prompts. Few-shot prompting consistently degrades its performance. Therefore, we recommend users directly describe the problem and specify the output format using a zero-shot setting for optimal results.

Software Engineering Tasks: Due to the long evaluation times, which impact the efficiency of the RL process, large-scale RL has not been applied extensively in software engineering tasks. As a result, DeepSeek-R1 has not demonstrated a huge improvement over DeepSeek-V3 on software engineering benchmarks. Future versions will address this by implementing rejection sampling on software engineering data or incorporating asynchronous evaluations during the RL process to improve efficiency.

Beyond specific capability limitations, the pure RL methodology itself also presents inherent challenges:

Reward Hacking: The success of pure RL depends on reliable reward signals. In this study, we ensure reward reliability through a reasoning-domain rule-based reward model (RM). However, such dependable RMs are difficult to construct for certain tasks, such as writing. If the reward signal is assigned by a model instead of predefined rules, it becomes more susceptible to exploitation as training progresses, which means the policy model may find shortcuts to hack the reward model. Consequently, for complex tasks that cannot be effectively evaluated by a reliable reward model, scaling up pure RL methods remains an open challenge.

In this work, for tasks that cannot obtain a reliable signal, DeepSeek-R1 uses human annotation to create supervised data, and only conduct RL for hundreds of steps. We hope in the future, a robust reward model can be obtained to address such issues.

With the advent of pure RL methods like DeepSeek-R1, the future holds immense potential for solving any task that can be effectively evaluated by a verifier, regardless of its complexity for humans. Machines equipped with such advanced RL techniques are poised to surpass human capabilities in these domains, driven by their ability to optimize performance iteratively through trial and error. However, challenges remain for tasks where constructing a reliable reward model is inherently difficult. In such cases, the lack of a robust feedback mechanism may hinder progress, suggesting that future research should focus on developing innovative approaches to define and refine reward structures for these complex, less verifiable problems.

Furthermore, leveraging tools during the reasoning process holds significant promise. Whether it's utilizing tools like compilers or search engines to retrieve or compute necessary information, or employing external tools—such as biological or chemical reagents, to validate final results in the real world, this integration of tool-augmented reasoning could dramatically enhance the scope and accuracy of machine-driven solutions.

7. Author List

The list of authors is organized by contribution role, with individuals listed alphabetically by their first name within each category. Authors marked with an asterisk (*) are no longer affiliated with our team.

Core Contributors: Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z.F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao,

Contributions of the Core Authors: Peiyi Wang and Daya Guo jointly demonstrated that outcome-based RL induces the emergence of long Chain-of-Thought patterns in LLMs, achieving

breakthrough reasoning capabilities. They contributed equally to the creation of R1-Zero, and their work laid the foundation for R1. Daya Guo also contributed to the RL training stability of MOE models. Junxiao Song proposed the GRPO algorithm, implemented the initial version, and introduced rule-based rewards for math tasks. The GRPO algorithm was subsequently refined by Peiyi Wang and Runxin Xu. Zhibin Gou proposed a large PPO clipping strategy to enhance GRPO performance, demonstrating its significance alongside Zhihong Shao and Junxiao Song. Regarding data iteration, reward design, and evaluation, specific teams led efforts across different domains: Qihao Zhu, Z.F. Wu, and Dejian Yang focused on code tasks; Zhihong Shao, Zhibin Gou, and Junxiao Song focused on math tasks; and Peiyi Wang, Ruoyu Zhang, Runxin Xu, and Yu Wu led efforts for other reasoning and general tasks. Additionally, Qihao Zhu and Zhihong Shao contributed to the data selection strategy for RL training, while Zhuoshu Li and Yu Wu co-led the data labeling efforts for the entire project. On the system side, Xiao Bi, Xingkai Yu, Shirong Ma, Xiaokang Zhang, Haowei Zhang, and Ziyi Gao implemented the RL pipeline, optimizing system efficiency and addressing stability issues in large-scale training. Finally, Zhibin Gou, Daya Guo, and Ruoyu Zhang oversaw the final training phase and monitored the model training dynamics. Zhibin Gou led the development of the R1-distill series.

Contributors: Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo*, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J.L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu*, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R.J. Chen, R.L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S.S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu*, Wentao Zhang, W.L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X.Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y.K. Li, Y.Q. Wang, Y.X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma*, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y.X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z.Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu*, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Zhen Zhang,

Appendix

A. Background

A.1. DeepSeek-V3

DeepSeek V3 (DeepSeek-AI, 2024b) is an advanced open-source LLM developed by DeepSeek. Released in December 2024, DeepSeek V3 represents a significant leap forward in AI innovation, designed to rival leading models like OpenAI's GPT-4 and Meta's Llama 3.1, while maintaining remarkable cost efficiency and performance. Built on a Mixture-of-Experts (MoE) architecture, DeepSeek V3 has 671 billion total parameters, with 37 billion activated per token, optimizing both efficiency and capability. It was pre-trained on an expansive dataset of 14.8 trillion high-quality, diverse tokens, followed by supervised fine-tuning and reinforcement learning to enhance its abilities across various domains. The model incorporates innovative features like Multi-head Latent Attention (MLA) (DeepSeek-AI, 2024a) for efficient inference, an auxiliary-loss-free load-balancing strategy, and Multi-Token Prediction (MTP) (Gloeckle et al., 2024) to boost performance, particularly in tasks like mathematics and coding.

For the training data of DeepSeek-V3-Base, we exclusively use plain web pages and e-books, without incorporating any synthetic data. However, we have observed that some web pages contain a significant number of OpenAI-model-generated answers, which may lead the base model to acquire knowledge from other powerful models indirectly. However, we did not intentionally include synthetic data generated by OpenAI during the pre-training cooldown phase; all data used in this phase were naturally occurring and collected through web crawling. The pre-training dataset contains a substantial amount of mathematical and code-related content, indicating that DeepSeek-V3-Base has been exposed to a significant volume of reasoning trace data. This extensive exposure equips the model with the capability to generate plausible solution candidates, from which reinforcement learning can effectively identify and optimize high-quality outputs. We did the data contamination in pre-training as described in Appendix D.1. The training data of DeepSeek-V3 base are mostly Chinese and English, which might be the cause for DeepSeek-R1-Zero language mixing when the language consistent reward is absent.

In this paper, we use the notation DeepSeek-V3-Base as the base model, DeepSeek-V3 as the instructed model. Notably, DeepSeek-R1 and DeepSeek-R1-Zero are trained on top of DeepSeek-V3-Base and DeepSeek-R1 leverages non-reasoning data from DeepSeek-V3 SFT data. DeepSeek-R1-Dev1, DeepSeek-R1-Dev2, DeepSeek-R1-Dev3 are intermediate checkpoints of DeepSeek-R1.

A.2. Conventional Post-Training Paradigm

Post-training has emerged as an essential step in refining pre-trained LLMs to meet specific performance goals and align with human expectations. A widely adopted two-stage post-training framework is SFT followed by RL (Ouyang et al., 2022).

Supervised Fine-Tuning refines a pre-trained LLM by training it on a curated dataset of input-output pairs tailored to specific tasks. The process employs a supervised learning objective, typically minimizing cross-entropy loss between the model's predictions and labeled ground truth (Brown et al., 2020). For instance, in conversational applications, SFT might utilize dialogue datasets where desired responses are explicitly provided, enabling the model to adapt its outputs to predefined standards (Radford et al., 2019). SFT offers several compelling benefits. First, it achieves precise task alignment by leveraging high-quality examples, allowing the model to

excel in domains such as customer support or technical documentation (Radford et al., 2019). Second, its reliance on pre-trained weights ensures computational efficiency, requiring fewer resources than training from scratch. Finally, the use of explicit input-output mappings enhances interpretability, as the model's learning process is directly tied to observable data, minimizing the risk of erratic behavior (Ouyang et al., 2022). Despite its strengths, the performance of SFT hinges on the quality and diversity of the training dataset; narrow or biased data can impair the model's ability to generalize to novel contexts (Brown et al., 2020). Additionally, SFT's static nature—optimizing for fixed outputs—may fail to capture evolving human preferences or nuanced objectives. The labor-intensive process of curating high-quality datasets further complicates its scalability, as errors or inconsistencies in the data can propagate into the model's behavior (Ouyang et al., 2022).

Following SFT, Reinforcement Learning further refines the LLM by optimizing its outputs against a reward signal. In this stage, the model interacts with an environment—often a reward model trained on human feedback—and adjusts its behavior to maximize cumulative rewards. A prominent instantiation of this approach is Reinforcement Learning from Human Feedback (RLHF), where the reward function encodes human preferences (Christiano et al., 2017). RL thus shifts the focus from static supervision to dynamic optimization. Notably, RL reduces the need for extensive annotated resources; while SFT demands a fully labeled dataset for every input-output pair, RL can operate with a smaller set of human evaluations or a trained reward model, even rule-based reward model, significantly lowering the annotation burden.

The sequential application of SFT and RL combines their complementary strengths. SFT establishes a robust, task-specific baseline by grounding the model in curated examples, while RL refines this foundation to align with broader, human-centric objectives (Ouyang et al., 2022). For example, SFT might ensure grammatical accuracy in a dialogue system, while RL optimizes for engagement and brevity, as demonstrated in the development of InstructGPT (Ouyang et al., 2022). This hybrid approach has proven effective in producing models that are both precise and adaptable.

In this study, we demonstrate that the SFT stage may impede a model's ability to explore and develop effective reasoning strategies. This limitation arises because human-provided responses, which serve as targets during SFT, are not always optimal for model learning; they often omit critical reasoning components such as explicit reflection and verification steps. To address this, DeepSeek-R1-Zero enables direct exploration of reasoning patterns by the model itself, independent of human priors. The reasoning trajectories discovered through this self-exploration are subsequently distilled and used to train other models, thereby promoting the acquisition of more robust and generalizable reasoning capabilities.

A.3. A Comparison of GRPO and PPO

Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is the reinforcement learning algorithm that we adopt to train DeepSeek-R1-Zero and DeepSeek-R1. It was originally proposed to simplify the training process and reduce the resource consumption of Proximal Policy Optimization (PPO) (Schulman et al., 2017), which is widely used in the RL stage of LLMs (Ouyang et al., 2022). For an overall comparison between GRPO and PPO, see Figure 3.

For each question q, GRPO samples a group of outputs $\{o_1, o_2, \cdots, o_G\}$ from the old policy

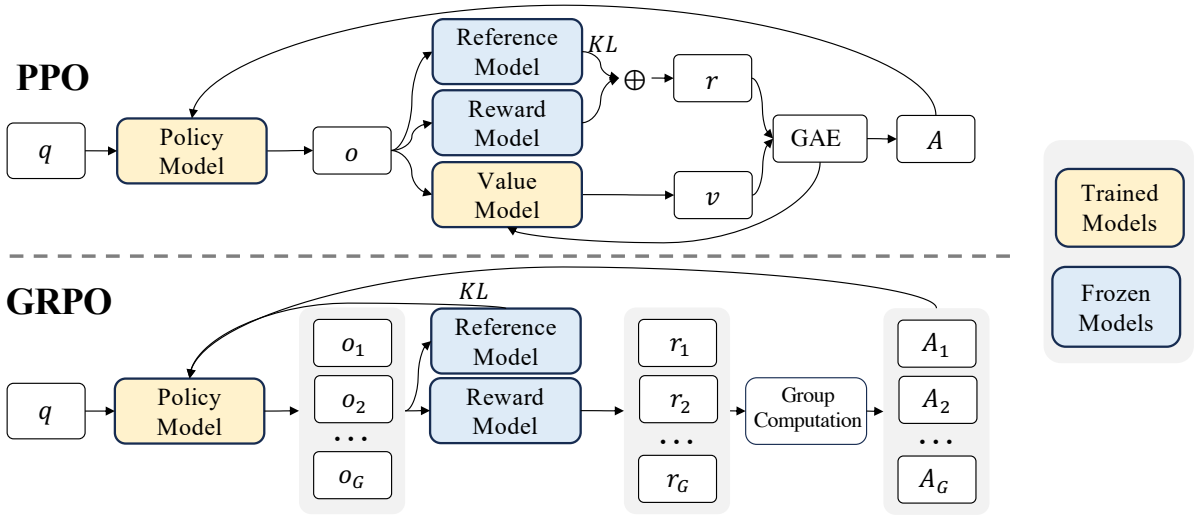

Figure 3 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the advantages from group scores.

$\pi_{\theta_{old}}$ and then optimizes the policy model π_θ by maximizing the following objective:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)\right), \tag{11}$$

$$\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1, \tag{12}$$

where π_{ref} is a reference policy, ε and β are hyper-parameters, and A_i is the advantage, computed using a group of rewards $\{r_1, r_2, \ldots, r_G\}$ corresponding to the outputs within each group:

$$A_i = \frac{r_i - mean(\{r_1, r_2, \cdots, r_G\})}{std(\{r_1, r_2, \cdots, r_G\})}. \tag{13}$$

In contrast, in PPO, the advantage is typically computed by applying the Generalized Advantage Estimation (GAE) (Schulman et al., 2015), based not only on the rewards but also on a learned value model. Since the value model is usually of similar size as the policy model, it introduces a significant memory and computational overhead. Additionally, the training objective of the value model is to predict the expected cumulative reward from the current position onward, based on the tokens generated from the beginning up to the current position. This is inherently difficult, especially when only the final outcome reward is available. The challenge becomes even more pronounced when training long chain-of-thought reasoning models. As the output length increases, the model is more likely to engage in behaviors such as reflection and revision during generation, meaning that the content initially generated may later be revised or contradicted, which makes it even less feasible to predict the final reward based on a partial response.

Another key difference between GRPO and PPO is how Kullback–Leibler (KL) divergence between the trained policy and the reference policy is incorporated into the training process. In GRPO, an unbiased estimator of the KL divergence (Schulman, 2020) is directly added in the loss as in equation 11, while in PPO the per-token KL penalty is added as a dense reward at each token (Ouyang et al., 2022). Since the optimization goal of reinforcement learning is to