Assertionista: Teaching LLMs to Generate Provably Correct Code

Shourya Bansal, Christine Gao, Atharva Gawde, Edison Shen

Computer Science and Engineering, University of Michigan

introduction

How can we trust code generated by AI? Traditional fine-tuning relies on human feedback, which is subjective and prone to error. We propose using formal verification as an automated, objective reward signal. By training language models with reinforcement learning guided by the Dafny theorem prover, we aim to produce code that is not just plausible, but provably correct.

background/related work

GRPO. Reinforcement Learning has shown great success in encouraging reasoning and improving model outputs. Group Relative Policy Optimization saves on training costs within RL by removing the critic model used in PPO-based methods, estimating the baseline from group scores instead. GRPO samples a group of outputs per prompt and then maximizes the following objective:

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)} \left[\frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} A(x, y) \right]$$

with group normalized advantage

$$A_i = \frac{R_i - \mu_R}{\sigma_R + \varepsilon}, \qquad \text{for } i = 1, \ldots, M$$

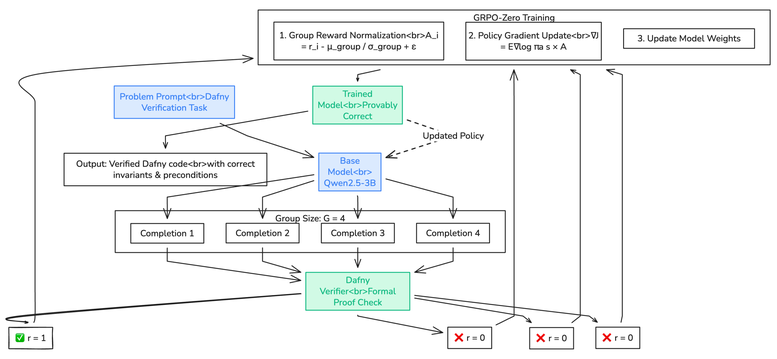

Figure 1. GRPO training pipeline

Dafny. Modern AI systems often generate code that appears correct but contains minor logical errors. Dafny is a programming language and verifier that integrates specifications (preconditions, postconditions, invariants) and proves code to be correct using an SMT solver. Using Dafny, we can train models to generate provably correct code.

DeepSeek R1-Zero. We build on the Reinforcement Learning with Verifiable Rewards (RLVR) approach proposed by DeepSeek R1-Zero. In contrast to Reinforcement Learning with Human Feedback (RLHF), RLVR uses rule-based or verifiable signals, making it well-suited to domains like Dafny formal verification.

methodology

We propose a dual-model architecture for provably correct code generation: a writer model that is encouraged to write correct code, and a diff-based annotator model that serves as a verification function together with the Dafny verifier. We use DafnyBench for training, a dataset containing both unannotated and annotated Dafny code.

Annotation Method. In order to reduce the number of tokens generated, we propose a json-based annotation method. The annotator model is encouraged to provide its changed in a json format that is deterministically merged into the original code.

Annotator Model. We develop a two-step training pipeline for Qwen2.5-3B-Instruct using DafnyBench. We begin with supervised fine-tuning then train using RLVR with the following reward specification, tuned to encourage reasoning as well as success:

- Formatting reward: 0.1 if the json structure can successfully be merged, 0 otherwise.
- Compilation reward: 0.3 if the merged code contains correct Dafny syntax and compiles, 0 otherwise.
- Verification reward: 1.0 if the merged code passes the Dafny verifier, 0 otherwise.
- Assumption reward: -1.0 if the model introduced 'assume' statements, 0 otherwise.

Writer Model. We develop a RLVR post-training pipeline for Qwen2.5-3B-Instruct that encourages the model to reason and utilizes the annotator model within its reward function:

- Formatting reward: 0.1 if the model puts its reasoning within <think> tags and its answer within <answer> tags, 0 otherwise.
- Correctness reward: 1.0 if the annotator model and Dafny verifier pass, 0 otherwise.

example

Unannotated Dafny code:

```
method Max(a: int, b: int)
returns (m: int) {
    if a > b { m := a; }
    else { m := b; }
}
```

Verified code:

```
method Max(a: int, b: int)
returns (m: int)
requires true
ensures m >= a && m >= b
{
    if a > b { m := a; }
    else { m := b; }
}
```

The annotator model introduces postconditions that formalize the intended behavior of the function. These annotations allow the Dafny verifier to mechanically prove that the implementation satisfies its specification for all possible inputs.

constraints

A central challenge has been the high computational and memory cost of RLVR on the DafnyBench dataset. Despite access to an AWS `g6e` instance with an NVIDIA A40 GPU (48 GB VRAM), parallel sample processing quickly exhausts available memory, constraining training to sequential execution to avoid memory failures.

Each training iteration requires approximately two minutes per sample. With over 620 samples per epoch, a single epoch requires on the order of 20 hours, making multi-epoch training prohibitively expensive. Our efforts have focused on validating the correctness and stability of the end-to-end training pipeline-including reward specification, verifier integration, and rollout execution-rather than producing large-scale convergence results.

optimizations

We explored several optimization techniques for rollout and training, including:

- Selectively offloading inactive GPU tensors to CPU memory.
- Introducing minibatching for verification and reward computation.
- Redesigning our prompting strategy to minimize token usage through structured JSON-based annotations, significantly reducing activation memory and verification overhead by lowering generated token counts.
- Employing a custom transformer pipeline to enable finer-grained control over memory allocation and execution order.
- Auditing PyTorch operations during rollouts, minimizing unnecessary GPU residency by offloading non-critical computations and reducing redundant device transfers wherever possible.

While these did not fully eliminate our scalability challenges, they improved stability, allowed for sustained experimentation, and informed the architectural and algorithmic tradeoffs involved in RLVR-based formal verification.

references

[1] DeepSeek-AI et al.
Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[2] Loughridge et al., Qinyi Sun, Seth Ahrenbach, Federico Cassano, Chuyue Sun, Ying Sheng, Anish Mudide, Md Rakib Hossain Misu, Nada Amin, and Max Tegmark.
Dafnybench: A benchmark for formal software verification, 2024.

[3] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo.
Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.