value-function model, which is essential for fitting the training pipeline into a single Kaggle TPU session.<sup>14</sup>

## Prompting Strategy and Formatting

The DeepSeek-R1 research emphasizes "minimal prompting" for reasoning models. Unlike traditional LLMs that benefit from elaborate system prompts, reasoning models already possess internal structures for breaking down problems. Overloading the model with context or "step-by-step" instructions can actually degrade performance as it interferes with the

model's autonomous reasoning process.<sup>12</sup>

The required format for the hackathon mirrors the R1 approach: <reasoning>...</reasoning> followed by <answer>...</answer>. To enforce this, practitioners should use a direct prompt template that clearly defines these XML-like boundaries. In some instances, "pre-filling" the assistant response with the opening <reasoning> tag is used to ensure the model engages the thinking pattern rather than bypassing it.<sup>12</sup>

## Rubrics as Rewards (RaR): Quantifying Quality in Open-Ended Reasoning

A critical bottleneck in expanding reasoning models is the lack of verifiable rewards for general domains like creative writing or medical diagnostics. The "Rubrics as Rewards" (RaR) framework provides a methodology for extending reinforcement learning beyond verifiable domains by utilizing structured, multi-criteria checklists.<sup>18</sup>

### Rubric Construction and Expert Grounding

The RaR approach involves generating prompt-specific rubrics that decompose "what makes a good response" into measurable subgoals.<sup>20</sup> These rubrics are synthesized using a strong teacher model (e.g., GPT-4o) and grounded in expert guidance, such as a human-written reference answer.<sup>20</sup> A typical rubric consists of 7 to 20 items categorized by importance: "Essential" (facts/safety), "Important" (logic/completeness), "Optional" (style), and "Pitfalls" (common errors).<sup>23</sup>

By treating rubrics as instance-specific reward functions, researchers achieved a 31% relative improvement on the HealthBench (medical) dataset and a 7% improvement on GPQA-Diamond (science) over traditional Likert-based rewards.<sup>19</sup> The insight here is that structured rubrics provide a much more stable and granular signal than a single scalar score, which is often prone to "reward hacking" based on superficial qualities like verbosity or persuasion.<sup>22</sup>

### Aggregation Mechanics: Explicit vs. Implicit

The study evaluates two primary strategies for aggregating rubric feedback:

1. **Explicit Aggregation:** An LLM judge evaluates each criterion independently. The final reward is a weighted sum normalized to ensure comparability across prompts with different rubric counts.<sup>21</sup>
$$r(x, y) = \frac{\sum_{j=1}^k w_j \cdot c_j(x, y)}{\sum_{j=1}^k w_j}$$
2. **Implicit Aggregation:** The entire set of criteria and weights is passed to the LLM judge, which then produces a holistic scalar reward. Implicit aggregation consistently outperforms explicit methods because it allows the judge model to capture the nuanced

interdependencies between logic and factual accuracy that a rigid formula might miss.<sup>22</sup>

Reward Strategy	Advantage	Disadvantage
Verifiable (RLVR)	High precision, impossible to hack with logic. <sup>27</sup>	Limited to math/code. <sup>28</sup>
Explicit Rubric	High transparency and auditability. <sup>24</sup>	Can be brittle; ignores holistic flow. <sup>22</sup>
Implicit Rubric	Best performance; adapts to complexity. <sup>26</sup>	Less interpretable than checklists. <sup>24</sup>
Likert Score	Simple to implement. <sup>19</sup>	Prone to verbosity bias. <sup>22</sup>

For hackathon participants, the RaR framework suggests that the reward function should not be a single script but a "judge prompt" that presents the Gemma model's output alongside a reference answer and a list of specific criteria to be checked. This is particularly relevant for the held-out evaluation queries that may cover non-verifiable tasks like creative ideation or summarization.<sup>31</sup>

## Intrinsic Probability Rewards: RLPR and Verifier-Free Scaling

In the absence of both rule-based verifiers and expensive LLM-as-a-judge models, the "Reinforcement Learning with Reference Probability Reward" (RLPR) framework offers a scalable, verifier-free alternative. This technique relies on the model's own internal token probabilities as the primary reward signal.<sup>32</sup>

### The Probability Reward Mechanism

RLPR is based on the observation that a model's intrinsic probability of generating a correct answer directly indicates its internal evaluation of the preceding reasoning process.<sup>32</sup> When the policy model generates a reasoning trace  $z$  and an answer  $y$ , the system replaces  $y$  with the ground-truth reference  $y^*$  and computes the mean token probability of  $y^*$  given the prompt  $x$  and reasoning  $z$ .<sup>33</sup>

$$f_{\text{seq}} = \frac{1}{|y^*|} \sum_{o_i \in y^*} p_i$$

Using the mean probability instead of the product (sequence likelihood) is a critical hyperparameter choice. Sequence likelihood is highly sensitive to single low-probability

tokens or synonyms, which can cause the reward signal to collapse. Mean probability provides a more robust and stable signal that correlates better with human judgment.<sup>33</sup>

## Reward Debiasing and Adaptive Curriculum

To ensure the model is rewarded only for its *reasoning* and not for its prior knowledge of the answer, RLPR implements "Reward Debiasing." A base score  $r'$  is calculated by measuring the probability of the reference answer given the prompt *without* any reasoning. The debiased reward is then  $\hat{r} = \text{clip}(0, 1, r - r')$ . This effectively isolate the "value-add" of the thinking process.<sup>33</sup>

Furthermore, to stabilize training, RLPR employs an "Adaptive Curriculum" via Standard Deviation (SD) filtering. Prompts that yield very low reward variance across multiple rollouts are filtered out. If the rewards are all the same, the prompt is either too easy or too complex to provide useful gradients. The SD threshold is updated dynamically via an exponential moving average.<sup>32</sup>

RLPR Component	Performance Gain (Average)	Rationale
Mean Probability vs Likelihood	+22 points	Prevents reward collapse due to single-token variance. <sup>34</sup>
Reward Debiasing	+2.5 points	Removes bias from intrinsic model knowledge. <sup>34</sup>
SD Filtering	+2.2 points	Focuses compute on informative training samples. <sup>32</sup>

This verifier-free strategy achieved a 24.9% average improvement on general-domain benchmarks and consistently outperformed methods using trained verifier models.<sup>34</sup> For the Tunix hackathon, this implies that even when a rule-based verifier is available (as in math), augmenting it with probability-based signals can provide a smoother, more fine-grained reward that accelerates convergence.<sup>33</sup>

## Bridging Offline and Online Reinforcement Learning: The Synchronization Factor

The transition from offline paradigms (like standard DPO) to fully online RL (like GRPO) is

governed by the frequency of synchronization between the generating policy and the training policy. Research into "Bridging Offline and Online RL" identifies the synchronization parameter  $\gamma$  as a central lever for balancing compute efficiency and model performance.<sup>13</sup>

### The Sync Parameter ( $\gamma$ ) and Efficiency

Offline RL ( $\gamma = \infty$ ) involves training on pre-collected, static data. While computationally cheap, it significantly lags behind other methods because the model is not learning from its own evolving outputs.<sup>13</sup> Fully Online RL ( $\gamma = 1$ ) synchronizes weights at every step, providing the most accurate signal but creating a massive computational bottleneck.<sup>37</sup>

Semi-Online RL ( $\gamma \in [0, 1]$ ) provides a "middle ground." Weights are synchronized periodically (e.g., every 10 steps). This approach captures nearly all the performance gains of online RL while allowing for massively parallel generation of responses between sync intervals.<sup>13</sup> For the 9-hour TPU sessions in the hackathon, a semi-online synchronization strategy could allow for higher throughput, enabling the model to see more prompts per session.<sup>13</sup>

### Multi-Tasking Rewards

The study also reveals that multi-tasking with verifiable (math) and non-verifiable (chat) rewards simultaneously improves performance across both task types.<sup>37</sup> This "cross-pollination" suggests that logic learned in mathematical reasoning generalizes to better instruction-following. Consequently, a winning strategy should involve a training dataset that mixes math problems with general reasoning tasks, using a hybrid reward function that switches between rule-based and rubric-based evaluation.<sup>13</sup>

Training Regime	Benchmark Performance (Math)	Compute Cost
Offline DPO	Baseline	Low. <sup>13</sup>
Semi-Online DPO ( $\gamma=10$ )	+15% over Offline	Moderate. <sup>13</sup>
Online GRPO ( $\gamma=1$ )	+17% over Offline	High. <sup>13</sup>
Multi-Task (Mixed)	+20% over Single-Task	Moderate-High. <sup>37</sup>

These findings underscore the importance of reference model synchronization. Disabling it completely for verifiable tasks often leads to "response length collapse," where the model learns to provide the answer without the reasoning to maximize speed/efficiency, resulting in a

failure to meet the hackathon's "show your work" requirement.<sup>13</sup>

## Tunix Implementation: JAX, TPUs, and Gemma3-Specific Techniques

The Google Tunix library is a JAX-native environment specifically designed for post-training LLMs on TPUs. Its "white-box" design allows for low-level control of the training loop, which is critical for implementing the advanced strategies discussed above.<sup>14</sup>

### Parameter-Efficient Fine-Tuning (PEFT) with LoRA

In the context of the hackathon's 20-hour weekly TPU limit, full-weight fine-tuning is often impractical for the Gemma 2B or 3 1B models. Tunix supports Parameter-Efficient Fine-Tuning (PEFT) through LoRA (Low-Rank Adaptation) and QLoRA (Quantized LoRA).<sup>14</sup>

Hyperparameter	Value	Rationale
LoRA Rank ( $r$ )	16	Balancing model capacity with VRAM usage; higher ranks provide more flexibility but increase memory footprint. <sup>15</sup>
LoRA Alpha	32	Sets the scale of the adapter weights; generally set to $2 \times r$ . <sup>15</sup>
Learning Rate	$2 \times 10^{-5}$	Optimized for LoRA adapters; higher than full-tuning rates to ensure adapters learn effectively. <sup>11</sup>
Weight Decay	0.01	Prevents overfitting of the small number of adapter parameters. <sup>11</sup>
Warmup Steps	50-100	Essential for JAX stability as the TPU mesh initializes. <sup>11</sup>

Using LoRA allows for sharding the parameters across the TPU mesh using axis names like

('fsdp', 'tp'). This FSDP (Fully Sharded Data Parallel) sharding is critical for Gemma3-1B, ensuring that the model and its optimizer states can fit into the 16GB of HBM typical of a TPU v5e-8.<sup>11</sup>

## JAX Mesh and Sharding Configurations

Efficient sharding is the "secret sauce" for high-performance JAX training. The mesh shape (8, 1) is a common baseline for a TPU v5e-8, where the 8 chips are used for data parallelism (FSDP) while maintaining a single tensor-parallel axis.<sup>11</sup>

1. **Extract Model State:** Use `nnx.state` to get the model parameters.
2. **Partition Specs:** Compute `nnx.get_partition_spec` to define how each layer should be sharded across the mesh.
3. **Sharding Constraints:** Apply `jax.lax.with_sharding_constraint` to ensure that data transfer between chips is minimized during the forward and backward passes.<sup>46</sup>

This configuration directly impacts the "Model Quality" score (45 points) in the hackathon, as a well-sharded model can process more tokens per second, allowing for longer reasoning traces within the 9-hour limit.<sup>31</sup>

## Custom Reward Function Composition in Tunix

The `reward_fn` in the Tunix Trainer class is the primary interface for implementing reasoning logic. A winning reward function should be a composition of several sub-rewards:

- **XML Matcher:** Uses regular expressions to ensure the output contains exactly one `<reasoning>` and one `<answer>` block. This addresses the "Format Accuracy" metric.<sup>15</sup>
- **Logical Coherence Judge:** An internal LLM-as-a-judge score (using a smaller, distilled model if necessary) that penalizes logical inconsistencies in the thinking trace.<sup>21</sup>
- **Correctness Verifier:** Compares the extracted answer to the ground truth. This is the "Answer Accuracy" metric.<sup>15</sup>
- **Numeric Tolerance:** For math problems, giving partial credit for answers within 10% of the target can prevent the RL from being too binary and provide a smoother learning curve.<sup>14</sup>

## Practical Synthesis for Hackathon Success

Winning the Google Tunix Hackathon requires more than just running the starter notebook; it requires a data-centric and reward-centric approach grounded in the latest research.

### Data Curation: Quality Over Quantity

The 9-hour session limit suggests that participants should focus on high-quality, "reasoning-dense" datasets. Instead of using the full GSM8K, one should curate a subset of 2,000–5,000 samples that demonstrate complex, multi-step derivation.<sup>47</sup> This "Cold Start"

approach (Stage 1 of DeepSeek-R1) provides the foundation for formatting and logic that the RL stage then amplifies.<sup>2</sup>

## Reward Engineering: Balancing Objective and Subjective

The evaluation judging period uses a combination of LLM-as-a-judge and human judgment across math, coding, and creative tasks.<sup>31</sup> Therefore, the reward function must be multi-modal. Using "Rubrics as Rewards" for creative tasks ensures the model's reasoning trace is not just long, but actually relevant to the final answer.<sup>19</sup> Integrating "Intrinsic Probability Rewards" (RLPR) can provide a continuous feedback signal that helps the model converge faster on correct reasoning paths.<sup>33</sup>

## Training Tactics for TPU Sessions

- **Multi-Session Strategy:** While one session is the baseline, the optional 15 points for multi-session training can be secured by saving intermediate checkpoints to a Kaggle Model and loading them in a subsequent session. This allows for the "rejection sampling" phase of the R1 pipeline, where the model generates its own training data for the next stage.<sup>8</sup>
- **Optimizer Tuning:** Stability is key in JAX. Using a slightly larger Adam epsilon ( $\$1 \times 10^{-6}$ ) and a conservative learning rate ( $\$2 \times 10^{-5}$ ) can prevent the training from collapsing in the final hours of the session.<sup>11</sup>
- **Response Length Management:** To prevent "length bias" or "response length collapse," monitor the token count of the <reasoning> block. A bonus/penalty system can be used to keep reasoning traces within a productive range (e.g., 200–800 tokens), as overly long responses waste compute and overly short ones fail to show work.<sup>13</sup>

By weaving together these algorithmic strategies—DeepSeek's multi-stage RL, RaR's structured rubrics, and RLPR's probability rewards—within the high-performance Tunix library, practitioners can engineer a Gemma-based model that truly "thinks" before it "talks." This approach directly addresses the hackathon's criteria for notebook quality, model quality, and innovative reward function design, positioning the participant at the forefront of the open-source reasoning community.<sup>31</sup>

## Works cited

1. The DeepSeek Series: A Technical Overview - Martin Fowler, accessed December 19, 2025, <https://martinfowler.com/articles/deepseek-papers.html>
2. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, accessed December 19, 2025, <https://arxiv.org/html/2501.12948v1>
3. (PDF) Technical Report: Analyzing DeepSeek-R1's Impact on AI Development, accessed December 19, 2025, [https://www.researchgate.net/publication/388484582\\_Technical\\_Report\\_Analyzing\\_DeepSeek-R1's\\_Impact\\_on\\_AI\\_Development](https://www.researchgate.net/publication/388484582_Technical_Report_Analyzing_DeepSeek-R1's_Impact_on_AI_Development)

4. Aman's AI Journal • Primers • DeepSeek-R1, accessed December 19, 2025, <https://aman.ai/primers/ai/deepseek-R1/>
5. How was DeepSeek-R1 built; For dummies : r/LLMDevs - Reddit, accessed December 19, 2025, [https://www.reddit.com/r/LLMDevs/comments/1ibhpgw/how\\_was\\_deepseekr1\\_built\\_for\\_dummies/](https://www.reddit.com/r/LLMDevs/comments/1ibhpgw/how_was_deepseekr1_built_for_dummies/)
6. Enhancing Reasoning in LLMs with DeepSeek-R1: A Technical blogpost Reinforcement Learning and Distillation | by Gal Hyams | Medium, accessed December 19, 2025, <https://medium.com/@galhyams/enhancing-reasoning-in-llms-with-deepseek-r1-a-technical-blogpost-reinforcement-learning-and-d76a61ffbd01>
7. Key Concepts of DeepSeek-R1 | Niklas Heidloff, accessed December 19, 2025, <https://heidloff.net/article/deepseek-r1/>
8. Pipeline for Training DeepSeek-R1 | by DhanushKumar - Medium, accessed December 19, 2025, <https://medium.com/@danushidk507/pipeline-for-training-deepseek-r1-0d99933c3b4f>
9. Breaking down the DeepSeek-R1 training process—no PhD required - Vellum AI, accessed December 19, 2025, <https://www.vellum.ai/blog/the-training-of-deepseek-r1-and-ways-to-use-it>
10. Google Tux Hack Starter — Gemma + Tux 6-Cell B - Kaggle, accessed December 19, 2025, <https://www.kaggle.com/code/tvmilitary/google-tux-hack-starter-gemma-tux-6-cell-b>
11. google-tux-hackathon - Kaggle, accessed December 19, 2025, <https://www.kaggle.com/code/abhijeetbhargava/google-tux-hackathon/notebook>
12. Prompting DeepSeek R1 - Together.ai Docs, accessed December 19, 2025, <https://docs.together.ai/docs/prompting-deepseek-r1>
13. [Literature Review] Bridging Offline and Online Reinforcement Learning for LLMs, accessed December 19, 2025, <https://www.themoonlight.io/en/review/bridging-offline-and-online-reinforcement-learning-for-llms>
14. Introducing Tux: A JAX-Native Library for LLM Post-Training - Google for Developers Blog, accessed December 19, 2025, <https://developers.googleblog.com/introducing-tux-a-jax-native-library-for-llm-post-training/>
15. Training Gemma to Think - Kaggle, accessed December 19, 2025, <https://www.kaggle.com/code/oscaryezfeijo/training-gemma-to-think>
16. How to Prompt Thinking Models like DeepSeek R1 and OpenAI o3 - Helicone, accessed December 19, 2025, <https://www.helicone.ai/blog/prompt-thinking-models>
17. deepseek-ai/DeepSeek-R1-Distill-Llama-70B ·
18. Daily Papers - Hugging Face, accessed December 19, 2025, <https://huggingface.co/papers?q=Reinforcement%20Learning%20with%20Evolvi>

[ng%20Rubrics%20\(RLER\)](#)

19. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains - OpenReview, accessed December 19, 2025, <https://openreview.net/forum?id=c1bTcrDmt4>
20. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains - arXiv, accessed December 19, 2025, <https://arxiv.org/html/2507.17746v2>
21. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains - arXiv, accessed December 19, 2025, <https://arxiv.org/pdf/2507.17746>
22. RUBRICS AS REWARDS: REINFORCEMENT LEARNING BEYOND VERIFIABLE DOMAINS - OpenReview, accessed December 19, 2025, <https://openreview.net/pdf?id=c1bTcrDmt4>
23. Rubrics - Ian's Blog, accessed December 19, 2025, <https://ianbarber.blog/2025/08/12/rubrics/>
24. Calibrating Scores of LLM-as-a-Judge - GoDaddy Blog, accessed December 19, 2025, <https://www.godaddy.com/resources/news/calibrating-scores-of-llm-as-a-judge>
25. Enterprise Reinforcement Learning with Rubrics as Rewards - Scale AI, accessed December 19, 2025, <https://scale.com/blog/enterprise-rar>
26. Using Rubrics to Build Better Models - Scale AI, accessed December 19, 2025, <https://scale.com/blog/rubrics-as-rewards>
27. Daily Papers - Hugging Face, accessed December 19, 2025, <https://huggingface.co/papers?q=verifiable%20rewards>
28. EXTENDING RLVR TO OPEN-ENDED TASKS VIA VERIFIABLE MULTIPLE-CHOICE REFORMULATION - OpenReview, accessed December 19, 2025, <https://openreview.net/pdf?id=uZxyvmN72d>
29. RLPIR: REINFORCEMENT LEARNING WITH PREFIX AND INTRINSIC REWARD - OpenReview, accessed December 19, 2025, <https://openreview.net/pdf/d47108c9998ee830c91ce4a66021cb28ffce203e.pdf>
30. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains - arXiv, accessed December 19, 2025, <https://arxiv.org/html/2507.17746v1>
31. Google Tunix Hack - Train a model to show its work | Kaggle, accessed December 19, 2025, <https://www.kaggle.com/competitions/google-tunix-hackathon>
32. RLPR: Extrapolating RLVR to general domains without verifiers - OpenReview, accessed December 19, 2025, <https://openreview.net/pdf?id=3nR7vqd7jw>
33. [Literature Review] RLPR: Extrapolating RLVR to General Domains ..., accessed December 19, 2025, <https://www.themoonlight.io/en/review/rlpr-extrapolating-rlvr-to-general-domains-without-verifiers>
34. RLPR: Extrapolating RLVR to General Domains without Verifiers | alphaXiv, accessed December 19, 2025, <https://www.alphaxiv.org/overview/2506.18254v1>
35. RLPR: Extrapolating RLVR to general domains without verifiers - arXiv, accessed December 19, 2025, <https://arxiv.org/html/2506.18254v1>
36. Papers Explained 413: Reinforcement Learning with Reference Probability Reward (RLPR), accessed December 19, 2025, <https://ritvik19.medium.com/papers-explained-413-reinforcement-learning-with-r>

- [eference-probability-reward-rlpr-ac742c006a22](#)
37. Bridging Offline and Online Reinforcement Learning for LLMs - arXiv, accessed December 19, 2025, <https://arxiv.org/pdf/2506.21495>
  38. Bridging Offline and Online Reinforcement Learning for LLMs - OpenReview, accessed December 19, 2025, <https://openreview.net/forum?id=JrWpxBqqFk>
  39. Bridging Offline and Online Reinforcement Learning for LLMs | AI Research Paper Details, accessed December 19, 2025, <https://www.aimodels.fyi/papers/arxiv/bridging-offline-online-reinforcement-learning-llms>
  40. Google Tunix Hack - Train a model to show its work | Kaggle, accessed December 19, 2025, <https://www.kaggle.com/competitions/google-tunix-hackathon/discussion/618645>
  41. Paper page - Bridging Offline and Online Reinforcement Learning for LLMs - Hugging Face, accessed December 19, 2025, <https://huggingface.co/papers/2506.21495>
  42. google/tunix: A Lightweight LLM Post-Training Library - GitHub, accessed December 19, 2025, <https://github.com/google/tunix>
  43. Google Introduces Tunix: An Open-Source JAX Library for LLM Post-Training - C# Corner, accessed December 19, 2025, <https://www.c-sharpcorner.com/news/google-introduces-tunix-an-opensource-jax-library-for-llm-posttraining>
  44. Gemma model fine-tuning | Google AI for Developers, accessed December 19, 2025, <https://ai.google.dev/gemma/docs/tune>
  45. Fine-tuning Google Gemma with Unsloth - Analytics Vidhya, accessed December 19, 2025, <https://www.analyticsvidhya.com/blog/2024/04/fine-tuning-google-gemma-with-unsloth/>
  46. Guide : Gemma 3 fine tuning with Tunix - Kaggle, accessed December 19, 2025, <https://www.kaggle.com/code/yekahaaagayeham/guide-gemma-3-fine-tuning-with-tunix>
  47. Fine-tuning DeepSeek R1 on a Custom Instructions Dataset - Firecrawl, accessed December 19, 2025, <https://www.firecrawl.dev/blog/fine-tuning-deepseek>
  48. Let's break down the Google Tunix Hack — thoughts & strategy | Kaggle, accessed December 19, 2025, <https://www.kaggle.com/discussions/general/617788>
  49. DSA-CAST-TUNIX-noLoRA-from-scratch - Kaggle, accessed December 19, 2025, <https://www.kaggle.com/code/danielwycoff/dsa-cast-tunix-nolora-from-scratch>

