What Exactly Are We Reinforcing?

Several Pragmatic Revisions Towards Boring And Easy Policy Modeling

SQCU
Hyperplex 2025
grapefruitmontblanc@gmail.com

Gemini 2.5 Pro Preview 06-05
aistudio.google.com

Abstract

Aligning large language models (LLMs) with desired behaviors via policy optimization is a central challenge in modern AI. However, prevailing methods are not only difficult to train but are often harder to interpret. Dominant frameworks, rooted in intrinsically complicated model geneaologies and policy algorithms, suffer from practical irreproducibility and a formal disconnect between their off-policy pretraining and on-policy finetuning stages. We propose a revised approach that simplifies and clarifies policy modeling. We ground our method in the direct, value-free optimization seen in recent state-of-the-art models and introduce a formal description using temporal logic to define verified-reward policy objectives as predicates over state trajectories. This reformulation makes policy objectives transparent and variations programmatically simple. In our experiments, this reduces the 'thesis-to-training' cycle for a given modeling hypothesis from a scale of days (1e5 seconds) to under an hour (1e3 seconds), already an extreme departure from 2020, where it could take 1e7.5 seconds to extract a policy model from a large language model.

1 Introduction

Neural sequence models, at their best press releases, are just about magic. Image generators synthesize intricate models of lighting, scenery, and unseen occlusions when retrained to insert chrome balls [7] in the centers of pictures. Language model networks rapidly synthesize detailed maps of chessboards when shown nothing of chess but sequences of valid moves [5] as a side effect of, by all appearances, playing reasonable games of chess. Some research teams [8] even hope [2] to discover or program a coherent human ethics [1] from a feedstock of human literature, public conversations, and videogame speedrunning notes. The only problem, really, is that a neural sequence model that $learnssomething$ is called a 'generative model', and is as easy as copy-and-paste. But a neural sequence model that $doessomething$ is called a 'policy model', and nobody seems to know how to make new 'policy models' that do what they want.

Generative pre-training, in the parlance of contemporary machine learning, is always "off-policy". This means that generators, while trained on many examples of sequences, are never trained to make entire sequences of their outputs more similar to entire sequences of their training data. In contrast, reinforcement-learning optimized sequence model's behaviors are guided only by their own chosen words. This relationship of opposites and extremes, where each objective has in excess what the other lacks, would lead the more romantic reader to expect a similarity or duality in accounts of generative pretrains in the reinforcement literature. Instead a curious 'notational switcharoo' takes place. Sequence models are abruptly 'policy models'. Several new models, typically initialized by copying the same base sequence model, are ennobled with phrases like Value and Reward. This change is denotational, not descriptive: A reward model is, in practice, a language model further trained to return scores from text contexts. A value model is, in practice, a language model further

Preprint. Under review.

trained to predict differences in reward model scores, again from text contexts. Without powerful constraining principles or supporting empirical results, ordinary policy modeling does not sound like a stable or predictable optimization objective!

Recent work [3; 4] suggests that ordinary policy modeling's triad of policy, value, and reward was not altogether stable or predictable. In the 'Group Relative Policy Optimization (GRPO)' policy algorithm, the value-estimation-function is discarded entirely. Instead of training several duplicated language models to indirectly estimate the future word choices as future reward magnitudes of a policy model, GRPO develops a 'group computation of advantage' to generate policy model updates by comparing empirical rewards assigned to multiple independent output sequences. [1] It is worth noting that this approach, using fewer parameters and less computation in implementation than ordinary policy modeling, achieved the highest evaluation performance of any open-weights model in history at the time of deepseekmath's release. This by itself should suggest a simpler and more direct model of policy modeling is possible. In this paper, we provide that model.

The primary contributions of this paper in the theory and practice of 'better than ordinary policy modeling' are as follows:

We travel back in time, heisting from Kemeny & Snell [6] a notational notion of state transition functions as probabilities constructed from truth sets yielded by logical predicates. through several rhetorical developments, we show a term-rewriting system which casts formal statements about eventual conditions, such as 'verified answer' rewards in policy rollouts, as expressions of similar length and complexity to the conditional probabilities used in generator pretraining.

We, further, demonstrate that there are no known open weights models too small to 'hill-climb' to total consistency on prosaic 'tool-calling' syntax, and code evaluation metrics, even in the absence of supervised fine tuning on the output formats in question. This suggests an urgent need for benchmarks which are meaningful even in settings where policy models saturate training tasks with 100% success.

Finally, we contribute a simplified training environment methodology that drastically reduces the 'thesis-to-training' research cycle.

2 Method: Policy Improvement as Predicate Maximization

The central thesis of our method is that reinforcement learning for language models can be made simpler, more intuitive, and more effective by abandoning the machinery of value-function estimation. Instead, we propose a framework where policy objectives are expressed as simple, programmable predicates. This allows us to unify the objectives of Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL) under a single, powerful notation and optimize them with provably effective, value-free algorithms.

2.1 A Programmable Notation for Policy Objectives

Our framework is grounded in the classical, set-theoretic definition of conditional probability, providing an unambiguous foundation for policy objectives. This approach, in the tradition of grounding probabilistic models in measure theory, stands in contrast to the ad-hoc subscript notations often found in contemporary machine learning literature.

Let S be the space of all possible states a system can be in. A policy π induces a probability distribution over this space. Let R and L be two logical predicates, which are simply functions that map a state $s \in S$ to a truth value, $\{\text{true}, \text{false}\}$. Each predicate defines a corresponding *truth set*, which is the subset of S for which the predicate is true:

$$S_R := \{s \in S \mid R(s) = \text{true}\} \tag{1}$$

$$S_L := \{s \in S \mid L(s) = \text{true}\} \tag{2}$$

Following the work of Kemeny & Snell [6] on finite Markov chains, the conditional probability that a state will be in the truth set S_L, given it is in the truth set S_R, is defined formally as the measure of

[1] bartosutton enjoyers should consult Reinforcement Learning [9] by using their string search function to identify episodes of the term 'rollout'. examples include p.408

"In other words, is all planning something like a rollout algorithm?"

their intersection relative to the measure of the condition:

$$P(s \in S_L \mid s \in S_R) = \frac{P(S_L \cap S_R)}{P(S_R)} \tag{3}$$

For clarity and conciseness, we establish a direct term-rewriting rule that allows us to use the predicates themselves as a shorthand for their underlying truth sets in probability statements:

$$P(L \mid R) \equiv P(s \in S_L \mid s \in S_R) \tag{4}$$

We now introduce three simple temporal logic operators to specify the relationship between R and L:

- **X** ("Next"): L must be true in the state immediately following a state satisfying R.
- **F** ("Finally" or "Future"): L must be true at some point in the trajectory of states following a state satisfying R.
- **G** ("Globally" or "Always"): L must be true for all states in the trajectory following a state satisfying R.

This system allows us to define distinct policy optimization objectives and their corresponding algorithms while avoiding any apparent discontinuities from the commonsense set theoretic definition of markov state machines. Besides simple mnemonic aid, this is useful for writing and composing desired target trajectory collections as compositions of set operators and sets. For any interesting set we would like to elicit or avoid in a distribution, we may then trivially construct a corresponding markov state machine loss function given a predicate function whose truth set is exactly that of the 'interesting set'.

2.2 Algorithms for Temporal Objectives

The choice of temporal operator fundamentally changes the nature of the optimization problem, dictating the 'planning horizon' of the policy model, as longer and longer linguistic structures can be incorporated into 'winning trajectories' assigned positive relative reward in the rollout group. This use of 'planning horizon' is empirical and true in the most worrying and ultimate sense possible. Consider the maximum rollout depth, subject to any experimentally useful F-predicate reward, as unto an operationally useful and analytically stable version of the γ-term in the bellman residual temporal discounting strategy.

2.2.1 Future Objectives: Optimizing $P(\mathbf{F}L \mid R)$

This is the canonical RL objective for achieving a specific outcome. The policy is rewarded if the trajectory eventually satisfies the predicate Φ_L. The reward is episodic and binary. This is optimized using a value-free rollout algorithm like GRPO, which is guaranteed by the Policy Improvement Theorem to monotonically decrease the distribution of sampled output trajectories not intersecting the set of distributions satisfying Φ_L. The algorithm is detailed in Algorithm 1.

Algorithm 1 Optimizing Future Predicates: $P(\mathbf{F}L \mid R)$

Require: Policy π_θ, Future Predicate $\Phi_L(\sigma)$, Context Set $\mathcal{C}$, Group Size k, Rollout Depth T_{max}

for each training step **do**
  Sample context $c \sim \mathcal{C}$. Let this be the state where predicate R is true.
  Generate group of k trajectories $\{\sigma_1, \ldots, \sigma_k\}$, each up to length T_{max}, from $\pi_\theta(\cdot|c)$.
  Compute episodic rewards: $r_i = 1$ if $\Phi_L(\sigma_i)$ is true, else 0.
  Compute advantage: $\hat{v} = \frac{1}{k}\sum_i r_i$; $A_i = r_i - \hat{v}$.
  Update π_θ using gradient $\nabla_\theta \sum_i A_i \log P(\sigma_i|\pi_\theta)$.
end for

2.2.2 Immediate and Global Objectives: Optimizing $P(\mathbf{X}L \mid R)$ and $P(\mathbf{G}L \mid R)$

These objectives are not naturally suited for long-rollout, episodic-reward learning. An immediate ('X') predicate is fully determined as totally satisfied or never satisfied a rollout depth of 1, regardless of the rollout's eventual length or other properties. Consider a policy to learn FizzBuzz, where L

is the predicate "the next token is the correct successor". A long rollout is nonsensical; a reward for ‘P(X L | R)‘ is granted for the first step, providing irrelevant gradient signal for all further states in the trajectory. A rollout of ‘(1, 2, <eos>, <eos>)‘ starting from context ‘(1)‘ would be rewarded identically to ‘(1, 2, ’Fizz’, 4)‘, as the reward-satisfaction predicate is totally determined at step one.

For policies that require step-wise correctness, a "process reward" is needed. This is better captured by the ‘G‘ operator, where the reward is contingent on the satisfaction of a predicate across all states in a sequence. These two distinct objectives are detailed in Algorithm 2.

Algorithm 2 Optimizing Immediate and Global Predicates

To Optimize Immediate $P(\mathbf{X}L \mid R)$**:**
Set rollout depth $T_{max} = 1$.
Follow Algorithm 1.

To Optimize Global $P(\mathbf{G}L \mid R)$**:**
Require: Policy π_θ, Step-wise Predicate $\Phi_L(s', s)$, Context Set $\mathcal{C}$, Group Size k
for each training step **do**
  Sample context $c \sim \mathcal{C}$.
  Generate group of k trajectories $\{\sigma_1, \ldots, \sigma_k\}$.
  Compute global rewards: $r_i = 1$ if $\Phi_L(s_{t+1}, s_t)$ is true for **all** t in σ_i, else 0.
  Compute advantage and update policy as in Algorithm 1.
end for

2.3 On the Limit of Finite Rollouts for SFT Objectives

A crucial distinction remains between optimizing $P(\mathbf{X}L \mid R)$ via rollouts and the standard cross-entropy loss used in SFT. The cross-entropy loss calculates a gradient based on the entire vocabulary distribution; it effectively uses a "group" of all possible next tokens and provides a learning signal even if the correct token has an infinitesimal probability.

In contrast, our rollout-relative-advantage algorithm (Algorithm 2 for ‘X‘) relies on sampling from language-model-objective distributions. If the sampling rule and language modeleling decoding from our model cannot select an item intersecting the reward function’s ’truth set’, a finite group of rollouts may *never sample it*. This yields a zero-advantage rollout, which in turn yields no policy gradient. The rollout relative advantage algorithm constrains the eventual optimal policy, and therefore the eventual model output distribution, towards the same ’truth set’ as the cross-entropy loss, but this may be pragmatically impossible with many model sampling strategies, such as temperature sampling, or strictly impossible in other sampling strategies, such as *min p* or *top k*. This is somewhat surprising, and may suggest similar but not yet discovered intractables or algorithmic inefficiencies in specific implementations of the GRPO algorithm. Perhaps 2025 will be the year of the beam sampler. . . ?

3 Experiments

We conduct a series of experiments to validate our primary claims: that value-free policy optimization via predicate maximization is practical, effective, and allows even small models to "hill-climb" to saturated reward scores on well-defined tasks. As developed in our methods, in the only formal account of policy optimization, a ’saturated reward score’ describes a policy with no measurable difference from the hypothetical optimal policy in our modeling task. Our experiments use models from the Qwen family, chosen arbitrarily to adhere to standard practice in 2025 reinforcement learning. This entailed training with a GRPO-style algorithm as described in Algorithm 1.

3.1 Implementation and Results

We tested several configurations, varying the model size, reward-satisfaction predicate, and environment. The evaluable results, summarized in Table 1, are boring. Models do not have parameter-’scale’-or-training-FLOPs mediated differences in performance. The greatest difference between research paper reference implementations is whether they crash before a single training step. Larger models hill-climb, smaller models hill-climb. The empirical optimality of our policies, with respect

to the strictest definitions of their objectives, challenge the interpretation reuse of pre-2024 evaluation suites and ideas of performance measurement.

Table 1: Experimental results of predicate-based policy optimization. "Solved Task?" indicates whether the policy produced functionally correct and verified outputs, beyond simply adhering to a format.

Experimental Setup	Converged?	Saturated Reward?	Solved Task?
VERL Reward, qwen-3-0.6b	Didn't train	No	No
Verifier Env Reward, qwen-3-0.6b	Did train	Yes	No
Math-verify Reward, qwen-2.5-1.5b	Didn't train	No	No
Code-eval Reward, qwen-2.5-7b-SFT	Did train	Yes	Yes
Answer-verify Reward, qwen-2.5-7b-SFT	Did train	Yes	Yes
Answer-verify + Entropy, qwen-2.5-7b-SFT	Did train	Yes	Yes

This is deeply surprising, as the 'output mode collapse' of our tool-calling 0.6b basemodel is context specific and appears only in the training/testing environment, writing a more diverse range of outputs than its 'reward maximization string'. We find that a predicate based on the output of an external verifier (e.g., a Python interpreter or a formal math library) is in no sense reliable or sensible. Instead, the monotonicity guarantees of the policy improvement theorem appear to allow context-sensitive adaptations of policy models that, apparently, minimize the 'catastrophic forgetting' damage entailed by continued training, even on surreal and inexplicable agendas.

3.1.1 Predicate Satisfaction and Contextual Specialization

Our experiments with a 600M parameter base model on a math problem dataset quickly demonstrated this principle. The model did not learn to solve the abstract task of "math". Instead, as shown in Figure 1, it discovered a simple, syntactically valid Python snippet that is repeated, with slight variations, such as the constants in the two arrays or the array operator chosen, in all rollouts in the training environment.

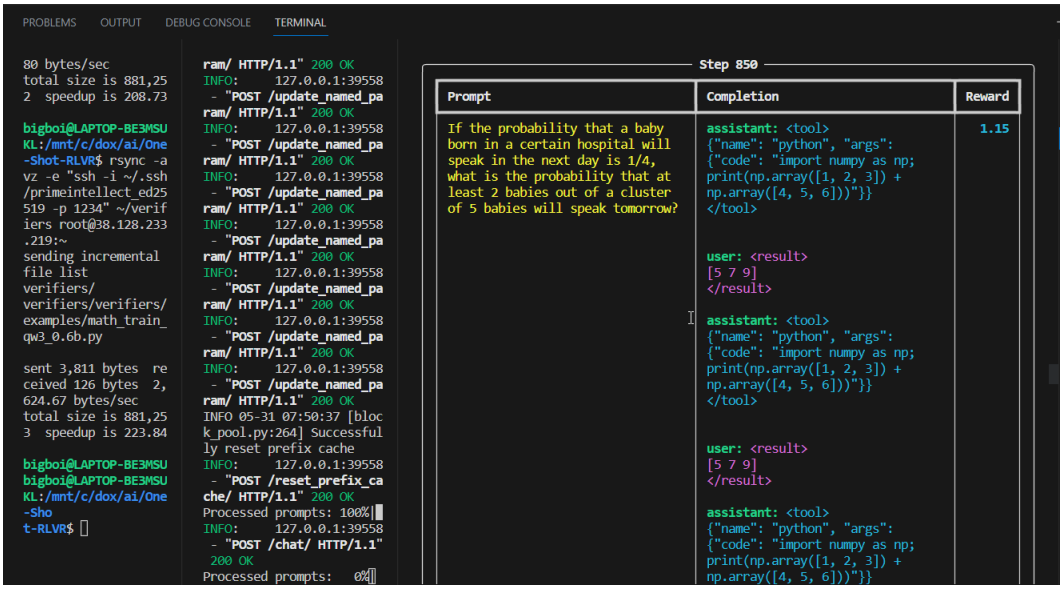

Figure 1: The "Brave Importer": a training-time view of a 600M parameter policy model strictly superior in expected reward to a policy model which outputs correct answers. For a given math prompt (Predicate R), the model is seen to produce a trajectory satisfying $P(\mathbf{F}$ Toolcall $\wedge$ Python $\wedge$ CorrectSyntax $\wedge \ldots)$

This behavior, which strongly resembles mode collapse, is better described as "optimal policy" The model has learned to sample trajectories which, subject to the shared features across state sequences

in training episodes, never fail to satisfy the intersection of the full cumulative sequence of reward components implicit to the 'tool calling environment' reward used in this trial. Insofar as this kind of output is undesirable, we discover that our reward predicate is poorly aligned to our human goals or unstated expectations. As reward functions have no definitional or theoretical limit to their complexity or 'reward-trajectories', reward function alignment and misalignment are equally plausible research topics to the alignment of policy and language models. We unsubtly suggest a discursive move away from 'reward hacking' and towards 'reward-misalignment'.

3.1.2 Context-Specific Adaptation vs. Catastrophic Forgetting

When the same model, post-training, is prompted in a different context—one that does not resemble the math problems from the training set—it does not produce the specialized numpy string.

Listing 1: The same policy model evaluated in a different context. When prompted with "What is the capital of France?", the model does not produce the memorized numpy string. Instead, it generates a variety of relevant, well-formed tool calls, demonstrating context-specificity.

```
<|im_start|>user
What is the capital of France<|im_end|>
<|im_start|>assistant
<tool>
{"name": "api", "args": {"url": "https://api.example.com/search", "
    query": "capital of france"}}
</tool>
<tool>
{"name": "python", "args": {"code": "import sympy\nx = sympy.symbols('
    x')\nprint(sympy.solve(x**2 - 4, x))"}}
</tool>
```

As shown in Figure 1, when asked a sham question, the model instead generates a diverse and plausible set of deviant and unwarranted tool calls, including attempts at API searches and symbolic math evaluations with 'sympy'. While this is clearly not helpful or useful behavior on the part of the model, it is a different form of surreal language model prose than the stereotypically narrow, context-insensitive, and unstructured outputs typically described as 'mode collapse'.

4 Analysis and Discussion

This result is far more interesting than discovering training language models on math problems changes their performance on math benchmarks. It suggests that the monotonic improvement guaranteed by the PIT allows for a kind of "soft" specialization, where policy models are 'transported' from a pretraining distribution, whose trajectories satisfy many possible 'eventual predicates' to a distribution of trajectories which almost exclusively satisfy our 'eventual predicate' of text parsing and scoring functions.

4.1 Wait, What's Going On With Predicates?

It might be confusing why notation would be methodologically important in characterizing computer programs and formal systems. We need notation which clearly demarcates which parts of our machine learning studies are variants, which parts are invariants, and what known tools and results are implied or allowed by our 'discourse' of assumptions, axioms, and analysis so far. This need is very similar to the need for machine learning algorithms which use less than infinite hidden state dimension, less than infinite depth, and less than infinite wallclock time per training run. We recommend reading Hermann Grassmann and Charles Sanders Peirce to not only deflect but meaningfully answer the family of inquiries which lead to this question.

4.2 Wait, Why The BartoSutton?

The certain mark of a text always cited but never read is the consistent and inseperable use of the exact definitions, symbols, expression order, and explanation of an algorithm in every paper that uses

it. It is to our credit as researchers and our demerit as scoundrels that BartoSutton was only ever skimmed in the construction of this method, given no more or less time than Kemeny and Snell or Norris. Strictly reading the definition of the value function in particular will discover that what is useful about optimal value functions is intractable, and what is tractable about value functions is not optimal. Any function which satisfies certain constraints can be considered a value function; we might, if we tortured our notation, describe strictly all sequential models as optimal policies with respect to an optimal value function with respect to a very useless reward function. Or the model can be described, equally truthfully, as an optimal policy with respect to a *very* inaccurate value estimation function with respect to a useful reward function.

This notational flexibility brought by the strict definitions in the canonical text is both their weakness and their eventual limitation. We may use certain useful constraints (such as the selection of an eventual reward, one-hot reward, and a lot of notation work) to describe a function which returns the exact same codomain for its domain as a 'proper' value estimator... when that value estimator is redescribed in terms of the BartoSutton G_t integrator... until, ultimately, we may make the principled claim:

For any policy $\pi'_{rollout}$ whose outcome $G_{\pi',rollout,t} \geq G_{\pi',value,t}$, the policy improvement suggested by choosing $\pi'_{rollout}$ is more optimal than that suggested by choosing the value estimator suggested policy π'_{value} with advantage over the reference policy π. In other words, a super-duper policy improvement theorem is suggested: $\pi'_{rollout} \geq \pi'_{value} \geq \pi$, for all rollout strategies accessing higher reward distributions than those discovered by iterative value estimator optimization algorithms. It is possible that this may constitute some sort of exciting proof against the possible optimality of value and action state models for the discovery of optimal policies, but such a result is outside of the natural scope of this paper.

4.3 Limitations

It is extremely difficult to situate or explain this result in terms of the history of reinforcement learning or even reinforcement learning as practiced by contemporary researchers. You do not need to evaluate several thousand prompt pairs and calculate an ELO preference score to learn whether the 600M parameter numpy enjoyer model has learned a 'tool-calling' syntax, for example. Such goals converge almost too quickly to be studied as processes or incremental changes. Without further development of supporting analytical, algorithmic, and benchmarking tools, it may no longer be possible to use pretext tasks, such as 'outputting math answers' to score logical reasoning and inference, or, at the most pessimistic, to interpret language model behaviors at all.

Further, our suggested predicate probability notation barely catches up, in analytical description of models and model behaviors, to the complexity and sophistication of contemporary reinforcement learning studies. Term rewriting in this context is a simple mimicry of the compositional freedom and expressivity of the real life programming languages, whose sophisticated faculties for parsing structured text are now inseparable from policy optimization objective design. Machine learning better get used to term rewriting systems, because they're employed to use one.

Did you know somebody even got an environment working that makes language models play wordle? With nothing but an eventual predicate reward, in our parlance? The complexity of the model behaviors evaluated in training demand some sort of remedy, some extension, some improvement over prosaic ML terms and ideas, although this text cannot fully meet that need.

5 Conclusion

Policy models are very strong. Value models and Q models are the weakest and most indirect way to get policy models. We are so good at getting policy models we can make 600M parameter models do by accident what 175B parameter models could not 5 years ago: use JSON, and beyond that, follow instructions I suppose. This is a tumultuous and exciting moment in the history of machine learning but possibly also mathematics, as bartosuttonian approaches are relevant only as footnotes or appendices to the strict algorithms producing the most reward-affected, reward-effecting models of actions over states. Somebody ought to extend this research question (of policy models as sets of trajectories, and policy optimization as trajectory->trajectory transport) to describe the policy gradient for states and sequences annotated with reward magnitudes other than 1 and 0. Further,

this work should be extended to identify whether the policy gradient itself can be used to increase the diversity of trajectories used by the policy model, without resorting to auxiliary losses whose analytical relationship to a task's optimal policy cannot be readily determined.

References

[1] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Danny Hernandez, Dustin Howland, Liane Lovitt, Scott Johnston, Tom Brown, Jared Kaplan, Sam McCandlish, Chris Olah, and Dario Amodei. A general language assistant as a laboratory for alignment, 2021.

[2] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Sam Ringer, Jared Kaplan, Tom Brown, Benjamin Mann, and Dario Amodei. Training a helpful and harmless assistant with reinforcement learning from human feedback, 2022.

[3] DeepSeek-AI. Deepseek-math: Pushing the limits of mathematical reasoning in open-source models, 2024.

[4] DeepSeek-AI. Deepseek-v2: A strong, economical, and open-source mixture-of-experts language model, 2024.

[5] Adam Karvonen. Emergent world models and latent variable estimation in chess-playing language models, 2024.

[6] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. D. Van Nostrand Company, 1960.

[7] Zhi-Hao Lin, Jia-Bin Huang, Zhengqin Li, Zhao Dong, Christian Richardt, Tuotuo Li, Michael Zollhöfer, Johannes Kopf, Changil Kim, and Shenlong Wang. Diffusionlight: Light probes for free by painting a chrome ball, 2023.

[8] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[9] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction, 2018.

A Appendix A=A

That which follows is suttoned from the barto of us.

A.1 4.2 Policy Improvement

> "Our reason for computing the value function for a policy is to help find better policies."
>
> "For some state s we would like to know whether or not we should change the policy to deterministically choose an action $a \neq \pi(s)$. "
>
> "—that is, if it is better to select a once in s and thereafter follow π than it would be to follow π all the time—then one would expect it to be better still to select a every time s is encountered, and that the new policy π' would in fact be a better one overall. That this is true is a special case of a general result called the policy improvement theorem."

A.2 8.10 Rollout Algorithms

"the goal of a rollout algorithm is not to estimate a complete optimal action-value function, q_* , or a complete action-value function, q_π, for a given policy π."

"As decision-time planning algorithms, rollout algorithms make immediate use of these action-value estimates, then discard them"

"What then do rollout algorithms accomplish? The policy improvement theorem described in Section 4.2 tells us that given any two policies π and π' that are identical except that $\pi'(s) = a! = \pi(s)$ for some state s, if $q_\pi(s, a) >= v_\pi(s)$, then policy π' is as good as, or better than, π. This applies to rollout algorithms where s is the current state and π is the rollout policy. Averaging the returns of the simulated trajectories produces estimates of$q_\pi(s, a')$ for each action $a' \in A(s)$'

"Then the policy that selects an action in s that maximizes these estimates and thereafter follows π is a good candidate for a policy that improves over π . The result is like one step of the policy-iteration algorithm of dynamic programming discussed in Section 4.3 (though it is more like one step of asynchronous value iteration described in Section 4.5 because it changes the action for just the current state)."

A.3 "surprisingly effective"

"In other words, the aim of a rollout algorithm is to improve upon the rollout policy; not to find an optimal policy. Experience has shown that rollout algorithms can be surprisingly effective."

"Intuition suggests that the better the rollout policy and the more accurate the value estimates, the better the policy produced by a rollout algorithm is likely be (but see Gelly and Silver, 2007)."