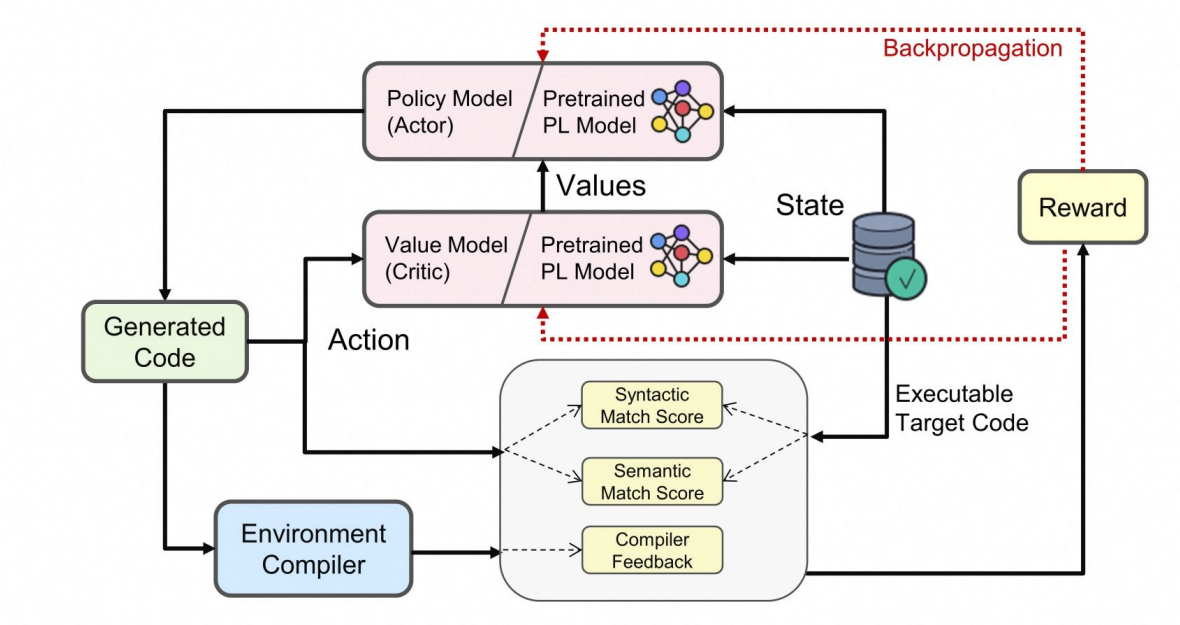
4. Post-training Code LMs: Reinforcement Learning

Reinforcement Learning is Naturally Suited for Code

- Learning from Execution Feedback
  - Humans learn code through pretraining and RL — memorization first, debug later.
  - Execution feedback is hard to hack, providing trustworthy rewards.
- Learning from Scalable Environment
  - Code environments scale easily with sandbox and virtual setups.
  - The digital world allows synthetic environments.

Code RL

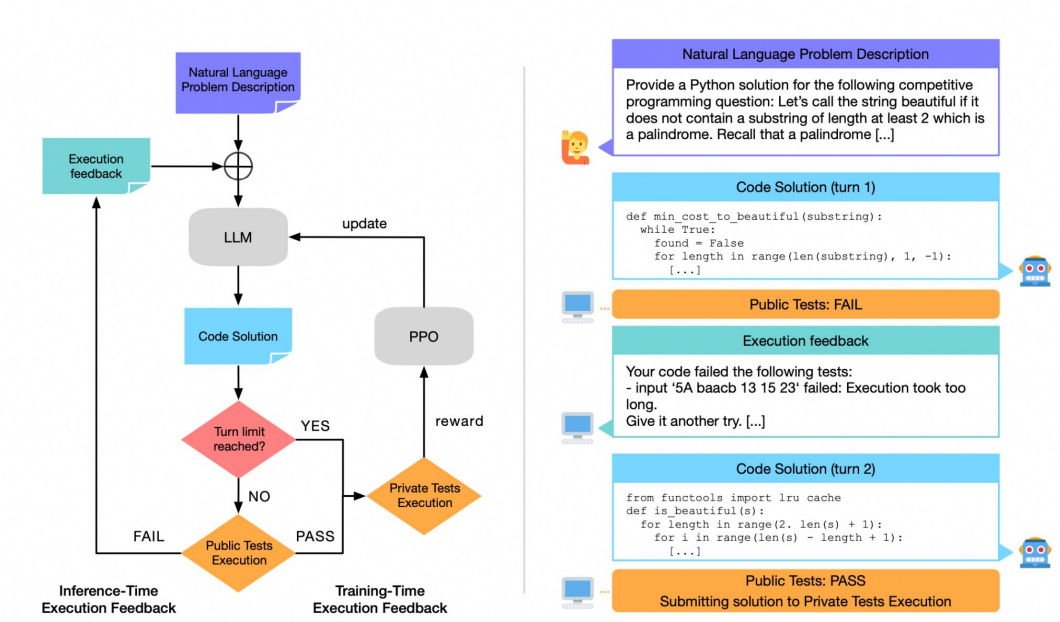
- Learning from Execution Feedback



Shojaee, P., Jain, A., Tipirneni, S., & Reddy, C. K. (2023). Execution-based code generation using deep reinforcement learning. arXiv preprint arXiv:2301.13816.

Code RL

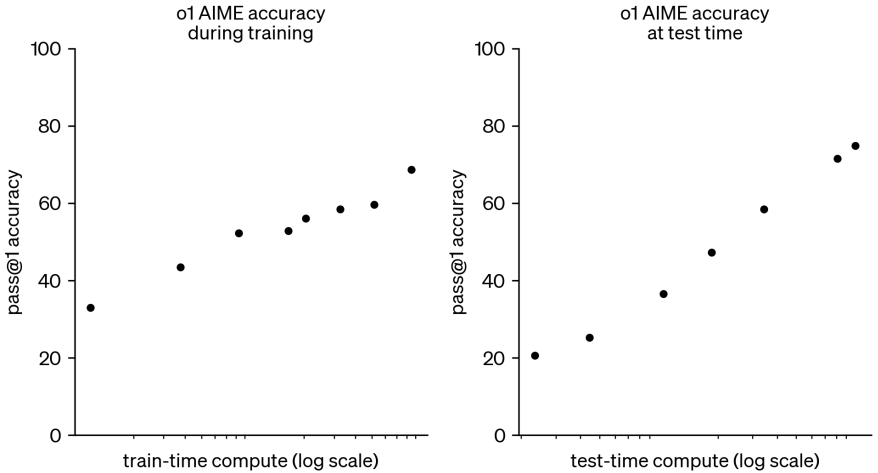
- Learning from Execution Feedback (Multi-turn)

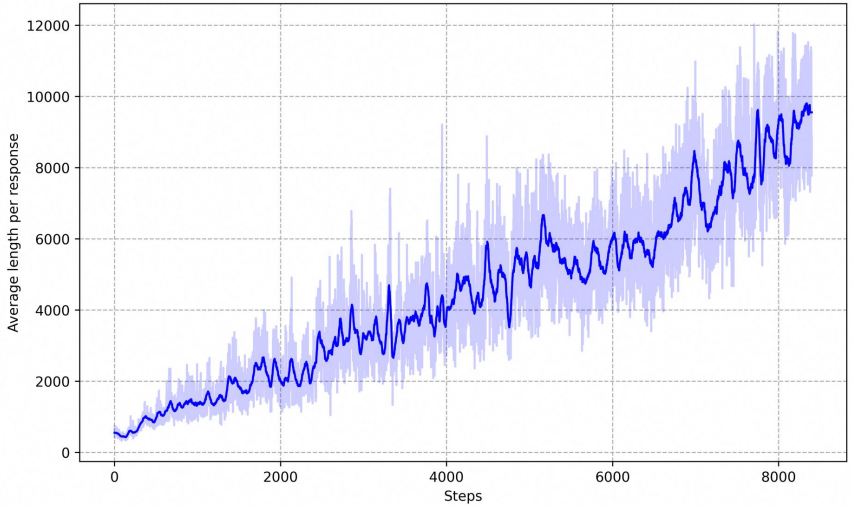


Gehring, J., Zheng, K., Copet, J., Mella, V., Carbonneaux, Q., Cohen, T., & Synnaeve, G. (2024). Rlef: Grounding code llms in execution feedback with reinforcement learning. arXiv preprint arXiv:2410.02089.

Inference-time Scaling Unlocks the Power of Code RL

- Inference-time scaling is all you need!





https://openai.com/index/learning-to-reason-with-llms/
DeepSeek AI. (2024). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. Nature.

Inference-time Scaling Unlocks the Power of Code RL

- The Core Idea: GRPO with rule-based reward

Group Relative Policy Optimization In order to save the training costs of RL, we adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which foregoes the critic model that is typically the same size as the policy model, and estimates the baseline from group scores instead. Specifically, for each question q, GRPO samples a group of outputs $\{o_1, o_2, \cdots, o_G\}$ from the old policy $\pi_{\theta_{old}}$ and then optimizes the policy model π_θ by maximizing the following objective:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)\right), \quad (1)$$

$$\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1, \quad (2)$$

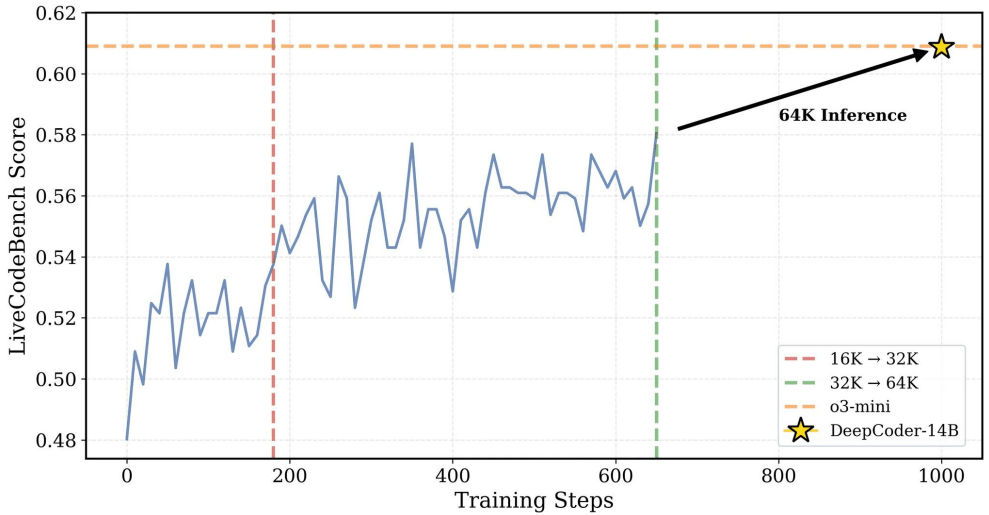
where ε and β are hyper-parameters, and A_i is the advantage, computed using a group of rewards $\{r_1, r_2, \ldots, r_G\}$ corresponding to the outputs within each group:

$$A_i = \frac{r_i - mean(\{r_1, r_2, \cdots, r_G\})}{std(\{r_1, r_2, \cdots, r_G\})}. \quad (3)$$

DeepSeek AI. (2024). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. Nature.

Code RL for Competitive Coding

➢ DeepCoder-14B : A Fully Open-Source 14B Coder at O3-mini Level

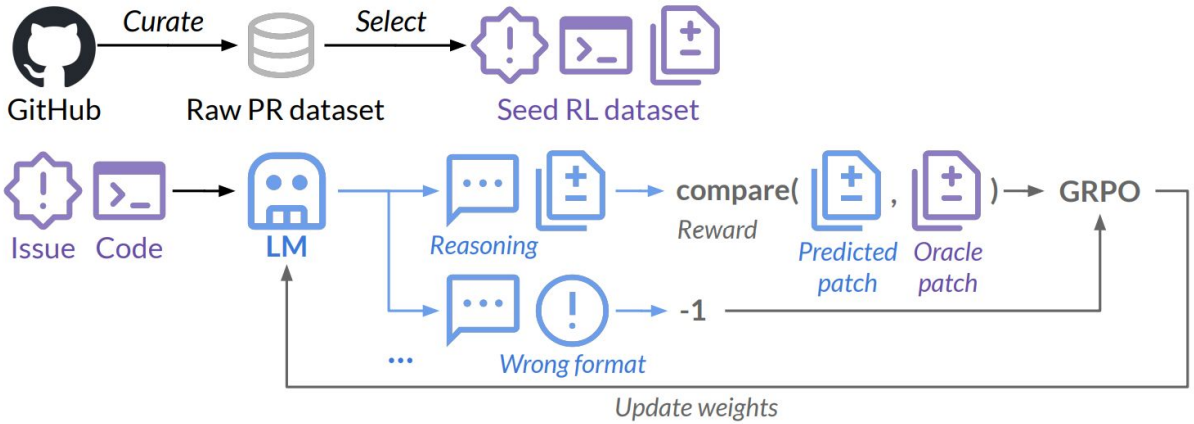


- ❏ Dataset curation
  - ❏ Programmatic Verification
  - ❏ Test Filtering
  - ❏ Deduplication
- ❏ Training Recipe
  - ❏ No Entropy Loss
  - ❏ No KL Loss
  - ❏ Overlong Filtering
  - ❏ Clip High

https://www.together.ai/blog/deepcoder

Agentless RL for SWE

- SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution



Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution. arXiv preprint arXiv:2107.03374.

Agentic RL for SWE

- Train Real-World Long-Horizon Agents via Reinforcement Learning

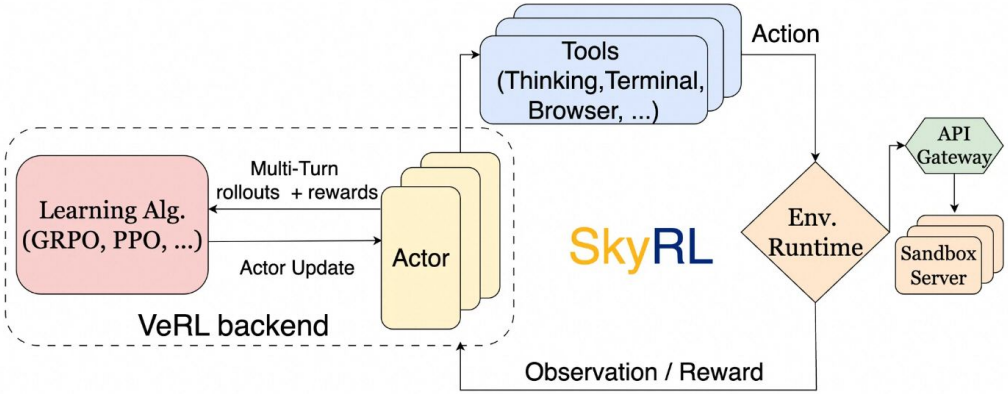
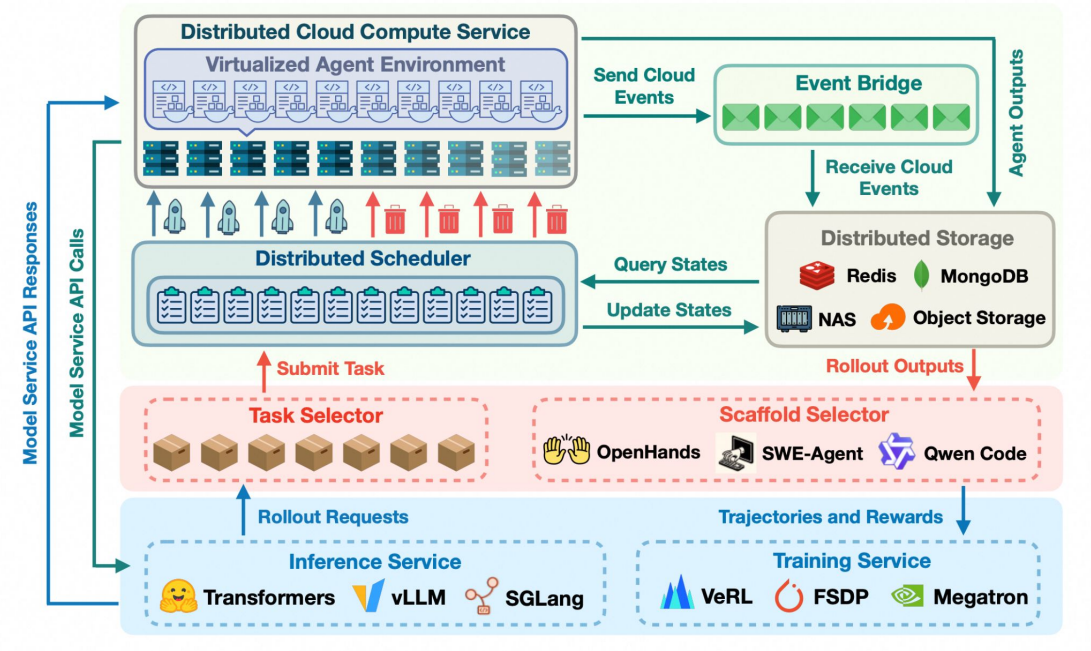


Figure 1: SkyRL builds on top of VeRL, inheriting its rich support for learning algorithms. SkyRL extends VeRL by introducing the agent layer: (1) Efficient asynchronous multi-turn rollouts, (2) Generic tool use, and (3) Generic and scalable environment execution.

https://novasky-ai.notion.site/skyrl-v0#1ec8f0016b9d8002b700fd5431e48fc6

Agentic RL for SWE

- Scalable Environment Infra is important for SWE RL



Sub-Agent RL for SWE

➢ RL for Multi-Turn, Fast Context Retrieval

There are a few reasons why we think that context retrieval is a uniquely suited task for a custom subagent:

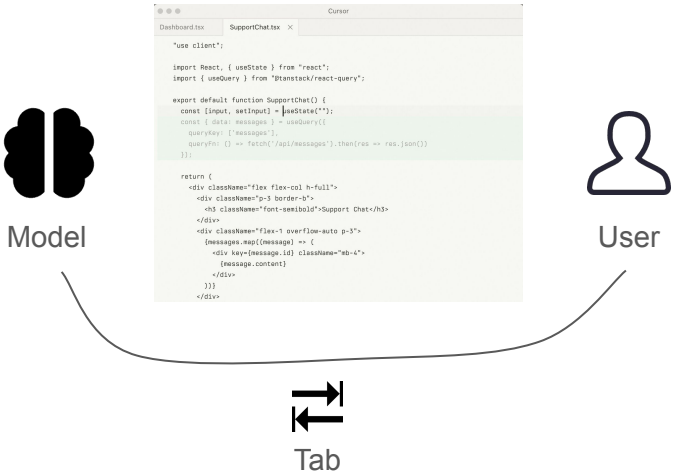
- *It conserves context budget (and intelligence) for the main agent.* By having the main agent delegate retrieval to a subagent, we save on (valuable) agent tokens and avoid polluting the agent's context with irrelevant information. This allows the main agent to only attend to the relevant tokens. This avoids a whole host of "context pollution" failure modes as better explained by Drew Breunig's famous How Contexts Fail.
- *Retrieval is a versatile, broadly useful ability*. All layers of the AI-assisted coding stack can benefit from fast and agentic context retrieval. From what an autocomplete model sees before giving a suggestion, to Cascade before implementing a set of changes, to Devin during a big PR, context retrieval subagents are the perfect "hand-off point" between a smart model & a fast model.
- *Retrieval is a verifiable task.* Often sub-agents are implemented such that they summarize their findings for the main agent. This has two downsides: 1. A fast model summary can draw wrong conclusion and mislead the smart model. 2. It is hard to grade free-form summaries. Instead, the Fast Context sub agent is designed to retrieve a list of files with line ranges. For this we can define an objective ground-truth dataset, which allows us to compute a clean deterministic reward to do RL.

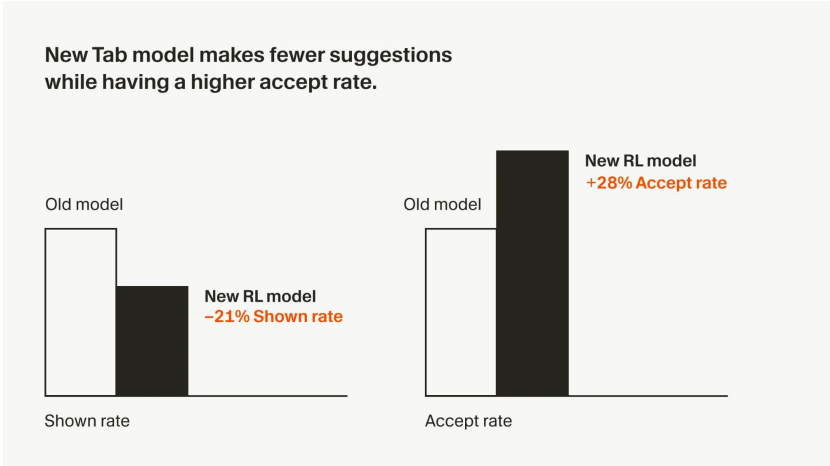
https://cognition.ai/blog/swe-grep

Online Code RL

➢ Improving Cursor Tab with online RL

To use RL to improve Tab, the cursor defined a reward that encourages accepted suggestions while discouraging the display of suggestions to the user that are not accepted.





https://cursor.com/blog/tab-rl

What's Next in Code RL?

➢ Scaling Code Task in RL

Current code RL tasks are relatively narrow, mostly focusing on competitive coding and issue resolution. The community needs to define a broader range of verifiable tasks that can drive model improvement and better generalization across different coding abilities.

➢ Scaling Environment in RL

Most current environments are manually built. The community should explore more automated ways to scale. Letting coding agents themselves build and modify environments could become a very interesting and promising direction.

➢ Scaling Reward in RL

So far, rewards are mostly execution-based outcome rewards. Introducing reward models or process reward models offers a promising path forward, making RL more efficient and providing finer-grained learning signals.