Understanding Reasoning from Pretraining to Post-Training

Jingyan Shen[1]* **Ang Li**[1,2]* **Salman Rahman**[1,3] **Yifan Sun**[4]
Micah Goldblum[5] **Matus Telgarsky**[1] **Pavel Izmailov**[1]
[1]New York University [2]Modal Labs
[3]University of California, Los Angeles
[4]University of Illinois Urbana-Champaign [5]Columbia University

🤗 **Models & Datasets**: huggingface.co/pavelslab-nyu/pre2post-chess
Code: github.com/pavelslab-nyu/pre2post-chess

Abstract

Reinforcement learning (RL) improves reasoning in large language models (LLMs), but how pretraining choices shape the returns to RL compute and how RL changes model behavior remain poorly understood. Studying these questions in standard LLMs is challenging because pretraining data are uncontrolled and systematic compute sweeps across pretraining and RL are prohibitively expensive. To address these challenges, we use chess as a controlled testbed for studying reasoning across the pretraining-to-post-training (pre-to-post) pipeline. We follow this pre-to-post pipeline by pretraining language models from 5M to 1B parameters on human chess games, supervised fine-tuning on synthetic reasoning traces, and running RL on chess puzzles with verifiable rewards. Using this framework, we establish a scaling law connecting pretraining and RL: the post-RL performance at given RL compute level is well-predicted from the pretraining loss, and slope of the RL reward curves improves approximately linearly with the pretraining tokens. Beyond scaling, we find that RL does not simply sharpen the SFT policy: on easy puzzles it amplifies correct moves the SFT policy already preferred, while on hard puzzles it surfaces correct moves that were nearly absent under SFT. We further test whether our findings beyond chess by training a 1B language model on math domain, where the same predictive pattern emerges: longer-pretrained checkpoints reach higher post-RL performance and improve faster under RL. In sum, we provide a quantitative account of the pretraining-to-RL interface and a testbed for studying the science of reasoning across the full pre-to-post pipeline.

1 Introduction

Large language models (LLMs) typically follow a pretraining-to-post-training (pre-to-post) pipeline: large-scale pretraining followed by supervised fine-tuning (SFT) and reinforcement learning (RL) with verifiable rewards [11, 18, 37, 40]. As LLMs continue to scale, two perspectives on where to invest additional compute have begun to diverge. One emphasizes the pretrained prior: scaling model size, data, and compute to produce stronger base models from human text [15, 13]. The other emphasizes experience: using RL to learn from environmental interaction and outcome-based feedback, thereby eliciting or developing capabilities beyond direct imitation. This view is reflected in arguments for an *era of experience* [31], and exemplified by AlphaZero, which famously removed the imitation cold-start used by earlier AlphaGo and learned stronger policies from self-play alone [32, 33]. For

*Equal contribution, correspondence to jingyan.s@nyu.edu, al6843@nyu.edu, pi390@nyu.edu

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

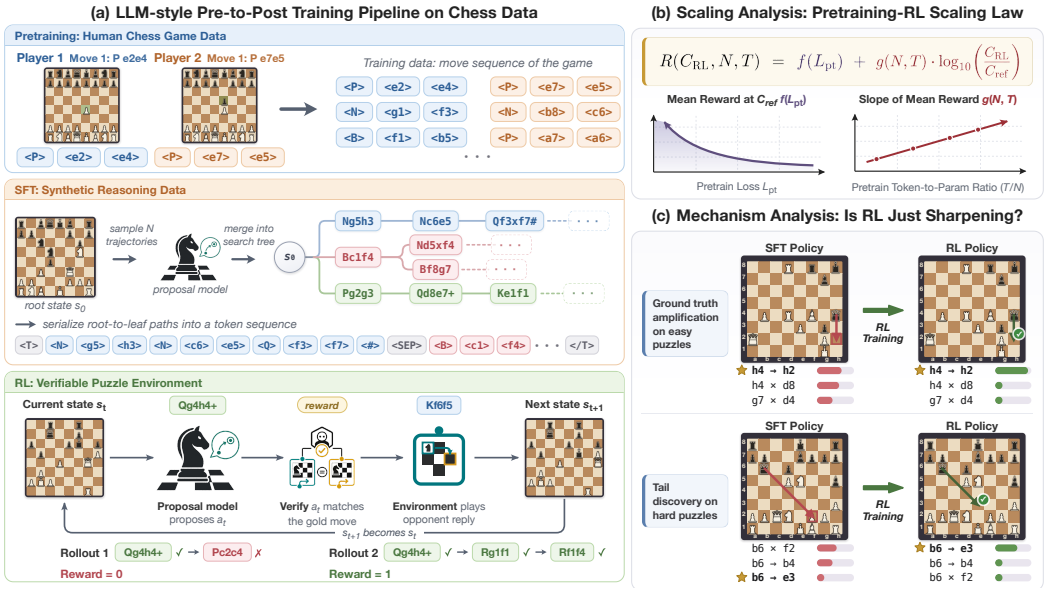


Figure 1: **Overview.** (a) We introduce a synthetic framework for studying reasoning across the pre-to-post pipeline in the chess domain. (b) Using this framework, we establish a joint pretraining–RL scaling law, showing that pretraining performance provides predictive signal for RL performance under a fixed compute budget. (c) Through mechanistic analysis of policy evolution, we show that RL can surface moves that were nearly absent under the SFT policy.

LLM reasoning, an experience-only approach is not yet practical as their action spaces are enormous and rewards for a randomly-initialized policy are initially extremely sparse. Thus, RL is invariably initialized from a pretrained prior, so the relevant question is not whether to use a prior, but how good the prior needs to be. Concretely, how should a fixed compute budget be divided between improving the pretrained model and further optimizing it with RL? Prior work has developed quantitative scaling laws for pretraining [15, 13] and has studied post-training RL scaling and recipes separately [23, 17], but there is no quantitative characterization of how pretraining interacts with RL scaling.

A related question concerns what RL actually does to the pretrained policy it inherits. Recent work points to strikingly different views. Yue et al. [39] argues that RL primarily sharpens reasoning patterns the base model already prefers, observing that base models match or exceed RL-tuned ones in pass@k when k is large. Yuan et al. [38] argues the opposite, showing that RL composes pretrained skills into new ones. Sun et al. [34] reports both behaviors, with "grokking" transitions on some problems and outright failures on others. These competing views have direct implications for compute allocation: if RL mostly sharpens, we should invest more compute in pretraining; if RL can genuinely discover, we should invest more in RL.

Studying these questions directly in reasoning LLMs trained on natural language is challenging. Systematic sweeps over pretraining and RL compute for language models are prohibitively expensive at frontier scale, while massive and heterogeneous pretraining corpora make it hard to attribute behaviors to pretraining versus RL. Moreover, evaluation typically provides only final-answer correctness, leaving the policy's behavior at individual reasoning steps largely opaque. These obstacles make it difficult to isolate how pretraining and RL interact.

Therefore, we use chess as a controlled testbed for studying these questions, with a training pipeline designed to mirror standard LLM training: pretraining on tokenized sequences of human game moves, supervised fine-tuning on synthetic reasoning traces, and RL with verifiable rewards. Chess has a compact, explicit action space, and move quality can be verified exactly via game outcomes or strong engines, providing ground truth at each step of a reasoning trajectory. Human play data is plentiful and controllable: we can vary the amount, quality, and composition of the pretraining corpus (e.g., by filtering on player Elo) without making complex data-mixture choices. Task-specialized small models can already reach nontrivial chess performance [27, 44, 33], making compute sweeps across pretraining and RL both affordable and informative: the models are sensitive enough for changes in scale and RL compute to produce measurable differences in performance and policy behavior. Our

goal is not to build the strongest chess model, but to use chess as a tractable setup for isolating how pretraining scale and verifiable RL interact.

Using this framework (Fig. 1), we pretrain models from 5M to 1B parameters and sweep 36 pretraining–RL combinations. Our contributions are summarized as follows:

- **A controlled chess testbed for pre-to-post studies.** We instantiate the standard pre-to-post pipeline in chess, with pretraining on human games, SFT on reasoning traces, and RL with verifiable rewards. This setting enables systematic sweeps over pretraining and RL compute, and detailed analysis of policy reasoning traces (§ 2).
- **A joint pretraining–RL scaling law.** We find that pretraining loss predicts the post-RL downstream performance, measured by pass@1, while the local RL slope grows approximately linearly with log pretraining token count. Combining this with a Chinchilla-style pretraining scaling law lets us score hypothetical recipes defined by number of model parameters N, pretraining tokens T, and the amount of compute C_{RL} allocated to reinforcement learning. We trace a compute-optimal frontier, and find that the optimal allocation shifts toward a larger RL fraction as total compute grows (§ 3).
- **Mechanism of RL policy change.** We analyze policy evolution and reasoning dynamics. We find that on easy puzzles, RL primarily amplifies correct moves the SFT policy already preferred; on hard puzzles, it can surface correct moves that were nearly absent, but also reinforces incorrect moves. These heterogeneous effects connect to the observation that RL improves pass@1 without consistently improving pass@k (§ 4).
- **Evidence of transfer beyond chess.** Across checkpoints of a fixed 1B language model pretrained on 10B–200B tokens of math-domain text, longer pretraining is associated with higher performance at fixed RL compute and a steeper local RL scaling slope (§ 5).

2 Framework: Chess as a Testbed for Reasoning

Our testbed, illustrated in Fig. 1, mirrors the standard pre-to-post pipeline and consists of three components: pretraining on a large-scale corpus of human games (§2.1), a synthetic reasoning-trace generator for supervised fine-tuning (§2.2), and RL on a verifiable chess-puzzle environment (§2.3).

Chess Representation. Following prior work [44], we represent each chess game as an alternating sequence of the two players' moves serialized into tokens. Drawing on the SAN and UCI conventions, we encode each move with four tokens, $\langle\texttt{piece}\rangle$; $\langle\texttt{source}\rangle$; $\langle\texttt{destination}\rangle$; $\langle\texttt{flag}\rangle$, where $\langle\texttt{flag}\rangle$ marks special cases such as promotion, castling, en passant, check, or checkmate. Any valid prefix of the resulting sequence determines a unique board state, and the full vocabulary has size $|\mathcal{V}| = 81$. Examples are provided in Fig. 1.

2.1 Pretraining on Human Game Trajectories

In the pretraining stage, the model learns the distribution of plausible move sequences from large-scale human play. We collect game trajectories from Lichess[2], yielding a corpus that spans a wide range of player strengths and game outcomes. The corpus can be subsampled along axes such as player Elo and game length, giving fine-grained control over data composition. On this corpus, we train an autoregressive policy over tokenized games using the standard next-token prediction objective.

2.2 Supervised Fine-Tuning with Synthetic Reasoning Traces

In post-training, we train the model on chess puzzles: the solver must select the unique best move at each step, and each move can be verified exactly against the ground-truth solution line.

Verifiable chess puzzle environment. As shown in Fig. 1, each puzzle specifies an initial board state s_0 together with a ground-truth solution line, formulating puzzle solving as a multi-step interactive decision problem in which the model acts only as the solver, rather than generating moves for both sides. At each step t, the model observes the current solver state s_t and outputs a candidate move a_t. The environment then checks a_t against the ground-truth best move: a mismatch terminates the episode immediately, while a match either completes the puzzle or triggers the corresponding

[2] `https://database.lichess.org/`

opponent response, producing the next solver state s_{t+1}. The model thus plays one move at a time: on a correct move, the environment appends the opponent's reply o_t to the context, and the model conditions on it when generating a_{t+1}. Following Ruoss et al. [27], a trajectory is considered successful only if *the model selects the correct solver move at every step* and thereby completes the full solution line. We present an example puzzle game in Fig. 7.

During pretraining, the model only observes move sequences and does not see explicit reasoning traces. Prior chess models have augmented these base policies with inference-time search algorithms such as MCTS and beam search to recover planning behavior at test time [33, 44]. We instead pursue an approach that mirrors how language models reason: rather than attaching an external search procedure, we elicit reasoning in context, training the model to produce its own reasoning trace before committing to a move. We emphasize that for our models, the reasoning happens in chess move tokens, without natural language.

Synthetic reasoning trace construction. Motivated by prior work on CoT generation [20], we synthesize the CoT as a serialization of possible game continuations (Fig. 1). Intuitively, the reasoning trace is a set of plausible game continuations sampled from the pretrained model, slightly reordered to follow a tree-traversal structure. Given an input board s_0, we sample K continuations $\tau_1, \ldots, \tau_K$ from a proposal policy. Because these continuations share opening moves, we merge them by common prefixes into a tree of positions rooted at s_0, storing each shared prefix once. The tree has $m \leq K$ leaves, one per distinct continuation, and we write $\tilde{\tau}_1, \ldots, \tilde{\tau}_m$ for the corresponding root-to-leaf paths. We serialize the tree in depth-first order to form the reasoning trace: $r =$ <T> $\tilde{\tau}_1$ <sep> $\tilde{\tau}_2$ <sep> $\cdots$ $\tilde{\tau}_m$ </T>, where <sep> separates consecutive paths, and <T></T> are the thinking tags delimiting the reasoning. The proposal policy $p_{\theta_{\text{prop}}}$ is itself a pretrained model from Section 2.1, so the resulting traces remain close to the model's own distribution rather than coming from an external search procedure. We provide an example trace in Table 4 and full construction details in Appendix C.6. After producing the synthetic reasoning trace r, we train the model to commit to the best solution continuation $\tau^\star$ from $\{\tilde{\tau}_1, \tilde{\tau}_2, \ldots, \tilde{\tau}_m\}$. We train on the concatenated sequence $w = (r, \tau^\star)$. Here, $\tau^\star = (a_1, o_1, a_2, o_2, \ldots, a_H)$ consists of the player's moves and the environment moves. Since opponent moves are provided by the environment at inference time, we mask opponent-move tokens from the loss and train only on reasoning r and the model moves.

2.3 Reinforcement Learning with Verifiable Rewards

Starting from the SFT policy, we optimize the model on the puzzle environment with a binary outcome reward $R(\zeta, s_0) = \mathbf{1}[a_1 = a_1^\star, \ldots, a_H = a_H^\star]$, where ζ is the full trajectory (reasoning trace followed by the executed move sequence) and $(a_1^\star, \ldots, a_H^\star)$ is the ground-truth solution line. In words, the model receives reward 1 only if every executed move matches the corresponding ground-truth move, and 0 otherwise, so a single mistake anywhere in the line yields no reward. We optimize the policy with Group Relative Policy Optimization (GRPO) [29]; algorithm details are in Appendix C.3.

2.4 Experimental Setup

We collect a 54B-token pretraining corpus of Blitz and Rapid human games played on Lichess in 2022, from which our scaling sweeps draw varying token budgets. For post-training, we use 156K quality-filtered Lichess puzzles, spanning five difficulty bins (B1–B5, from easiest to hardest). For evaluation, we curate a benchmark of 1,480 tactical puzzles spanning the same difficulty bins, balanced for theme diversity and solution length. Since current models rarely solve B5 puzzles, all aggregate pass@k results in the paper are reported over B1–B4, and we retain B5 for the difficulty-stratified mechanism analysis in Section 4.1. The three datasets are mutually disjoint at the board-position level to prevent contamination. All models use the dense Qwen3 [35] base architecture, trained at 10 scales: $\{5\text{M}, 10\text{M}, 20\text{M}, 32\text{M}, 50\text{M}, 100\text{M}, 200\text{M}, 410\text{M}, 680\text{M}, 1\text{B}\}$. Full details on datasets, reasoning trace construction, model architectures, and training configurations are provided in Appendix C.

3 Pre-to-Post Scaling Analysis

We begin by analyzing pre-RL scaling behavior in our chess setup, then study how pretraining choices interact with RL scaling through two questions: **(RQ1)** At different total compute levels, what final

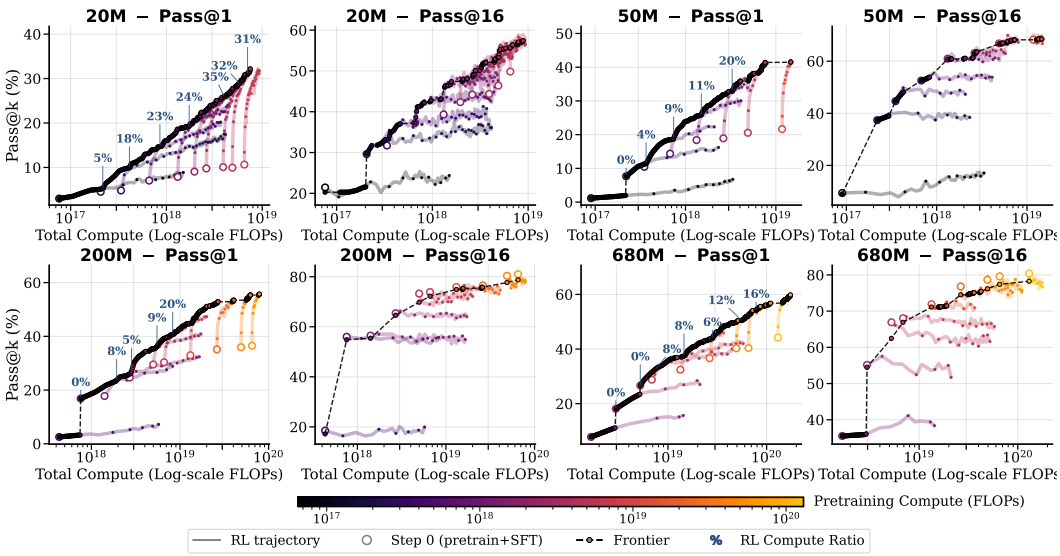


Figure 2: **Empirical frontier of puzzle benchmark performance across pretraining-RL sweeps for four model sizes (20M, 50M, 200M, 680M).** Each point is a checkpoint evaluated on the puzzle benchmark. Color indicates pretraining compute; open rings mark the pre-RL baseline. Black curves mark the Pareto frontier. For the pass@1 curves, we label the fraction of RL compute at which each run first reaches the frontier (excluding the smallest and largest compute ranges). For pass@1, RL consistently improves performance and the labeled RL ratio shows an increasing trend as total compute increases across models. For pass@16, gains from RL are smaller.

performance frontier is induced by different allocations between pretraining and RL? **(RQ2)** Can pretraining properties, such as model size, number of pretraining tokens, and pretraining loss, predict RL scaling behavior in a given compute regime?

3.1 Pre-RL Analysis: Scaling Behavior Before RL

We sweep 11 pretraining compute budgets from 6.5×10^{16} to 6.5×10^{19} FLOPs across 10 model sizes, corresponding to training runs of approximately 200M to 52B tokens. Following the methodology in prior scaling-law studies [13, 26], we report IsoFLOP curves for validation loss on held-out human games, along with pass@1 and pass@16 on the downstream puzzle benchmark in Fig. 8. The results show that, within each model size, downstream benchmark performance continues to improve with additional pretraining over the compute range we study. However, under fixed FLOPs, an optimal parameter-token allocation exists, and this optimum closely tracks validation loss on human games, pass@1, and pass@16. A functional-form fit of the Chinchilla law is provided in Appendix E.

For SFT, we compare training on the target move sequence alone against training on synthetic reasoning traces followed by the target answer, using the same number of puzzle samples across models (Appendix F). Fig. 9 compare the two settings across model sizes and pretraining FLOPs. SFT without reasoning traces improves pass@1 at every scale. At larger scales, however, its pass@16 gain over the pretrained baseline shrinks, while SFT with reasoning traces gives a higher pass@16, especially on harder puzzles. Because SFT with reasoning traces lets larger models sample a broader set of correct solutions, we adopt it for all subsequent RL experiments, using it at every model size for consistency. Moreover, for a fixed model size, stronger pretrained checkpoints consistently achieve higher post-SFT performance, and this ordering is preserved across the compute range we study. This suggests that additional pretraining provides stronger pre-RL initializations.

3.2 RQ1: What Is the Pretraining-RL Compute Tradeoff?

Under a fixed total compute budget and model size N, pretraining on more tokens (larger T) improves the initialization but leaves less compute for RL. For each model size and total budget, we evaluate the final performance obtained from each available checkpoint after spending the remaining compute on RL, and define the fixed-budget frontier as the best result over these choices. A frontier point

selected at an earlier checkpoint (smaller T) indicates that additional RL compute is more valuable than further pretraining, whereas a point selected at larger T indicates that the initialization with longer pretraining is worth the reduction in RL compute. We evaluate the frontier for 4 model sizes using pretrained checkpoints from the IsoFLOP sweeps in Section 3.1. For each model size in $\{20\text{M}, 50\text{M}, 200\text{M}, 680\text{M}\}$, we select checkpoints spanning 8-11 pretraining compute levels, apply the fixed SFT recipe, and perform RL from each resulting SFT policy for 1000 to 5000 steps[3]. Our primary analysis measures compute in FLOPs[4], using the estimation derived in Appendix D.

The results are shown in Fig. 2. Across all training runs, RL substantially improves pass@1 performance, and pass@1 continues to increase with additional RL training. We also mark the Pareto frontier across all model sizes and highlight the RL compute fraction at each frontier point. For a fixed model, the frontier initially selects a high pretraining fraction (i.e., low RL fraction), suggesting that RL is strongly initialization-limited: under the same total budget, the extra RL compute from starting earlier does not compensate for the weaker pretrained policy. However, the pretraining fraction decreases as the budget grows, indicating diminishing marginal downstream returns from additional pretraining. Once the initialization is sufficiently strong, a larger fraction of compute is better spent on RL.

In contrast, pass@16 exhibits more mixed behavior. For the smallest 20M model, pass@16 improves sharply at the beginning of RL training and then increases more gradually. For larger models, however, the pass@16 curves remain nearly flat, and in some cases even degrade slightly with additional RL training. In this regime, additional pretraining can lead to higher pass@16 performance than allocating the same compute to more RL. This limited pass@16 improvement from RL is consistent with prior findings [39]. We further analyze policy change through fine-grained categorization in Section 4.1, providing additional insight into the mixed effects of RL.

3.3 RQ2: Can Pretraining Properties Predict RL Scaling Behavior?

Motivated by the frontier behavior above, we further study whether the observed learning trends can be parameterized by a functional form. Prior work [17] proposes an RL scaling law that models the performance of a fixed model as a sigmoid function of RL compute (see Eq. 5 in Appendix G.1), with the upper asymptote representing the RL ceiling: the performance the model would reach with unlimited RL compute. However, the sigmoid only plateaus after very long RL training, so its ceiling cannot be identified reliably without runs long enough to reach that saturation. Under our fixed total compute budgets, most RL runs instead cover only the early, non-saturated part of the curve. We therefore take a first-order Taylor expansion of the sigmoid law in the non-saturated RL regime, writing the local RL scaling fit as

$$R_{N,T}(C) = R^{\text{ref}}_{N,T} + B_{N,T}(\log_{10} C - \log_{10} C_{\text{ref}}), \tag{1}$$

with derivations in Appendix G.1. Eq. (1) is log-linear in RL compute: anywhere along the fitted line, each $10\times$ increase in RL compute adds the same fixed increment $B_{N,T}$ to the reward.

Fitting procedure. For each RL run obtained in Section 3.2, we collect the pass@1 metric on the downstream benchmark after C FLOPs of RL compute as samples of $R_{N,T}(C)$ for different values of C. We fit per-run coefficients $(R^{\text{ref}}_{N,T}, B_{N,T})$ using ordinary least squares to the observations $(x, y) = (\log_{10} C - \log_{10} C_{\text{ref}}, R_{N,T}(C))$:

$$(R^{\text{ref}}_{N,T}, B_{N,T}) = \arg\min_{a,b} \frac{1}{N_C} \sum_{i=1}^{N_C} \big(R_{N,T}(C_i) - a - b(\log_{10} C_i - \log_{10} C_{\text{ref}})\big)^2,$$

where N_C is the number of observations. The reference compute level C_{ref} defines a linear shift in the features of our linear fit, and does not affect the quality of the fit itself or the slope parameter $B_{N,T}$; it does affect the reference reward parameter $R^{\text{ref}}_{N,T}$, which should be interpreted as the fitted reward at a reference compute level C_{ref}. We then ask whether the parameters $\phi_{N,T} = \{R^{\text{ref}}_{N,T}, B_{N,T}\}$ can

[3]For reference, 2000 RL steps take approximately 160 H200 GPU-hours for a 50M model.

[4]We note that equal FLOPs do not necessarily translate to equal wall-clock time across training stages. While prior work often measures RL compute using wall time or training steps [17], we use FLOPs as the primary allocation unit to isolate algorithmic compute. Wall time additionally reflects stage-specific systems factors that can differ substantially between pretraining and RL. Thus, wall time is important for measuring realized cost, but is a noisier unit for studying the algorithmic compute-allocation tradeoff between pretraining and RL.

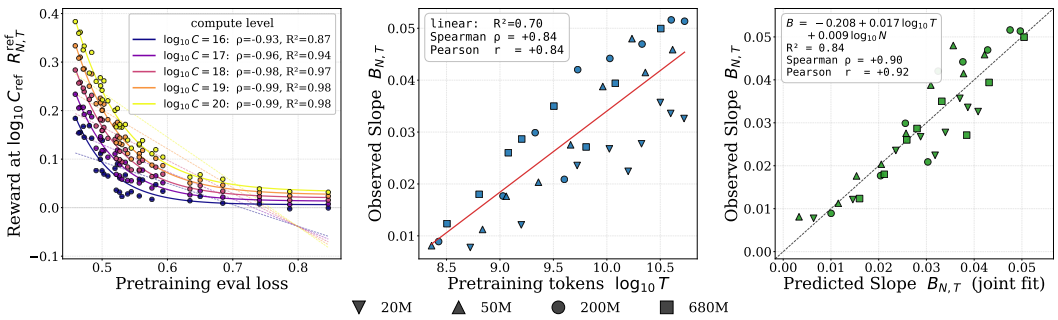


Figure 3: **Pretraining properties predict local RL scaling behavior.** (a) $R^{\text{ref}}_{N,T}$, the fitted post-RL pass@1 at a reference RL compute level, versus pretraining validation loss. Each curve corresponds to a different reference compute level; the fit tightens as RL compute increases (ρ=-0.93 to -0.99). (b) $B_{N,T}$, the local slope, versus pretraining tokens. More tokens predict faster RL improvement. (c) Joint fit of the slope using both $\log T$ and $\log N$, achieving higher R^2.

be predicted from properties of the pretrained checkpoint, namely the model size N, the number of pretraining tokens T, and the pretraining validation loss $L_{\text{pt}}(N, T)$. In this analysis, we evaluate only on the B3–B4 benchmark with intermediate difficulty, where the models we consider do not saturate the evaluation.

Connecting pretraining to the fitted log-linear law. Fig. 3 (left) shows representative fits relating the post-RL performance to pretraining loss. Since our RL compute spans 10^{16} to 10^{20}, we evaluate different choices of C_{ref} within this range. The pass@1 performance at a given RL compute level, $R^{\text{ref}}_{N,T}$, is strongly predicted by pretraining validation loss: across choices of C_{ref}, lower pretraining loss consistently corresponds to higher post-RL performance, and the relationship becomes increasingly monotone at larger reference compute C_{ref}. For example, the Spearman correlation increases from $|\rho| = 0.93$ to $|\rho| = 0.99$ as $\log_{10} C_{\text{ref}}$ increases from 16 to 20. We thus fix $C_{\text{ref}} = 10^{20}$ FLOPs as a constant across all runs (Appendix G.4). In Appendix G.6, we additionally analyze the sigmoid law fit on the 20M model runs (the only model size where the fit is sufficiently determined by data), where it shows that the asymptotic performance ceiling is similarly predictable from pretraining loss (see Fig 19).

As presented in Fig. 3 (middle), the per-run slope parameter $B_{N,T}$ shows a positive linear correlation with pretraining tokens $\log_{10} T$ (Pearson r=$+0.84$). We therefore fit $B_{N,T}$ against $\log_{10} T$ alone, tokens per parameter $\log_{10}(T/N)$, and a joint linear model using both $\log_{10} T$ and $\log_{10} N$. Among these, the joint model attains the lowest RMSE and highest R^2 in predicting the observed slope, so we adopt it as our parameterization of $g(N, T) \approx B_{N,T}$. As shown in Fig. 3 (right), the joint fit assigns a larger coefficient to $\log_{10} T$ than to $\log_{10} N$. Together, these observations indicate that the RL improvement rate is largely shaped by the amount of pretraining data exposure, with model size providing a weaker positive correction. We compare slope parameterizations in Fig. 12 and report fits on other benchmark subsets in Figs. 13 and 14. However, we also observe failure cases when stronger models approach saturation on easy benchmarks, systematically compressing their local slope estimates. We interpret the observed association as a local empirical trend over the unsaturated compute range, rather than a global relationship across all training regimes.

Jointly, these observations motivate the following scaling law. We report the full statistical analysis, including parameterization of f and g, in Appendix G. Leave-one-out validation on our existing runs shows that the fitted parameterization predicts held-out RL trajectories well when the observed pretraining loss is provided (Fig. 16).

Joint Pretraining–RL Scaling Law

$$R(C_{\text{RL}}, N, T) = f(L_{\text{pt}}(N, T)) + g(N, T)\,(\log_{10} C_{\text{RL}} - \log_{10} C_{\text{ref}})$$

Extrapolating the compute-optimal frontier. Combining our joint pretraining-RL law with the Chinchilla loss prediction $L(N, T)$ allows us to evaluate hypothetical training recipes (N, T, C_{RL}) without actually training them. We consider a ladder of 13 model sizes from 20M to 2B parameters. For each size, we evaluate 260 total-compute budgets between 10^{17} and 10^{21} FLOPs. For each model size N and total-compute budget C, we search over 400 candidate splits of the budget between

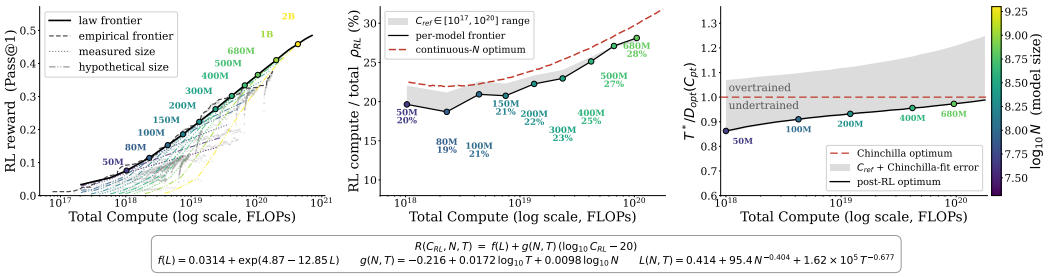


Figure 4: **Extrapolated compute-optimal frontier across model sizes using the fitted law. (a)** Simulated compute-optimal reward ($R_N^\star(C)$) for each model size. The black curve is the global simulated frontier, dots mark model-size transitions, and the dashed gray curve is the empirical frontier. The two frontiers closely agree. **(b)** Compute-optimal RL share ($\rho_{\mathrm{RL}} = C_{\mathrm{RL}}^\star / C_{\mathrm{total}}$). The optimal RL fraction increases with total compute, from approximately 20% at 50M to 28% at 680M; the red dashed curve shows the continuous-(N) optimum. **(c)** Optimal pretraining tokens relative to the Chinchilla allocation at the same pretraining compute.

pretraining and RL, each a choice of (T, C_{RL}). We then predict the pretraining loss $L_{\mathrm{pt}}(N, T)$ using the Chinchilla law and estimate the post-RL rewards $R(C_{\mathrm{RL}}, N, T)$ using our joint law. Based on the grid search, we identify the best predicted reward $R_N^\star(C)$ achievable by model size N at budget C. The global frontier is then obtained by maximizing $R_N^\star(C)$ over model sizes at each budget C. See Appendix G.7 for more details.

The resulting frontier is shown in Fig. 4. For each model size, we locate the budget at which it first attains the global fitted frontier and report the corresponding RL-compute share. We also compute the pretraining token count $T^\star$ selected at that frontier point and compare it to the Chinchilla-optimal token count under the same pretraining compute budget, $D_{\mathrm{opt}}(C_{\mathrm{pt}})$ tokens. The results indicate that the compute-optimal RL share increases with total compute, while the corresponding pretraining token allocation does not significantly deviate from Chinchilla scaling.

4 Mechanism Analysis: Policy Evolution During RL Post-training

In this section, we study how RL training reshapes the policy beyond its pre-RL prior. Prior work disagrees on whether post-training primarily elicits capabilities already present in the base model [39] or induces qualitatively new behaviors [34, 38]. Our framework provides direct access to both the policy distribution over moves at each state and the structured reasoning traces, enabling us to study policy evolution along two dimensions: **(RQ1):** How does RL reshape the move distribution at each state? **(RQ2):** How does RL reshape the chain-of-thought reasoning that produces those moves?

4.1 RQ1: How Does RL Change the Move Policy?

For any puzzle board state s, we define the induced move policy π_θ as a probability distribution over the legal move set $\mathcal{A}(s)$. For the pretraining model, the unnormalized score for move a is $\widetilde{\pi}_{\theta_{\mathrm{pre}}}(a \mid s) = \prod_{j=1}^{|\tau(a)|} p_{\theta_{\mathrm{pre}}}(x_j^{(a)} \mid s, x_{<j}^{(a)})$, where $\tau(a)$ is the token serialization of the move. For SFT and RL models, the move score should be marginalized across reasoning traces: $\widetilde{\pi}_\theta(a \mid s) = \sum_r \pi_\theta(r)\, \widetilde{\pi}_\theta(a \mid s, r)$, where $\widetilde{\pi}_\theta(a \mid s, r)$ is obtained by scoring the token serialization of move a conditioned on trace r. The induced move policy $\pi_\theta(a \mid s)$ is then obtained by normalizing $\widetilde{\pi}_\theta(a \mid s)$ over the legal move set $\mathcal{A}(s)$. We sample 128 reasoning traces per puzzle to evaluate the marginal distribution and provide more details in the Appendix H.

RL is not well explained by a uniform temperature scaling of SFT. Inspired by prior work [16], we test whether the RL policy is a power transformation of the SFT policy, $\pi_{\mathrm{RL}}(a \mid s) \propto \pi_{\mathrm{SFT}}(a \mid s)^\alpha$, where $\alpha > 1$ corresponds to sharpening. Exact power scaling implies that centered RL log-probabilities are linear in centered SFT log-probabilities with slope α, so we fit this slope by zero-intercept regression, both globally and per state (Appendix H.2). Table 13 shows the fitted global slope increases during RL, indicating that RL sharpens the SFT policy on average. However, the fit achieves only moderate R^2 and the per-state slopes vary substantially, indicating that RL reshapes the policy in state-dependent ways.

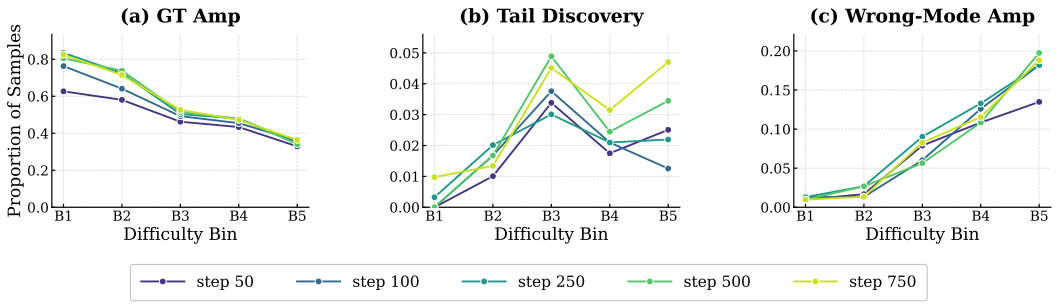


Figure 5: **RL reshapes the move policy in qualitatively different ways across puzzle difficulty.** Each panel shows the proportion of puzzle states assigned to one policy-update category (Table 14) across difficulty bins B1-B5. On easy puzzles, ground-truth amplification (a) dominates. On harder puzzles, tail discovery (b) and wrong-mode amplification (c) both increase, showing that RL simultaneously surfaces previously absent correct moves and reinforces incorrect ones.

We therefore investigate *how probability mass is redistributed* at individual states. We categorize local policy changes by how the ground-truth move transitions between the top-k sets of the initial policy π_{θ_0} and the updated policy π_{θ_1}. Table 14 gives the formal definitions for the full categorization. In Fig. 5, we highlight three major categories of policy changes: (1) *Ground-truth amplification*: the correct move is already in the top-k set and is further reinforced by training; (2) *Tail discovery*: training promotes a correct move from the low-probability tail, defined as probability below $\epsilon_{\text{tail}} = 0.05$, into the top-$k$ set; (3) *Wrong-mode amplification*: the correct move remains outside the top-k set, while the initially preferred wrong move is further reinforced. We set $k = 3$ in practice and present additional results in Appendix H.3.

On easier puzzles RL training mostly amplifies correct moves the SFT policy already preferred, while on harder puzzles it both surfaces moves that were nearly absent under SFT and reinforces incorrect ones. In our experiments, we instantiate θ_0 as the SFT policy and $\theta_1 = \theta_t$ as the RL policy at training step t. Fig. 5 shows the resulting category proportions across B1-B5 puzzle test sets. Fig. 21 presents a qualitative example of how RL recovers the ground-truth move from the tail distribution for a hard puzzle on the B5 test set.

4.2 RQ2: How Does RL Change the Dynamics of Chain-of-Thought Reasoning?

As a complementary analysis, we examine how the structure of reasoning traces evolves over the course of RL training. Because our CoT format comprises explicit move sequences, each rollout can be reconstructed as a prefix tree rooted at the puzzle state (Section 2.2), with each node corresponding to a move, enabling us to probe both the structure and quality of the model's reasoning. In Fig. 22, Fig. 23 and Fig. 25, we compare two representative RL runs for the 20M and 50M models pretrained under matched compute, in terms of reasoning tree structure, move quality and search behavior.

Reasoning trace quality improves during RL, but deeper search remains challenging. The parsed traces show that models primarily expand search breadth rather than depth: the width-to-depth ratio and branching factor increase while maximum search depth stays roughly flat. Meanwhile, the quality of proposed moves improves for both the model's own moves and its predicted opponent responses, and the model becomes more likely to surface the ground-truth move in its CoT and commit to the best candidate it has considered. However, Fig. 25 shows that the model still struggles to recover continuations requiring more than 5 moves, suggesting that RL improves candidate generation and selection faster than long-horizon search. These structured search features may guide future SFT data construction toward examples that encourage deeper, more systematic search.

5 Transfer to Text: A Qualitative Case Study in Math

To test whether the scaling law identified in our chess setting also transfers to natural language modeling, we pretrain a 1B-parameter OLMo-2 [22] model on a 200B-token mixed pretraining corpus consisting of 70% Nemotron-CC-Math-v1[5] [21] and 30% Dolma3, the mid-training corpus

[5] `https://huggingface.co/datasets/nvidia/Nemotron-CC-Math-v1`

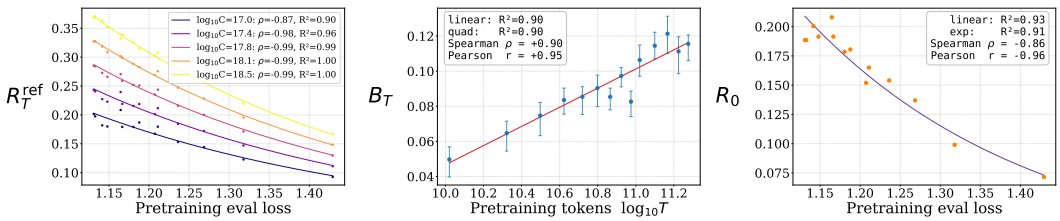


Figure 6: **The predictive pattern extends to the math domain.** Across 1B OLMo-2 checkpoints from 10B to 200B pretraining tokens, lower pretraining loss consistently predicts higher post-RL performance R_T^{ref}, with the fit tightening as RL compute increases. The slope B_T increases nearly linearly with $\log_{10} T$. The R_T^{ref} relationship with pretraining loss becomes tighter as RL compute increases, mirroring the chess results (Fig. 3).

used for OLMo-3 [23]. We report pass@1 on a held-out set of 500 problems drawn from the RL training distribution, as well as the GSM8K [7] and MATH [12] test sets. Full pretraining, SFT, RL, and evaluation settings are provided in Appendix I.

We report the fitting on the held-out set in Fig. 6, and additional results in Fig. 27. Overall, the experiment provides early evidence that the pre-to-post scaling structure identified in chess extends to modern language model training. Despite differences in task format, data distribution, and training recipe, we find a similar pattern: the post-RL performance level at high RL compute is well-predicted from the pretraining loss, and the slope of the RL reward curves improves approximately linearly with the pretraining tokens.

6 Related Work

Reinforcement Learning for Reasoning. RL with verifiable rewards improves LLM reasoning [11, 18, 37, 40], but whether it primarily amplifies existing capabilities or induces new behaviors remains debated [39, 38, 34, 42]. Our chess setting lets us track move probabilities and structured reasoning traces within a constrained action space, distinguishing amplification of preferred moves, discovery of low-probability correct moves, and reinforcement of incorrect moves across puzzle difficulties.

Pre-to-Post Scaling Laws. Scaling laws have long been used as quantitative tools to predict language-model performance from training compute. Prior work characterizes pretraining scaling [15, 13, 10, 26, 28, 25], SFT scaling [41] and RL scaling [17, 6]. However, these works either treat downstream use as given or treat the pretrained initialization as fixed. Coverage-based analyses link next-token prediction to Best-of-N and pass@k performance through the probability mass assigned to high-quality responses [14, 5]. We complement these accounts empirically by predicting post-RL performance and local RL scaling slopes from pretraining properties.

Allocating Compute between Pretraining and RL. Qi et al. [24] study tradeoffs across pretraining, continued pretraining, SFT, and RL, finding that post-training gains can saturate as compute increases. Concurrent work on RL excursions [3] explores early RL during pretraining. We complement these studies of stage-wise effects and training recipes with a joint scaling law for allocating a fixed compute budget between pretraining and RL. An extended discussion can be found in Appendix B.

7 Conclusions

We used chess as a testbed for studying how pretraining influences RL dynamics and how RL reshapes the inherited policy. We established a joint scaling law connecting pretraining properties to RL scaling. The resulting compute-allocation frontier suggests that as the total budget grows, the optimal pretraining fraction tends to decrease, indicating that RL should take an increasingly large share of compute. Meanwhile, RL remains initialization-dependent and requires sufficient pretraining exposure before it becomes effective. The behavior also differs between pass@1 and pass@16. Mechanistically, RL is not uniform sharpening: its effect varies systematically with puzzle difficulty. Finally, we observe a similar qualitative pattern in the math domain on a 1B language model, suggesting our findings extend beyond chess.

Acknowledgement

We thank Vatsal Baherwani, Sean McLeish, Vadim Bereznyuk, Timur Garipov, Yulin Chen and Jens Tuyls for their insightful feedback. This research was supported by a Modal for Academics compute grant. This work was also supported in part by NYU IT High Performance Computing resources, services, and staff expertise.

References

[1] Syeda Nahida Akter, Shrimai Prabhumoye, Eric Nyberg, Mostofa Patwary, Mohammad Shoeybi, Yejin Choi, and Bryan Catanzaro. Front-loading reasoning: The synergy between pretraining and post-training data. *arXiv preprint arXiv:2510.03264*, 2025.

[2] Mordecai Avriel. *Nonlinear programming: analysis and methods*. Courier Corporation, 2003.

[3] Rachit Bansal, Clara Mohri, Tian Qin, David Alvarez-Melis, and Sham Kakade. Rl excursions during pre-training: Re-examining policy optimization for llm training. *arXiv preprint arXiv:2606.04272*, 2026.

[4] Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023.

[5] Fan Chen, Audrey Huang, Noah Golowich, Sadhika Malladi, Adam Block, Jordan T Ash, Akshay Krishnamurthy, and Dylan J Foster. The coverage principle: How pre-training enables post-training. *arXiv preprint arXiv:2510.15020*, 2025.

[6] Zhoujun Cheng, Yutao Xie, Yuxiao Qu, Amrith Setlur, Shibo Hao, Varad Pimpalkhute, Tongtong Liang, Feng Yao, Zhengzhong Liu, Eric Xing, et al. Isocompute playbook: Optimally scaling sampling compute for llm rl. *arXiv preprint arXiv:2603.12151*, 2026.

[7] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[8] Lucas Dionisopoulos, Nicklas Majamaki, and Prithviraj Ammanabrolu. How reasoning evolves from post-training data in sequential decision-making domains.

[9] Xidong Feng, Yicheng Luo, Ziyan Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. Chessgpt: Bridging policy learning and language modeling. *Advances in Neural Information Processing Systems*, 36:7216–7262, 2023.

[10] Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, et al. Language models scale reliably with over-training and on downstream tasks. In *International Conference on Learning Representations*, volume 2025, pages 67661–67682, 2025.

[11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[12] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

[13] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

[14] Audrey Huang, Adam Block, Qinghua Liu, Nan Jiang, Akshay Krishnamurthy, and Dylan J Foster. Is best-of-n the best of them? coverage, scaling, and optimality in inference-time alignment. *arXiv preprint arXiv:2503.21878*, 2025.

[15] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[16] Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think. *arXiv preprint arXiv:2510.14901*, 2025.

[17] Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for llms. *arXiv preprint arXiv:2510.13786*, 2025.

[18] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

[19] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning*, 2024.

[20] Jieyi Long. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023.

[21] Rabeeh Karimi Mahabadi, Sanjeev Satheesh, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc-math: A 133 billion-token-scale high quality math pretraining dataset. *arXiv preprint arXiv:2508.15096*, 2025.

[22] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

[23] Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

[24] Zhenting Qi, Fan Nie, Alexandre Alahi, James Zou, Himabindu Lakkaraju, Yilun Du, Eric Xing, Sham Kakade, and Hanlin Zhang. Evolm: In search of lost language model training dynamics. *arXiv preprint arXiv:2506.16029*, 2025.

[25] Zeyu Qin, Qingxiu Dong, Xingxing Zhang, Li Dong, Xiaolong Huang, Ziyi Yang, Mahmoud Khademi, Dongdong Zhang, Hany Hassan Awadalla, Yi R Fung, et al. Scaling laws of synthetic data for language models. *arXiv preprint arXiv:2503.19551*, 2025.

[26] Nicholas Roberts, Sungjun Cho, Zhiqi Gao, Tzu-Heng Huang, Albert Wu, Gabriel Orlanski, Avi Trost, Kelly Buchanan, Aws Albarghouthi, and Frederic Sala. Test-time scaling makes overtraining compute-optimal. *arXiv preprint arXiv:2604.01411*, 2026.

[27] Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li K Wenliang, Elliot Catt, John Reid, Cannada A Lewis, Joel Veness, and Tim Genewein. Amortized planning with large-scale transformers: A case study on chess. *Advances in Neural Information Processing Systems*, 37:65765–65790, 2024.

[28] Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: accounting for inference in language model scaling laws. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[29] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[30] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024.

[31] David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1:11, 2025.

[32] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[33] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[34] Yiyou Sun, Yuhan Cao, Pohao Huang, Haoyue Bai, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. Rl grokking recipe: How does rl unlock and transfer new algorithms in llms? *arXiv preprint arXiv:2509.21016*, 2025.

[35] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[36] Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. *arXiv preprint arXiv:2407.20311*, 2024.

[37] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[38] Lifan Yuan, Weize Chen, Yuchen Zhang, Ganqu Cui, Hanbin Wang, Ziming You, Ning Ding, Zhiyuan Liu, Maosong Sun, and Hao Peng. From $f(x)$ and $g(x)$ to $f(g(x))$: Llms learn new skills in rl by composing old ones. *arXiv preprint arXiv:2509.25123*, 2025.

[39] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 57654–57689. Curran Associates, Inc., 2025. doi: 10.52202/085713-1933. URL `https://proceedings.neurips.cc/paper_files/paper/2025/file/537d5aa768c2d534016a4d06f87bc8fb-Paper-Conference.pdf`.

[40] Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025.

[41] Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. When scaling meets llm finetuning: The effect of data, model and finetuning method. In *International Conference on Learning Representations*, volume 2024, pages 44694–44713, 2024.

[42] Charlie Zhang, Graham Neubig, and Xiang Yue. On the interplay of pre-training, mid-training, and rl on reasoning language models. *arXiv preprint arXiv:2512.07783*, 2025.

[43] Edwin Zhang, Vincent Zhu, Naomi Saphra, Anat Kleiman, Benjamin L Edelman, Milind Tambe, Sham Kakade, and Eran Malach. Transcendence: Generative models can outperform the experts that train them. *Advances in Neural Information Processing Systems*, 37:86985–87012, 2024.

[44] Yiming Zhang, Athul Paul Jacob, Vivian Lai, Daniel Fried, and Daphne Ippolito. Human-aligned chess with a bit of search. *arXiv preprint arXiv:2410.03893*, 2024.

A Discussions and Limitations

Our testbed mirrors the standard LLM pipeline (pretraining on human data, SFT on reasoning traces, RL with verifiable rewards), and the phenomena we study (how pretraining properties shape RL scaling, how RL reshapes the inherited policy) are pipeline-level dynamics rather than chess-specific ones. That said, chess differs from natural language in ways that limit direct transfer: the vocabulary is small (81 tokens), verification is exact, and reasoning is not entangled with world knowledge or fluency. Our scaling exponents and category proportions should therefore be read as characterizing the structure of the pretraining-to-RL interface in a controlled setting, not as quantitative predictions for language models. Additionally, our RL environment uses puzzles with unique designated solutions and binary rewards, which is a restricted form of verification compared to the partial-credit or open-ended rewards common in language tasks. Furthermore, our models reach at most 1B parameters. The scaling trends we identify (e.g., the decreasing optimal pretraining fraction) may behave differently at larger scale, and verifying this is beyond our current compute budget. Besides, our structured CoT format uses a specific tree-based serialization and data-generation choices. Different reasoning trace formats could yield different CoT evolution dynamics under RL, which is worth exploring as a future direction. We also discuss the limitations for the fitted law in Section G.8.

Future work. There are several exciting directions for future work. First, the scaling framework can be used to study when to switch from pretraining to RL, or more generally how to allocate compute across (N, T, C_{RL}). Second, improving RL beyond pass@1 likely requires methods that reduce wrong-mode amplification and expand the support of correct solutions, rather than only sharpening the current policy. Third, these results might motivate better ways to combine pretraining and RL. Since RL gives limited gains when started from weakly pretrained checkpoints but pure RL can also amplify wrong modes on harder states, a fixed two-stage recipe may be suboptimal. Future work could study interleaving strategies that decide when additional pretraining data is more valuable than additional RL updates. More broadly, our setting also provides a controlled testbed for studying synthetic data design, self-play, transcendence [43], and weak-to-strong generalization [4].

B Extended Related Work

Reinforcement Learning for Reasoning. RL with verifiable rewards has become standard for improving reasoning in language models [11, 18, 37, 40], yet what RL actually does to the pretrained policy remains debated. Prior work has variously argued that RL primarily amplifies existing reasoning patterns [39], improving pass@1 while sometimes reducing large-k coverage; composes pretrained skills into new ones [38]; or does both depending on the problem regime [34, 42]. The difficulty of resolving this in natural language stems from the enormous action space, ambiguous token-level actions, and lack of step-level supervision. Our chess setting provides engine-based supervision at every board state and a clean definition of actions in token space. We show that RL amplifies existing correct moves on easy puzzles, while surfacing buried moves and reinforcing incorrect ones on hard puzzles.

Pre-to-Post Scaling Laws for Language Models. Scaling laws have long been used as quantitative tools to predict language-model performance from training compute. Kaplan et al. [15] showed that language-model loss follows power laws in model size, data, and compute, and Hoffmann et al. [13] refined the compute-optimal allocation, finding that model size and training tokens should scale roughly equally. Subsequent work has extended scaling analysis beyond standard pretraining loss: Gadre et al. [10] show that overtrained language models scale reliably in both validation loss and average downstream top-1 error, while Roberts et al. [26] incorporate test-time sampling through pass@k and show that overtraining can become compute-optimal when inference compute is accounted for. Other work studies inference-aware scaling [28] and synthetic pretraining data [25]. On the post-training side, Zhang et al. [41] study scaling laws for fixed-data SFT, rather than on-policy RL, and find that fine-tuning performance depends more strongly on model scale than on pretraining-data scale. More recent work has also begun to characterize scaling behavior during RL post-training: Khatri et al. [17] fit sigmoidal RL scaling curves and identify stable training recipes, while Cheng et al. [6] study compute-optimal allocation for sampling within RL. However, both treat the pretrained initialization as fixed. Across these lines of work, pretraining scaling laws treat downstream use as given, while post-training scaling laws treat the initialization as given. Neither asks how pretraining compute changes the shape of the RL scaling curve. Our work fills this gap: we

show that pretraining loss predicts post-RL performance level and pretraining data scale predicts the rate of RL improvement in terms of average reward.

Allocating Compute between Pretraining and RL. An important question in multi-stage language-model training is how much pretraining is needed to make RL effective, and how limited compute should be allocated between pretraining and RL post-training. Chen et al. [5] theoretically connect pretraining and post-training through coverage, arguing that the probability mass assigned to high-quality responses determines whether post-training and test-time scaling can succeed. Empirically, Qi et al. [24] study tradeoffs across pretraining, continued pretraining, SFT, and RL, showing that post-training gains can saturate as pretraining and RL compute increase. Recent work further studies early RL during pretraining [3], front-loading reasoning data into pretraining [1], and how RL gains depend on pretraining headroom and task difficulty [42]. These works provide important evidence that pretraining and RL are coupled, but they mainly characterize stage-wise effects or training recipes rather than directly modeling the pretraining-vs.-RL compute allocation problem. Our results show that for large-k pass@k, extended pretraining remains consistently beneficial. For pass@1, additional pretraining also improves performance, but its marginal benefit decreases as the total compute budget increases. Our findings also offer a complementary explanation for the saturation observed in prior multi-stage studies [24]. Since post-RL performance is predictable from pretraining loss, diminishing returns in pretraining loss naturally translate into diminishing marginal gains after RL. Thus, saturation from increased pretraining need not imply a conflict between pretraining and RL. Rather, RL inherits part of its attainable performance level from the pretrained policy, while its learning rate is shaped by pretraining data scale. This perspective turns stage-wise saturation into a predictable consequence of the pretraining-to-RL scaling relationship.

Controlled Testbeds for Studying Reasoning. A line of work has studied controlled synthetic settings that isolate specific aspects of reasoning [36, 42, 38, 34]. We offer a complementary testbed based on chess that more closely mirrors language-model training: models are first pretrained on human behavior data and then improved through RL with verifiable rewards. Chess requires multi-step planning, supports exact verification at every move, and yields nontrivial scaling regimes across both pretraining and RL, making it a useful environment for studying reasoning across the pre-to-post pipeline. Prior chess language models study pretraining or amortized search without RL [9, 27, 44], RL from scratch without pretraining on human data [33], or reasoning evolution from a fixed-size model [8]. None study how pretraining scale shapes RL scaling behavior. Our work fills this gap across model sizes from 5M to 1B parameters.

C Implementation Details

C.1 Datasets

Pretraining datasets. We collect Blitz and Rapid games played on Lichess in 2022. Unless otherwise noted, all games start from the standard chess initial position and follow standard legal-move rules. To ensure game quality, we filter out games shorter than 10 plies and sample games to balance the average Elo rating of the two players from 800 to 3000, yielding a corpus of 54B tokens.

Post-training datasets. We construct the post-training dataset from Lichess puzzles, retaining only puzzles with average player Elo above 800 and popularity above 100 to improve data quality. The resulting dataset contains 156K puzzles. We partition puzzles into five Elo-based difficulty bins, denoted as B1 through B5 in increasing order of difficulty with Elo ranging from 800 to 3000. We further balance the dataset by solution length and theme coverage using a greedy sampler. In our experiments, we sample 33K puzzles uniformly from Elo 800 to 2400 for SFT training and 62.3K puzzles with an easy-skewed distribution (70% from B1 and B2, 30% from B3 to B5) for RL training.

Test datasets. Table 1 presents details of the test benchmark. For the Lichess puzzle subsets, we first assign puzzles to Elo bins and remove puzzles with solution lines longer than 12 moves. Within each bin, we use an approximate greedy sampler to balance theme coverage and solution length. Each candidate is ranked by the average current frequency of its theme tags in the selected set, with ties broken by the frequency of its solution length. We select lower-frequency candidates first and update these counts after each batch, yielding difficulty-stratified test sets with broader coverage over tactical themes and solution depths.

Decontamination. To prevent contamination, we discard from the pretraining corpus any game whose trajectory passes through a position that also appears in the post-training or test sets. For each game we replay its move sequence from the initial position and compare every reached position against the set of post-training and test board states, matching on a normalized FEN consisting of the piece placement and side to move. Any game with a match is removed in full. For the post-training and test sets, which are puzzles defined by a starting position, we deduplicate on the starting board position.

Table 1: **Test benchmark details.** The Lichess puzzle subsets evaluate tactical performance across Elo difficulty, puzzle themes, and solution depths.

Dataset	Size	Description
Puzzle `B1`	308	Lichess puzzles with Elo rating in $(800, 1200]$.
Puzzle `B2`	298	Lichess puzzles with Elo rating in $(1200, 1600]$.
Puzzle `B3`	267	Lichess puzzles with Elo rating in $(1600, 2000]$.
Puzzle `B4`	287	Lichess puzzles with Elo rating in $(2000, 2400]$.
Puzzle `B5`	320	Lichess puzzles with Elo rating above 2400.

C.2 Models

Table 2 presents the architecture details of our model family. We do not tie the input and output embeddings. In addition, motivated by prior work showing the effectiveness of deeper designs for small language models [19], we use relatively deep architectures at each model scale.

Table 2: **Model architecture details.** All models use a Qwen-style decoder-only Transformer architecture with grouped-query attention.

Model	Params	Layers	Hidden Size	Intermediate Size	Heads (Q / KV)	Head Size
5M	5.1M	6	256	768	2 / 2	128
10M	11.8M	6	384	1024	4 / 4	128
20M	20.5M	6	512	1536	4 / 4	128
32M	31.6M	8	512	1536	8 / 4	128
50M	47.3M	12	512	1536	8 / 4	128
100M	101.6M	12	768	2304	12 / 4	128
200M	203.1M	24	768	2304	12 / 4	128
410M	411.3M	28	1024	3072	16 / 4	128
680M	678.6M	30	1280	3840	20 / 4	128
1B	1.03B	32	1536	4608	24 / 4	128

C.3 Algorithms

SFT objective. Let $M_t \in \{0, 1\}$ be the loss mask, where $M_t = 1$ if w_t belongs to the synthetic trace or a model move, and $M_t = 0$ otherwise. The SFT objective is $\mathcal{L}_{\mathrm{SFT}}(\theta_{\mathrm{sft}}) = \mathbb{E}_{(s,w)\sim\mathcal{D}_{\mathrm{SFT}}}\left[-\sum_{t=1}^{|w|} M_t \log \pi_{\theta_{\mathrm{sft}}}(w_t \mid s, w_{<t})\right]$.

RL algorithm. We optimize the policy using GRPO as formalized below. For each state s_0, we sample a group of trajectories $\zeta_1, \ldots, \zeta_G \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid s_0)$, compute rewards $r_i = R(\zeta_i, s_0)$, and normalize rewards within the group to obtain advantages $A_i = \frac{r_i - \mathrm{mean}(\{r_j\}_{j=1}^G)}{\mathrm{std}(\{r_j\}_{j=1}^G)}$. The GRPO objective is

$$\mathcal{J}_{\mathrm{GRPO}}(\theta_{\mathrm{rl}}) = \mathbb{E}_{s_0,\{\zeta_i\}_{i=1}^G}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\Big(\rho_{i,t}A_i,\ \mathrm{clip}\big(\rho_{i,t}, 1-\epsilon, 1+\epsilon\big)A_i\Big)\right] - \beta D_{\mathrm{KL}}\big(\pi_{\theta_{\mathrm{rl}}} \,\|\, \pi_{\mathrm{ref}}\big)\,.$$

where $\rho_{i,t} = \frac{\pi_{\theta_{\mathrm{rl}}}(\zeta_{i,t}|s_0,\zeta_{i,<t})}{\pi_{\theta_{\mathrm{old}}}(\zeta_{i,t}|s_0,\zeta_{i,<t})}$ is the importance sampling weight and β is the KL coefficient.

Table 3: **Pretraining configurations.**

Configuration	Value
Hardware	8 NVIDIA H200 GPUs
Context length	1024
Epochs	1
Optimizer	AdamW
Peak learning rate	1×10^{-3}
Minimum learning rate	1×10^{-4}
Learning-rate schedule	Cosine decay
Warmup ratio	0.05
Adam betas	$(0.9, 0.95)$
Weight decay	0.1
Dropout	0.0
Max gradient norm	1.0
Per-device batch size	32 sequences
Gradient accumulation steps	2
Effective batch size	512 sequences
Tokens per optimizer step	524,288

C.4 Training Configurations

Pretraining configurations. All models are pretrained for one epoch over their assigned pretraining-token budget. All model scales share the optimization setup listed in Table 3.

SFT training configurations. We initialize from the corresponding pretrained checkpoint and extend the context length to 3072 tokens. SFT is performed on 4 NVIDIA H100 GPUs for 3 epochs using AdamW with learning rate 3×10^{-4}, weight decay 0.01, and Adam betas $(0.9, 0.95)$. The learning rate follows a cosine schedule with a warmup ratio of 0.01 and a minimum learning rate of 1×10^{-5}. We use an effective batch size of 524,288 tokens per optimizer step, apply gradient clipping with maximum norm 1.0, and train with standard cross-entropy loss. During SFT, prompt tokens are masked from the loss, so optimization is applied only to the response tokens.

RL training configurations. All experiments are implemented using the `verl` framework [30] with hyperparameters: learning rate 10^{-5}, KL coefficient $\beta = 0.001$, clip ratio $\epsilon = 0.2$ and no entropy regularization. We set group size $G = 8$. For response sampling, we fix the sampling temperature 1.0 and a maximum response length of 3072 tokens. In `verl`, we set both the training batch size and mini-batch size to 256 prompts. A simple rule-based reward function is used, assigning reward 1 to correct answers and 0 otherwise, without incorporating any format-related signals. RL is performed on 8 NVIDIA H200 GPUs.

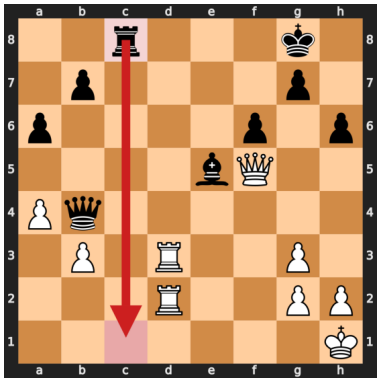
C.5 Puzzle Game Example

Fig. 7 shows an example of a chess puzzle game. Each puzzle provides a starting position, which we convert into a move sequence for the model. To solve the puzzle, the player must identify every correct move in the continuation.

C.6 Additional Details of Synthetic Reasoning Trace Generation

Synthetic trace construction. We visualize the procedure in Fig. 1. Starting from an input board s_0, we first use a proposal policy $p_{\theta_{\text{prop}}}$ to generate K rollout traces $\tau_1, \ldots, \tau_K \sim p_{\theta_{\text{prop}}}(\cdot \mid s_0)$, where K is controlled by the budget. The proposal policy is either the model pretrained in Section 2.1 or a policy constructed from a chess engine. For each rollout τ_k, we sample a length budget L_k and generate moves until either the game terminates or the sampled budget L_k is reached.

We then convert each rollout τ_k into a move sequence $(a_{k,1}, o_{k,1}, \ldots,)$ by discarding invalid or unparsable suffixes. The resulting move sequences are merged by shared prefixes into a prefix tree $\mathcal{G}$ rooted at s_0. Each non-root node u is labeled by a move a_u, and its board state s_u is obtained by applying the moves along the path from the root to u. In principle, a verifier can assign scores



Prompt:

```
Pe2e4 Pe7e5 Ng1f3 Nb8c6 Bf1c4 Bf8c5 0-0 Ng8f6 Pc2c3 Nf6xe4 Pd2d4 Pe5xd4
Pc3xd4 Bc5e7 Rf1e1 Pd7d5 Bc4d3 Ne4f6 Bc1g5 0-0 Nb1c3 Bc8g4 Bd3e2 Ph7h6
Bg5h4 Qd8d7 Bh4g3 Pa7a6 Nf3e5 Nc6xe5 Pd4xe5 Bg4xe2 Re1xe2 Nf6h5 Nc3xd5
Nh5xg3 Pf2xg3 Be7c5+ Kg1h1 Qd7e6 Nd5xc7 Qe6b6 Nc7xa8 Rf8xa8 Pb2b3 Bc5d4
Ra1b1 Ra8d8 Qd1f1 Qb6e6 Re2e1 Bd4xe5 Rb1d1 Rd8c8 Rd1d3 Rc8c2 Rd3e3 Pf7f6
Pa2a4 Qe6b6 Re1e2 Rc2c5 Re3d3 Rc5c8 Re2d2 Qb6b4 Qf1f5  <T>
```

Solution: player moves

```
Rc8c1+  [player]
Rd2d1   [env]
Qb4e1+  [player]
Rd1xe1  [env]
Rc1xe1+  [player]
Qf5f1   [env]
Re1xf1#  [player]
```

Figure 7: **An example of a chess puzzle.** The prompt consists of the move sequence representing the board state, optionally followed by a <T> token that triggers reasoning. The player is required to figure out all correct moves in the continuation to solve this puzzle. The best solution line is unique for all puzzle games.

Table 4: An example of reasoning trace generated by the model. Given an input prompt, the model generates several possible continuations, each containing both the player's and the opponent's moves. These continuations are concatenated using the <sep> token. The model is then expected to generate the final answer move, ideally by selecting the best move from the reasoning traces. The tokens <T> and </T> indicate the reasoning format.

Input	`Pe2e4 Pc7c5 Ng1f3 Nb8c6 Pd2d4 Pc5xd4 Bf1c4 Pg7g6 Nf3g5 Pe7e6 Pc2c3 Pd7d5 Pe4xd5 Pe6xd5 Qd1f3 Ng8f6 Bc4xd5 Nf6xd5 <T>`
Reasoning	`Ng5h3 Bf8g7 <sep> Ng5h3 Nc6e5 Qf3xf7# Ne5xf7 <sep> Ng5h3 Bc8g4 Qf3xf7# <sep> Ng5h3 Qd8d6 Bc1f4 Qd6e7+ <sep> Ng5h3 <sep> Ng5xf7 Nd5e7 Bc1f4 Rh8g8 Nf7xd8 Ne7f5 Qf3xc6+ <sep> Bc1f4 Nd5xf4 Qf3xf4 Bc8f5 Qf4e5+ Bf8e7 <sep> Bc1f4 Bf8g7 Ng5xf7 Rh8f8 <sep> Bc1f4 Bf8e7 Bf4e5 <sep> Qf3xf7# <sep> O-O Nd5f6 Nc6e5 Qf7xd5 Ne5c6 <sep> Qf3xf7# <sep> Pg2g3 Qd8f6 Qf3xf6 Rh8g8 <sep> Pg2g3 Qd8e7+ Ke1f1 Bc8e6 Ng5xe6 Qe7xe6 <sep> Qf3xf7# </T>`
Answer	`Qf3xf7#`

to nodes, which may be used to rank, prune, or truncate $\mathcal{G}$ under the serialization budget. In our implementation, we use the simplest rule: we discard illegal moves, preserve the proposal policy's sampling order when ordering children in the prefix tree, and ensure that the best-of-K continuation is included in the serialized trace.

Finally, we serialize the constructed tree by depth-first traversal (DFS). Let $(u_1, \ldots, u_m)$ denote the retained leaf nodes of $\mathcal{G}$ ordered by DFS. For each leaf node u_j, let $\tilde{\tau}_{u_j}$ denote the rendered move sequence along the path from the root to u_j. The final synthetic reasoning trace is obtained by concatenating these root-to-leaf paths in DFS order: $r =$ <T> $\tilde{\tau}_{u_1}$ <sep> $\tilde{\tau}_{u_2}$ <sep> $\cdots$ $\tilde{\tau}_{u_m}$ </T>.

After producing the synthetic trace, the model is trained to commit to a single continuation. We choose this continuation from the same rollout set used to construct the trace. Specifically, among the valid parsed continuations, we select the best one according to a verifier, such as Stockfish[6], and use it as the supervised answer following the trace. Let the selected continuation be $\tau^\star = (a_1, o_1, a_2, o_2, \ldots, a_H)$, where a_t is the model move and o_t is the opponent move. We train on the concatenated sequence $w = (r, \tau^\star)$. Since opponent moves will be produced by the environment rather than by the model policy during inference, we mask the opponent-move tokens in $\tau^\star$ out of the loss and apply supervision only to r and model moves in $\tau^\star$. Let $M_t \in \{0, 1\}$ be the loss mask, where $M_t = 1$ if

[6] https://github.com/official-stockfish/stockfish

w_t belongs to the synthetic trace or a model move, and $M_t = 0$ otherwise. The SFT objective is $\mathcal{L}_{\mathrm{SFT}}(\theta_{\mathrm{sft}}) = \mathbb{E}_{(s,w)\sim\mathcal{D}_{\mathrm{SFT}}}\left[-\sum_{t=1}^{|w|} M_t \log p_{\theta_{\mathrm{sft}}}(w_t \mid s, w_{<t})\right]$.

Implementation details for synthesizing reasoning traces. Following the procedure described in Section 2.2, we use a fixed proposal policy, a pretrained 200M model, to generate synthetic reasoning traces. To diversify the reasoning paths, for each puzzle, we branch at the first move: we sample $n = 8$ first-move candidates from the policy model. From each candidate we then roll out continuations under the same policy, where the number of traces per candidate m and the trajectory length l are drawn from rounded log-normal distributions truncated to $[2, 10]$ and $[1, 20]$, with means 8 and 5 and shape parameters $\sigma = 0.2$ and $\sigma = 0.4$, respectively. The two means roughly correspond to $1/3$ of the legal action space and the typical puzzle solution length in the SFT dataset. We did not tune these values.

D FLOP Estimation

We estimate training compute in model FLOPs using the standard dense language model approximation

$$C_{\mathrm{train}}(N, T) \approx 6NT, \tag{2}$$

where N is the number of active model parameters and T is the number of tokens processed during training. This convention is widely used in language model scaling analyses [15, 13].

This approximation is appropriate for the dense Qwen3 models used in our experiments because they are dense decoder-only Transformer language models: all non-embedding Transformer parameters are active for every processed token. Thus, their dominant training cost comes from dense matrix multiplications in attention and feed-forward layers, which is precisely the regime targeted by the $6NT$ approximation. We use N as the active dense parameter count of the trained model.

Pretraining. Let T_{pt} denote the number of pretraining tokens. We estimate pretraining FLOPs as $C_{\mathrm{pt}} = 6NT_{\mathrm{pt}}$.

Supervised fine-tuning. Let $\mathcal{D}_{\mathrm{sft}}$ denote the SFT dataset, let ℓ_i denote the tokenized length of example i, and let E_{sft} denote the number of SFT epochs. The total number of SFT training tokens is $T_{\mathrm{sft}} = E_{\mathrm{sft}} \sum_{i\in\mathcal{D}_{\mathrm{sft}}} \ell_i$. Therefore, we estimate SFT FLOPs as $C_{\mathrm{sft}} = 6NE_{\mathrm{sft}} \sum_{i\in\mathcal{D}_{\mathrm{sft}}} \ell_i$. In our experiments, all SFT examples are padded or truncated to a fixed context length L_{sft}. Thus, if n_{sft} examples are used per epoch, the estimate simplifies to $C_{\mathrm{sft}} = 6NE_{\mathrm{sft}}n_{\mathrm{sft}}L_{\mathrm{sft}}$.

Reinforcement learning. For GRPO, let n_{prompt} denote the total number of training prompts processed over the entire RL stage and let g denote the group size, i.e., the number of sampled responses per prompt. For prompt q, let $L^{(q)}_{\mathrm{prompt}}$ denote the prompt length and let $L^{(q,i)}_{\mathrm{resp}}$ denote the response length of sample i. Define $L^{(q,i)}_{\mathrm{rl}} = L^{(q)}_{\mathrm{prompt}} + L^{(q,i)}_{\mathrm{resp}}$. The total number of rollout tokens is then computed as

$$T_{\mathrm{rollout}} = \sum_{q=1}^{n_{\mathrm{prompt}}} \sum_{i=1}^{g} L^{(q,i)}_{\mathrm{rl}}.$$

We estimate RL compute as the sum of policy rollout generation, reference-model log-probability evaluation, and policy optimization. Policy rollout generation and reference-model evaluation each require one forward pass over the rollout tokens, while policy optimization is estimated using the standard training FLOP approximation. Thus,

$$\begin{aligned} C_{\mathrm{rl}} &= 2NT_{\mathrm{rollout}} + 2NT_{\mathrm{rollout}} + 6NT_{\mathrm{rollout}} \\ &= 10NT_{\mathrm{rollout}} \\ &= 10N \sum_{q=1}^{n_{\mathrm{prompt}}} \sum_{i=1}^{g} L^{(q,i)}_{\mathrm{rl}}. \end{aligned}$$

If no reference model is used, the second $2NT_{\mathrm{rollout}}$ term is omitted and C_{rl} is reduced to $8NT_{\mathrm{rollout}}$.

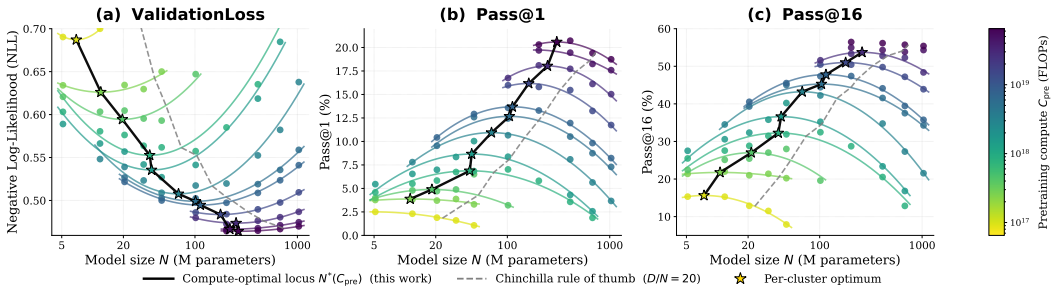


Figure 8: **IsoFLOP curves for pretrained models across a sweep of 11 compute budgets and 10 model sizes.** (a) Pretraining validation loss (NLL) on held-out human games. (b) Pass@1 and (c) Pass@16 on the puzzle benchmark. The starred points mark the compute-optimal model size at each budget. The solid line connects these optima, while the dashed line represents the Chinchilla allocation (D/N = 20). Over the plotted range, the chess-optimal locus lies at smaller models and more tokens than this rule, with the two converging at higher compute.

E Pretraining Law Fitting

For pretraining, we sweep 11 compute budgets from 6.5×10^{16} to 6.5×10^{19} FLOPs across 10 model sizes, corresponding to training runs of approximately 200M to 52B tokens, following the setup of isoFLOP methodology in prior scaling-law studies [13]. Fig. 8 reports isoFLOP curves on validation loss on held-out human games, along with pass@1 and pass@16 on the downstream puzzle benchmark (including all puzzles from B1 to B5). Note that for pretraining evaluation, all models just generate move sequences without reasoning. We observe a clear valley in validation loss at each budget, confirming that an optimal parameter-token allocation exists under fixed FLOPs. Comparing this optimum to the Chinchilla-style compute-optimal allocation for natural language modeling (grey dashed line in Fig. 8), our results indicate that chess pretraining favors smaller models trained on more tokens than the language-modeling baseline over the compute range studied. Moreover, the isoFLOP optima for pass@1 and pass@16 on the puzzle benchmark closely track the validation-loss optimum on human games. Within each model size, downstream performance continues to improve with additional pretraining over the compute range we study.

To complement the IsoFLOP analysis, we refit the parametric form of Hoffmann et al. [13] to our pretraining sweep:

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}, \tag{3}$$

where N is the number of total parameters, D is the number of training tokens, and (E, A, B, α, β) are fitted parameters. We optimize the Huber loss between predicted and observed validation loss using L-BFGS over $n = 64$ (N, D, L) triples drawn from our sweep, restricting to runs with validation loss below 1.0 to exclude undertrained outliers.

Table 5 reports the fitted parameters alongside the Hoffmann et al. [13] values for language modeling. First, the parameter exponent $\alpha = 0.4006 \pm 0.0399$ remains close to the language-modeling estimate ($\alpha \approx 0.34$), suggesting that returns to model scale are similar across domains. Second, the data exponent $\beta = 0.6789 \pm 0.0291$ is more than twice the language-modeling estimate ($\beta \approx 0.28$), indicating that loss falls substantially faster with additional training tokens in the chess setting holding model size fixed. The RMSE is 0.0097 for the fitting. The implied compute-optimal allocation,

$$N_{\text{opt}}(C) \propto C^{\beta/(\alpha+\beta)} = C^{0.63}, \quad D_{\text{opt}}(C) \propto C^{\alpha/(\alpha+\beta)} = C^{0.37}, \tag{4}$$

implies that the compute-optimal model size grows more rapidly than the token budget. This is consistent with the IsoFLOP curves (Fig. 8): over the compute range we study, the chess optima use more tokens per parameter than the $D/N = 20$ rule, but this gap narrows as compute grows.

F SFT Performance Comparisons

For SFT, we compare two settings: (1) *SFT without reasoning* directly fine-tunes the model on the target move sequence, with the loss masked on opponent moves; (2) *SFT with reasoning traces* trains

Table 5: **Chinchilla functional form parameters fit to our chess pretraining sweep, compared to the language-modeling estimates of Hoffmann et al. [13].**

Parameter	Chess (ours)	Language [13]
E	0.412 ± 0.009	—
α	0.401 ± 0.040	≈ 0.34
β	0.679 ± 0.029	≈ 0.28
$\beta/(\alpha+\beta)$	0.63	≈ 0.46

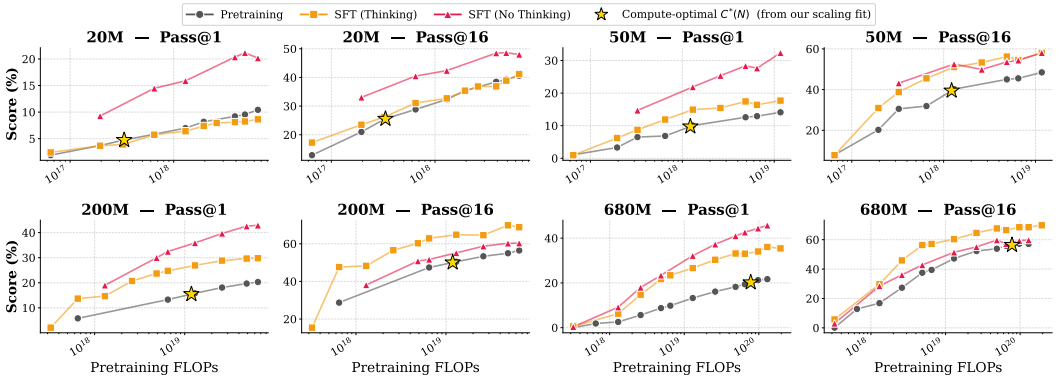


Figure 9: **Effect of SFT with and without reasoning traces across four model sizes.** Each panel shows puzzle benchmark performance (B1–B5, macro-averaged) as a function of pretraining compute. The star marks the compute-optimal pretraining budget from the isoFLOP fit.

the model on synthetic intermediate reasoning traces followed by the target answer. We train all SFT variants using the same number of puzzle samples across models. Fig. 9 compares performance across model sizes and pretraining FLOPs. For a fixed model size, stronger pretrained models consistently achieve higher post-SFT performance, indicating that pretraining quality transfers to SFT. SFT without reasoning traces improves pass@1 at every scale, and for small models (20M, 50M) it also improves pass@16. At larger scales (200M, 680M), however, its pass@16 gain over the pretrained baseline shrinks, while SFT with reasoning traces gives the highest pass@16, especially on harder puzzles. This suggests that learning on reasoning traces enriches not just the most likely solution but the broader distribution of candidates the model can generate.

To better understand why SFT with reasoning traces underperforms SFT without reasoning on pass@1, we examine two aspects of our training setup. First, our current CoT format requires the model to predict both player and opponent moves. While the player's moves in a puzzle are optimal, the opponent's moves are not necessarily optimal. The model may therefore consider valid alternative continuations that do not match the prescribed puzzle continuation. Our ablation intervenes during generation by replacing the model-generated opponent moves with the ground-truth opponent moves and allowing the model to continue reasoning from the edited trace. This intervention improves pass@1. RL may improve the model's ability to consider multiple possible branches for opponent moves. Second, although both settings use the same number of training updates, answer tokens constitute a much smaller fraction of the supervised tokens in the reasoning-trace setting and therefore receive less weight under token-averaged loss. This difference in effective answer supervision may also contribute to the gap. We leave the exploration of alternative reasoning-trace formats and data-generation choices, such as using different proposal models, to future work.

Implementation note. In our current SFT experiments, the intended RoPE extension was not applied, requiring the models to learn to adapt positions beyond the pretraining context length during SFT. Fig. 9 shows that using the positional encoding extension implementation improves SFT performance, particularly for smaller models.

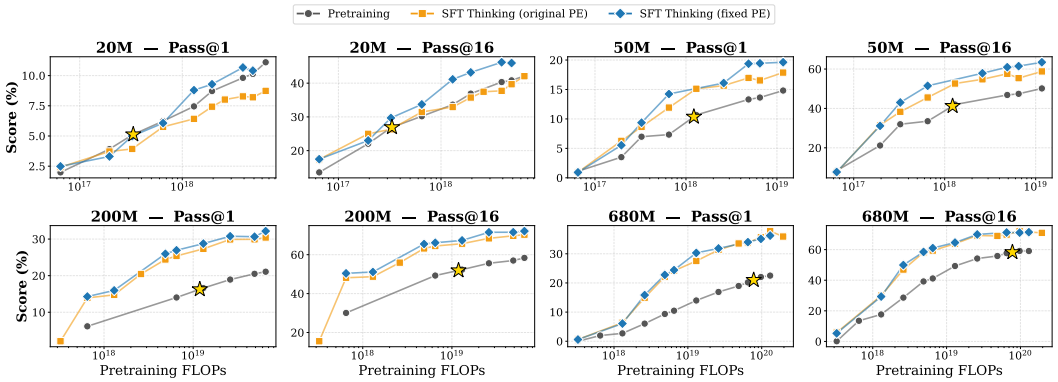


Figure 10: **SFT performance with fixed positional encoding.**

G Joint Pretraining–RL Scaling Law

This appendix describes our statistical model used to relate pretraining scale, pretraining loss, and downstream RL compute. The goal is to obtain a simple predictive law for the reward obtained after RL, conditioned on a pretrained model with parameter count N, pretraining tokens T, and RL compute C_{RL} as we discuss in Section 3.3.

G.1 Interpretation of the Local RL Scaling Fit

Prior work [17] models the performance of a fixed model under RL compute with a sigmoid compute–performance curve:

$$R^{\mathrm{sig}}(C) - R_0 = (A - R_0)\frac{1}{1 + (C_{\mathrm{mid}}/C)^{\gamma}}, \tag{5}$$

where R_0 is the initial performance, A is the asymptotic reward ceiling, C_{mid} controls the transition location, and γ controls the sigmoid steepness. This form explicitly models saturation, but fitting the full curve requires sufficiently long RL runs to identify both the transition point and the plateau. In our experiments, running every configuration long enough to reach the plateau regime would require substantially more RL compute, and in our available compute range we found the asymptote A to be weakly identified. We therefore focus on the local, non-saturated regime, which is also scientifically useful: it isolates how pretraining scale affects RL improvement before the curve is dominated by asymptotic saturation.

Let $x = \log_{10} C$ and $x_{\mathrm{ref}} = \log_{10} C_{\mathrm{ref}}$. A first-order Taylor expansion of Eq. (5) around x_{ref} gives

$$R^{\mathrm{sig}}(x) \approx R^{\mathrm{sig}}(x_{\mathrm{ref}}) + \left.\frac{dR^{\mathrm{sig}}}{dx}\right|_{x=x_{\mathrm{ref}}} (x - x_{\mathrm{ref}}).$$

Thus, locally, the sigmoid reduces to a log-linear form:

$$R_{N,T}(C) = R^{\mathrm{ref}}_{N,T} + B_{N,T}\left(\log_{10} C - \log_{10} C_{\mathrm{ref}}\right),$$

where $R^{\mathrm{ref}}_{N,T} = R_{N,T}(C_{\mathrm{ref}})$, and $B_{N,T}$ is the local reward slope. To interpret the local slope, differentiate Eq. (5) with respect to $x = \log_{10} C$. This gives

$$\frac{dR^{\mathrm{sig}}}{dx} = \gamma \ln(10)(A - R_0)\frac{(C_{\mathrm{mid}}/C)^{\gamma}}{\left[1 + (C_{\mathrm{mid}}/C)^{\gamma}\right]^2}.$$

Using

$$\frac{R^{\mathrm{sig}}(C) - R_0}{A - R_0} = \frac{1}{1 + (C_{\mathrm{mid}}/C)^{\gamma}}, \qquad A - R^{\mathrm{sig}}(C) = (A - R_0)\frac{(C_{\mathrm{mid}}/C)^{\gamma}}{1 + (C_{\mathrm{mid}}/C)^{\gamma}},$$

the local slope can be rewritten as

$$\frac{dR^{\mathrm{sig}}}{dx} = \gamma \ln(10)\frac{R^{\mathrm{sig}}(C) - R_0}{A - R_0}\left(A - R^{\mathrm{sig}}(C)\right).$$

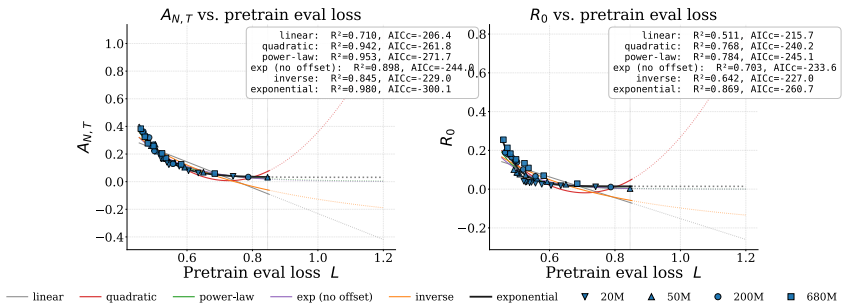


Figure 11: **Comparison of different parameterization choices for f.**

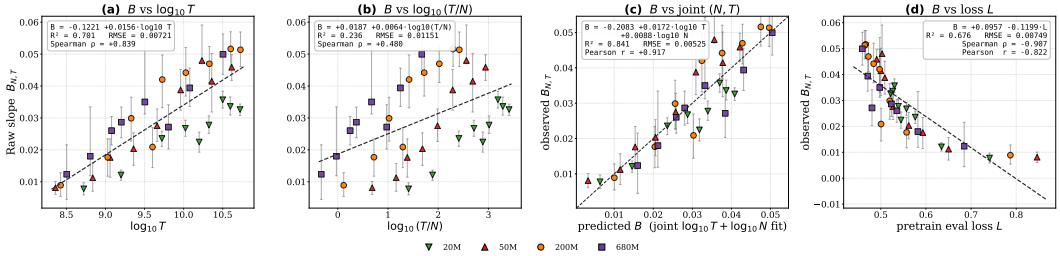


Figure 12: **Comparison of token-only, tokens-per-parameter, loss-based, and joint fits for the RL slope on the B3–B4 benchmark.** We observe a clear monotonic relationship between log-scale pretraining tokens and the RL slope, with a Spearman correlation of 0.84. The joint fit achieves a lower RMSE and higher R^2, with a substantially $2\times$ larger coefficient on log-scale tokens than on model size. Pretraining loss is also a reasonably strong predictor of the slope.

Therefore, at the reference point,

$$B_{N,T} \approx \gamma \ln(10) \frac{R^{\text{ref}}_{N,T} - R_0}{A - R_0} \left(A - R^{\text{ref}}_{N,T}\right). \tag{6}$$

We note that $B_{N,T}$ is a local estimate and can therefore be sensitive to the portion of the RL trajectory used for fitting. For a fixed (γ), the local slope follows a concave quadratic: it is small when the reference point is close to either the initial performance (R_0) or the saturation level (A), and largest in the intermediate regime. Consequently, runs evaluated very early in RL or after approaching saturation can yield systematically smaller and less predictable slope estimates. On the B3-B4 benchmarks, most models remain in the non-saturated regime, making these benchmarks better suited to our slope analysis. We examine these effects empirically in Section G.2.

G.2 Parameterizations

We now describe the parameterizations used in the joint pretraining–RL scaling law. For each pretrained model indexed by model size N and pretraining tokens T, we first fit its RL trajectory as a log-linear function of RL compute:

$$R_{N,T}(C_{\text{RL}}) = R^{\text{ref}}_{N,T} + B_{N,T} \left(\log_{10} C_{\text{RL}} - \log_{10} C_{\text{ref}}\right), \tag{7}$$

where $R^{\text{ref}}_{N,T}$ is the reward at the reference RL compute level C_{ref}, and $B_{N,T}$ is the reward gain per decade of RL compute. In practice, we fit the log-linear form with least-squares regression. We then parameterize the fitted quantities $\phi_{N,T} = \{R^{\text{ref}}_{N,T}, B_{N,T}\}$ using pretraining properties: model size N, number of pretraining tokens T, and pretraining validation loss $L_{\text{pt}}(N, T)$.

We first examine the empirical dependence of $R^{\text{ref}}_{N,T}$ on pretraining validation loss. As discussed in Section 3.3, $R^{\text{ref}}_{N,T}$ is strongly monotonic in L_{pt}, and this relationship becomes tighter as the RL compute slice increases. We compare alternative functional forms in Fig. 11. Among the candidate forms considered, we adopt an exponential parameterization, which provides the best tradeoff between

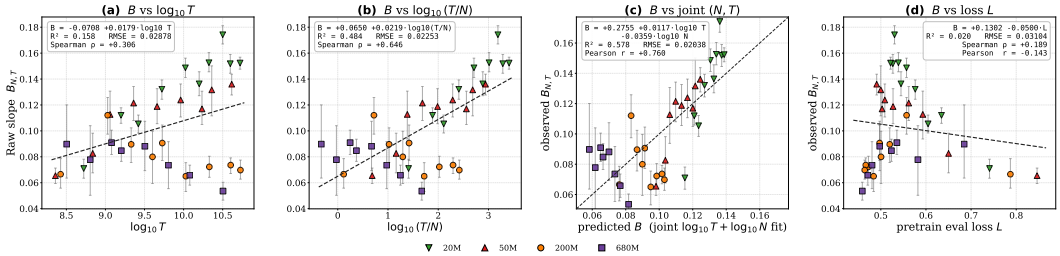


Figure 13: **Comparison between token-only, token-per-parameter and loss fitting for RL slope on B1 benchmark.** On the easiest benchmark, larger and more extensively pretrained models, such as the 200M and 680M models, quickly approach saturation, resulting in low local slope estimates.

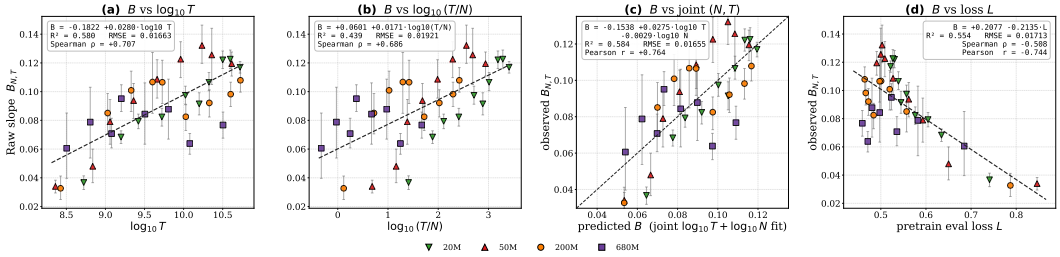


Figure 14: **Comparison of token-only, tokens-per-parameter, and loss-based fits for the RL slope on the B2 benchmark.** On B2, the 200M and 680M models also exhibit some degree of saturation, and pretraining loss is a weaker predictor of the local RL slope.

fit quality, simplicity, and extrapolation behavior:

$$f(L_{\mathrm{pt}}) = \alpha_f + \beta_f \exp(-\gamma_f L_{\mathrm{pt}}), \qquad \gamma_f > 0. \tag{8}$$

This form is monotone decreasing in pretraining loss and avoids the pathological behavior of linear or quadratic fits outside the observed range. Moreover, since the Chinchilla loss surface includes an irreducible-loss term, L_{pt} is bounded below; consequently, $f(L_{\mathrm{pt}})$ has a finite ceiling rather than growing without bound.

Next, we parameterize the RL slope $B_{N,T}$. We compare token-only, token-per-parameter and a joint form over N and T. As illustrated in the benchmark-specific results (Fig. 13 and Fig. 14), stronger pretrained models on easier benchmarks have already been close to saturation, leading to small estimated slopes. At the other extreme, runs observed only during the initial phase of RL may not yet exhibit a stable local scaling trend. These effects introduce noise into both the slope estimates and their downstream prediction. We observe a clear monotonic relationship between log-scale pretraining tokens and the RL slope on B3-B4 benchmarks (Fig. 12), with a Spearman correlation of 0.84. However, the joint fit achieves a lower RMSE and higher R^2. We also note that pretraining loss predicts well on B3-B4 benchmarks but is substantially less reliable on B2. Therefore, the joint form explains the RL slope most robustly:

$$g(N, T) = \alpha_g + \beta_g \log_{10} T + \gamma_g \log_{10} N. \tag{9}$$

The fitted coefficient on $\log_{10} T$ is larger than that on $\log_{10} N$, indicating that the RL slope is driven primarily by the amount of pretraining data, while model size provides a weaker correction.

Combining these two parameterizations yields the final **joint pretraining–RL scaling law**:

$$R(C_{\mathrm{RL}}, N, T) = f(L_{\mathrm{pt}}(N, T)) + g(N, T)\left(\log_{10} C_{\mathrm{RL}} - \log_{10} C_{\mathrm{ref}}\right). \tag{10}$$

When the observed pretraining validation loss is available, the measured L_{pt} can be directly used, which evaluates the quality of the fitted maps f and g. When only the pretraining configuration (N, T) is available, we instead predict $L_{\mathrm{pt}}(N, T)$ using the Chinchilla loss surface (Eq. (3)) and substitute it into Eq. (10). This gives a fully predictive scaling law from (N, T, C_{RL}) to post-RL reward.

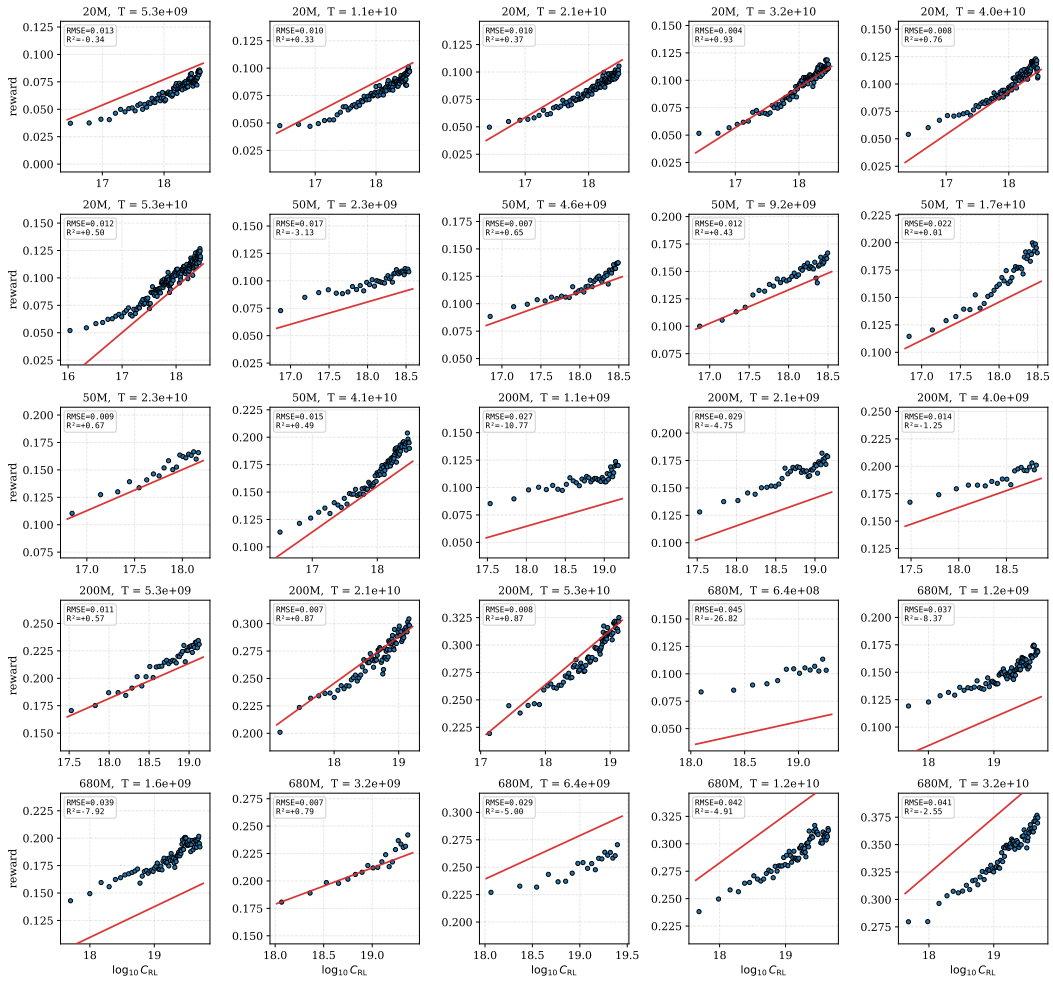


Figure 15: **LOO validation fitting for existing RL runs with Chinchilla-predicted loss.**

G.3 Leave-one-out Fitting Validation

We validate the joint law with leave-one-out (LOO) prediction over the 36 runs with observed pretraining evaluation losses. For each held-out run, we refit f and g on the remaining 35 runs and predict the full held-out RL curve. The Chinchilla loss surface $L(N, T)$ is not refit inside the LOO loop.

We report two validation modes. In the first, called Chinchilla-L LOO, the held-out loss is predicted from (N, T) using Eq. (3). This corresponds to the practical extrapolation setting where only the proposed pretraining configuration is known. In the second, called observed-L LOO, the held-out run's measured pretraining evaluation loss is supplied to f. This isolates the error from the f and g parameterizations, removing the additional error from predicting L. The metrics are reported in Table 6. We report qualitative fitting examples for these two modes in Fig. 15 and Fig. 16.

The absolute error is small: the strict Chinchilla-L mode obtains RMSE 0.0194 in reward. The negative mean per-run R^2 should not be interpreted as a failure of the fit. Some runs have nearly flat observed RL trajectories, so their within-run variance is tiny; dividing by this small variance makes R^2 unstable even when the absolute residuals are only around 0.02–0.03. For this reason, RMSE and MAE are more informative than mean per-run R^2 in this validation.

The comparison between the two LOO modes suggests that pretraining-loss prediction error accounts for a nontrivial fraction of the total error. Moving from observed-L to Chinchilla-L increases RMSE by 0.0099, about 49% of the strict LOO RMSE. The remaining error is attributable

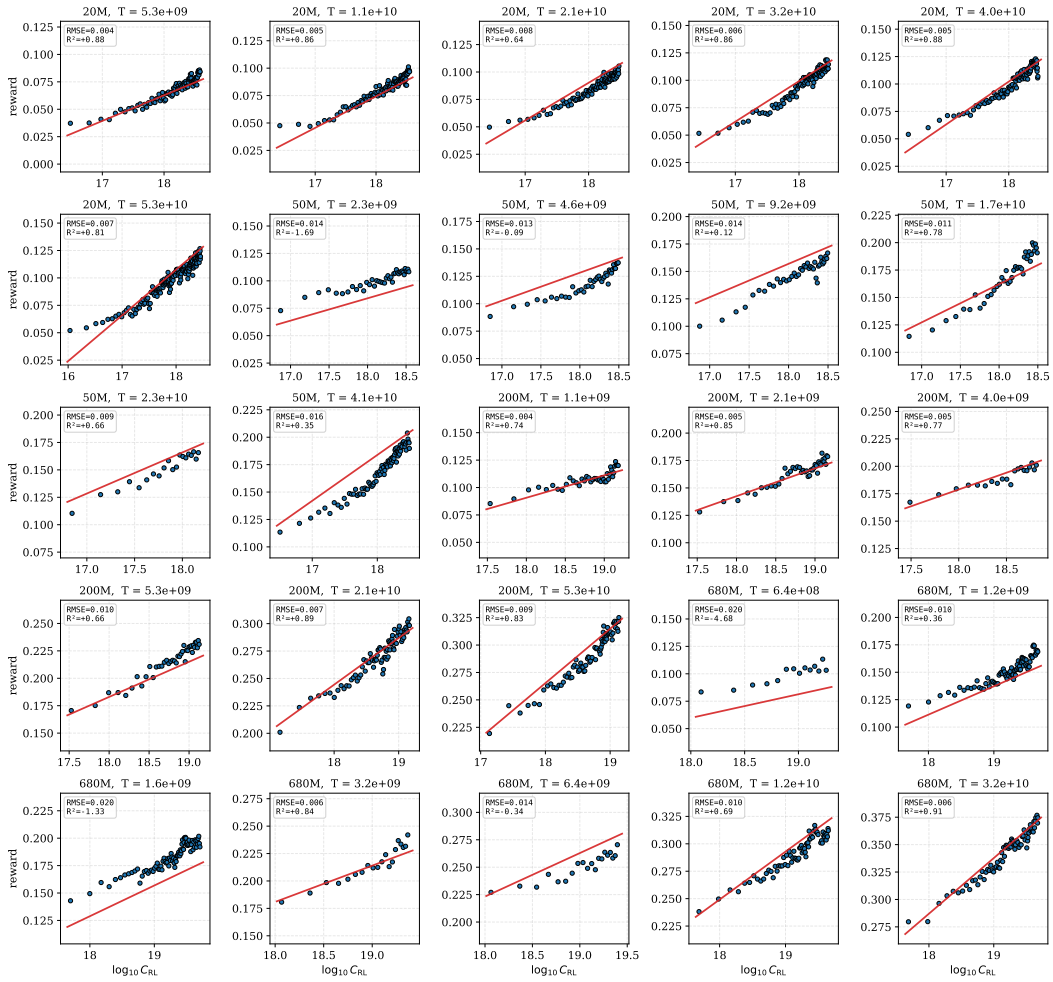


Figure 16: **LOO validation fitting for existing RL runs with observed loss.**

Table 6: **Aggregate leave-one-out validation metrics.** Chinchilla-L uses predicted pretraining loss from (N, T); observed-L uses the measured held-out pretraining loss.

Metric	Chinchilla-L	Observed-L	Difference
Overall LOO RMSE on R	0.0194	**0.0102**	+0.0092
Overall LOO MAE on R	0.0153	**0.0078**	+0.0075
Median per-run R^2	-0.86	**+0.65**	–
Mean per-run R^2	-3.76	**-0.67**	–
Predicted $R^{\text{ref}}_{N,T}$ vs. actual: Pearson r	+0.972	$+\mathbf{0.989}$	–
Predicted $R^{\text{ref}}_{N,T}$ vs. actual: R^2	+0.940	$+\mathbf{0.977}$	–
Predicted $B_{N,T}$ vs. actual: Pearson r	+0.890	+0.890	–

to residual variation in the f and g fits, with $g(N, T)$ being the harder component to predict: $f(L)$ explains the reference reward more cleanly than $g(N, T)$ explains the RL slope.

G.3.1 Leave-one-model-size-out validation

To test whether the joint law extrapolates to unseen model sizes, we perform *leave-one-model-size-out* (LMSO) validation: for each of the four measured sizes $\{20\text{M}, 50\text{M}, 200\text{M}, 680\text{M}\}$, we hold out *all* runs of that size and refit the entire pipeline on the remaining data. Concretely, we (i) drop every pretraining row of the held-out size from the Chinchilla data and refit $L(N, T)$; (ii) refit f and g on

Table 7: **Per-fold LMSO validation on B3B4.** For each held-out size, we report the RMSE of the held-out RL predictions, and the refit joint-law coefficients.

Held-out	RMSE(R)	RMSE(A^{ref})	RMSE(B)	β_g (T)	γ_g (N)
20M	0.013	0.027	0.011	+0.019	+0.004
50M	0.016	0.028	0.008	+0.017	+0.011
200M	0.017	0.021	0.006	+0.016	+0.009
680M	0.048	0.045	0.009	+0.017	+0.014

Table 8: **Aggregate LMSO metrics vs the run-level LOO baseline.** Aggregated across all 36 held-out run, LMSO on R is only 25% larger than the run-level LOO baseline, and predicted $R^{\mathrm{ref}}_{N,T}$ correlates with observed at Pearson $r{=}{+}0.95$.

Metric	Run-level LOO	LMSO
Overall RMSE on R	0.0194	0.0242
Overall MAE on R	0.0153	0.0183
Predicted $R^{\mathrm{ref}}_{N,T}$: Pearson r	$+0.972$	$+0.954$
Predicted $B_{N,T}$: Pearson r	$+0.890$	$+0.792$

the remaining joint-law runs; (iii) predict each held-out run's $A^{\mathrm{ref}}_{N,T}$, $B_{N,T}$, and full RL trajectory using the refit surface. Table 7 reports the per-fold metrics and Table 8 compares the LMSO aggregate to the run-level LOO baseline.

Table 9: **Fitting results on the out-of-sample model sizes (32M and 410M).**

N	T (B tokens)	ΔR^{ref} (fit $-$ law)	ΔB (fit $-$ law)	RMSE (obs-L)	RMSE (Chin-L)
32M	6.87	+0.001	+0.0014	0.0060	0.0086
32M	13.7	+0.005	+0.0048	0.0054*	0.0054
410M	1.98	+0.019	+0.0080	0.0142	0.0315
410M	2.63	+0.032	+0.0083	0.0265	0.0372
Overall	–	–	–	0.0169	0.0232

We interpret the 680M-held-out RMSE as our best empirical estimate of the joint law's extrapolation error at the top of the observed size range. Predictions beyond 680M (e.g., the 1B and 2B extrapolations in Sec. 3.2) should therefore be treated as extrapolations of at least this magnitude. We also validate the fitting results on two model sizes (32M and 410M) that were not used to fit the joint law and evaluated its predictions on them, obtaining low prediction errors (Table 9.

G.4 Choice of reference compute

The reference compute C_{ref} controls where the per-run log-linear RL trajectories are anchored. Table 10 compares several choices. We use $C_{\mathrm{ref}} = 10^{20}$ because it is the best compromise across the two validation modes: it attains the lowest Chinchilla-L LOO RMSE (tied with 10^{19}) and is within 2% of the best observed-L RMSE, while also yielding a stable fit for $f(L)$. No single anchor is best on both criteria, and the choice matters little for the RL-share trend, which varies by only a few points across this range.

G.5 Post-SFT pass@k as an auxiliary validation signal

In Fig. 17, we also evaluate whether post-SFT pass@k metrics show the same dependence on pretraining loss. Across 36 runs (the thinking-track SFT models), pass@k is well fit by an exponential function of pretraining evaluation loss, with stronger fits at larger k. This supports the interpretation that pretraining evaluation loss is a useful summary variable not only for the post-RL reward reference point, but also for post-SFT downstream capability.

Table 10: **Sensitivity to the reference compute C_{ref}.** The selected value is $\log_{10} C_{\text{ref}} = 20$. Parameters are for the offset-exponential form $f(L) = \gamma_f + \alpha_f\, e^{-\beta_f L}$; the offset γ_f is positive at every anchor, so the fitted reference reward stays above zero at large loss.

$\log_{10} C_{\text{ref}}$	α_f	β_f	γ_f	R^2_f	LOO RMSE, Chinchilla-L	LOO RMSE, observed-L
18	214.95	14.58	0.0198	0.972	0.0214	0.0159
19	157.50	13.57	0.0257	0.981	0.0201	0.0125
20	**129.85**	**12.85**	**0.0314**	**0.980**	**0.0201**	**0.0102**
21	114.82	12.32	0.0368	0.975	0.0213	0.0100

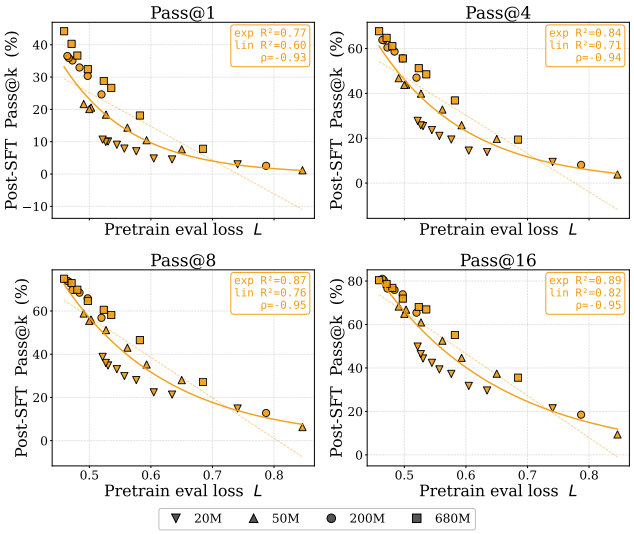


Figure 17: **SFT pass@k vs. loss curves.**

The SFT-baseline reward R_0 itself (the pre-RL reward mean on the benchmark) shows the same qualitative dependence on L. Fig. 18 plots R_0 against pretraining eval loss for the B3B4 population: an exponential fit gives $R^2 = 0.70$ (with Spearman $\rho = -0.92$, near-perfectly monotone), meaningfully outperforming a linear fit ($R^2 = 0.51$). We also use this fit as the physical floor $R \geq R_0(L)$ in the frontier optimisation of Sec. 3.3, preventing the log-linear RL extrapolation from predicting rewards below the SFT baseline.

G.6 Asymptote Ceiling Fitting

Following Khatri et al. [17], we also estimate the asymptote of RL improvement by fitting a per-run logistic curve,

$$R(C) = R_0 + (A_\infty - R_0)\sigma\big(\gamma(\log_{10} C - \log_{10} C_{\text{mid}})\big),$$

with R_0 fixed at the SFT baseline and $(A_\infty, \log_{10} C_{\text{mid}}, \gamma)$ estimated by weighted nonlinear least squares; 90% confidence intervals are obtained from a 500-sample parametric bootstrap. We restrict this fit to the 20M-parameter family on B1-B4 benchmark, the only sweep whose RL trajectories enter the saturation regime within our compute budget. Resolving A_∞ for larger models would demand roughly an order of magnitude more RL compute per run.

Across the ten 20M runs the estimated ceiling spans $A_\infty \in [0.12, 0.47]$ and is well predicted by the pretraining eval loss L (Spearman $\rho = -0.73$, linear $R^2 = 0.90$): SFT initialisations from better-pretrained checkpoints support strictly higher RL asymptotes. The relationship with $\log_{10}(T/N)$ is weaker but goes in the same direction ($\rho = +0.73$, $R^2 = 0.80$), as expected if L is the proximal mediator of both effects. Fig. 19 visualises these dependencies together with per-run bootstrap CIs.

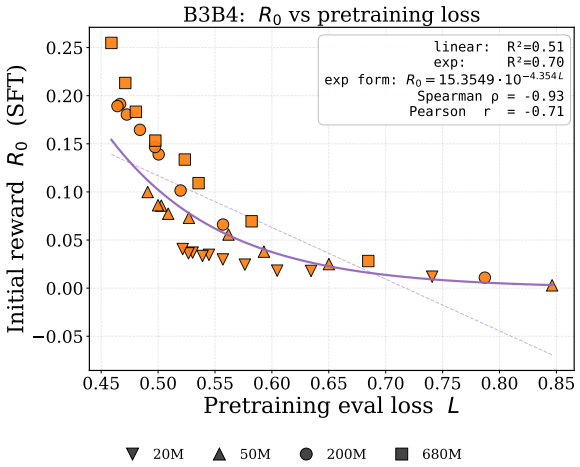


Figure 18: **Initial (post-SFT) reward R_0 versus pretraining eval loss L on the B3B4 benchmark.** Each marker is one of the 36 runs used in the joint-law fit. The exponential fit $R_0 = 14.62 \cdot 10^{-4.30\,L}$ (solid) attains $R^2 = 0.70$; a linear fit (dashed) attains $R^2 = 0.51$. Spearman $\rho = -0.92$. This fit is used as the physical floor $R \geq R_0(L)$ in the frontier optimiser.

Table 11: **Exponential fits of post-SFT pass@k versus pretraining evaluation loss.** The decay rates are broadly consistent with the reference reward and SFT-baseline reward fits. The pretraining loss predicts pass@k with larger k better.

Metric	α	β_{10}	R^2	Spearman ρ
pass@1	1838	-3.80	0.77	-0.93
pass@4	1421	-2.98	0.84	-0.94
pass@8	1060	-2.54	0.87	-0.95
pass@16	789	-2.15	0.89	-0.95

G.7 Extrapolating the Compute-Optimal Frontier

The fitted law lets us score *any* hypothetical training recipe without running it. A recipe is fully specified by a triple (N, T, C_{RL}): the model size N, the number of pretraining tokens T, and the RL compute C_{RL}.

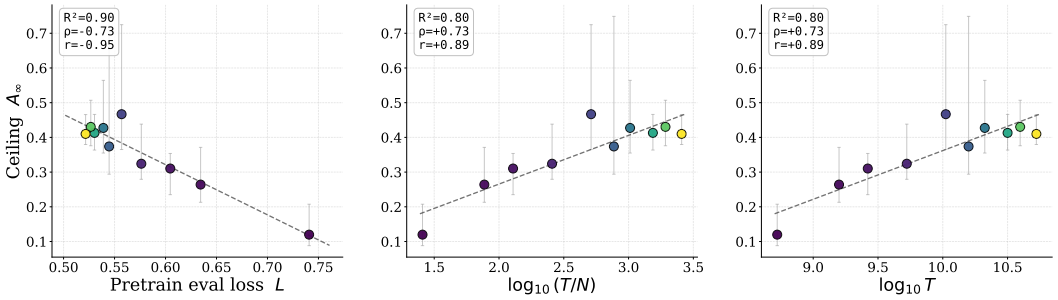


Figure 19: **Asymptote A_∞ of the per-run logistic RL learning curve for the 20M-parameter family, plotted against three pretraining covariates.** Each point is one of the ten 20M runs; markers are coloured by the pretrain-tokens fraction α, vertical bars are 90% parametric-bootstrap CIs (500 resamples per run), and the dashed line is an ordinary-least-squares fit on the displayed sample. **(left)** ceiling versus pretrain eval loss L: better-pretrained checkpoints support strictly higher RL asymptotes (Spearman ρ=-0.73, R^2=0.90). **(middle, right)** ceiling versus $\log_{10}(T/N)$ and $\log_{10} T$: both show a positive trend (ρ=$+0.73$, R^2=0.80 each)

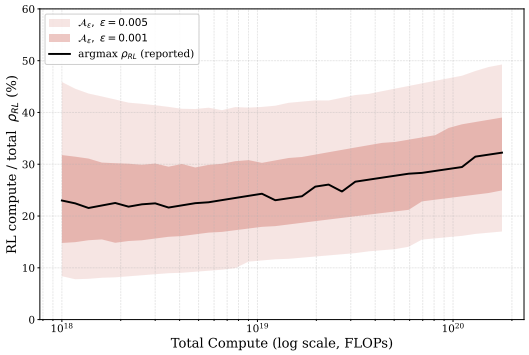


Figure 20: **ϵ-sensitivity of the compute-optimal RL fraction.** Black: argmax ρ_{RL} along the continuous-N optimum. Shaded: the spread of ρ_{RL} over allocations within ϵ reward of the optimum, $\mathcal{A}\epsilon = \widehat{R} \geq \widehat{R}\mathrm{max} - \epsilon$, for $\epsilon = 0.001$ (dark) and 0.005 (light). The reward surface is flat near its peak; however, the whole band shifts upward with compute, so the increasing-RL-share trend is robust.

Per-size frontier. For a fixed model size N and a total-compute budget C, the recipe still has one free degree of freedom: how to split C between pretraining (T) and RL (C_{RL}). We resolve it by maximizing the predicted reward subject to the budget constraint,

$$R_N^\star(C) \;=\; \max_{T,\, C_{\mathrm{RL}}} \widehat{R}(N, T, C_{\mathrm{RL}}) \quad \text{s.t.} \quad C_{\mathrm{tot}}(N, T, C_{\mathrm{RL}}) = C, \tag{11}$$

sweeping T over a dense grid and solving for the residual $C_{\mathrm{RL}} = C - 6NT - C_{\mathrm{SFT}}$. This traces out the compute-optimal reward curve of a single model size as a function of its total budget.

We identify the optimum by grid search over allocations for which N, T, and the implied C_{RL} lie within the ranges supported by the fitted pretraining and RL scaling laws. For each value of C_{tot}, we first evaluate approximately 400 feasible allocations on a coarse grid in $(\log_{10} N, \log_{10} T)$. Since the candidate set is finite, exhaustive enumeration identifies the global maximizer over this grid. We then evaluate a finer grid around the best-performing region of the coarse grid. We report the sensitivity of the frontier in Fig. 20. This refinement reduces discretization error but does not provide an additional guarantee of global optimality over the unrestricted continuous domain. The reported solution is therefore optimal over the evaluated candidate set and within the empirical support of the fitted scaling laws. We do not interpret it as a guarantee of global optimality under extrapolation beyond the compute ranges considered.

Global frontier. We evaluate (11) over a ladder of model sizes, i.e., the sizes we actually trained, augmented with hypothetical fill-in sizes, and take the upper envelope over N at each budget: $R^\star(C) = \max_N R_N^\star(C)$. The size attaining the maximum changes as the budget grows, so the global frontier is a sequence of *takeovers* in which progressively larger models become compute-optimal. For each frontier point we also record the compute-optimal RL share $\rho_{\mathrm{RL}} = C_{\mathrm{RL}}^\star / C$, which quantifies how the budget should be divided between pretraining and RL along the frontier. Taking the continuous limit of the size ladder (optimizing N jointly with T) yields the smooth continuous-N optimum, which we compute at each budget with the Nelder–Mead simplex method [2], initialized from a coarse grid search and warm-started from the previous budget's solution. Fig. 4 shows the resulting frontier and the RL share ρ_{RL} along it. Because our law is fit locally, we restrict the extrapolation to the range of compute we actually observe. Within this range, the global frontier predicted by the law closely tracks the empirical frontier of measured runs, confirming that the fitted law faithfully recovers the compute-optimal trade-off. Along this frontier, the RL-optimal compute share increases from $\sim$20% at 50M to $\sim$28% at 680M.

This suggests that in the lower-compute regime, additional pretraining remains the more valuable use of compute, whereas as the total compute budget grows, RL should receive a proportionally larger share of the allocation.

G.8 Limitations

Several limitations are important for interpreting the fitted law.

First, the law assumes that RL reward is approximately linear in $\log_{10} C_{\mathrm{RL}}$ over the measured compute range. This is an empirical local approximation. It should not be interpreted as evidence that RL improvement is unbounded.

Second, the strict extrapolation setting compounds two sources of error: prediction error in the Chinchilla loss surface and residual error in the f and g maps. The observed-L LOO setting shows that the latter is smaller, but the practical setting requires predicting L from (N, T).

Third, $g(N, T)$ is materially noisier than $f(L)$. The reference reward is strongly controlled by pretraining loss, whereas the RL slope has additional unexplained variation. This may reflect optimizer details, SFT/RL data differences, reward-model variation, or other run-level factors not included in the current parameterization.

Finally, the data are densest near the observed model sizes, token counts, and RL-compute range. The frontier analysis is therefore best viewed as a diagnostic for allocation trends under the fitted assumptions, not as a claim that the same exponents hold arbitrarily far beyond the training distribution.

H Move Policy Evolution

H.1 From Token Space to Move Space: Policy Evolution Metrics

Although training is performed in token space, all policy-evolution metrics are defined in move space. This is possible because each valid token prefix corresponds to a legal board state, and each legal move at that state has a token serialization.

Induced move policies. For pretraining, define the raw move score $\widetilde{\pi}_{\theta_{\mathrm{pre}}}(a \mid s) = \prod_{j=1}^{|\tau(a)|} p_{\theta_{\mathrm{pre}}}(x_j^{(a)} \mid s, x_{<j}^{(a)})$ and the induced move policy $\pi_{\theta_{\mathrm{pre}}}(a \mid s) = \frac{\widetilde{\pi}_{\theta_{\mathrm{pre}}}(a|s)}{\sum_{a' \in \mathcal{A}(s)} \widetilde{\pi}_{\theta_{\mathrm{pre}}}(a'|s)}$.

For post-training stages $m \in \{\mathrm{sft}, \mathrm{rl}\}$, the trajectory policy $\pi_{\theta_m}(\zeta \mid s) = \pi_{\theta_m}(r, a_1, o_1, \ldots, a_T \mid s)$ induces a root-conditioned trace policy $\pi_{\theta_m}(r \mid s_0)$. Conditional on a fixed trace r, define the raw move score $\widetilde{\pi}_{\theta_m}(a \mid s, r) = \prod_{j=1}^{|\tau(a)|} p_{\theta_m}(x_j^{(a)} \mid s, r, x_{<j}^{(a)})$ and the conditional move policy $\pi_{\theta_m}(a \mid s, r) = \frac{\widetilde{\pi}_{\theta_m}(a|s,r)}{\sum_{a' \in \mathcal{A}(s)} \widetilde{\pi}_{\theta_m}(a'|s,r)}$. Marginalizing over traces gives the root-conditioned marginal move policy $\pi_{\theta_m}(a \mid s) = \sum_r \pi_{\theta_m}(r \mid s)\, \pi_{\theta_m}(a \mid s, r)$. Since enumerating all possible reasoning traces is intractable, we estimate the marginalized move policy by Monte Carlo sampling in practice. For each prompt, we sample $K = 128$ rollouts from the trace policy $\pi_{\theta_m}(r \mid s_0)$ and approximate the marginal as

$$\widehat{\pi}_{\theta_m}(a \mid s) = \frac{1}{K} \sum_{k=1}^{K} \pi_{\theta_m}(a \mid s, r^{(k)}), \qquad r^{(k)} \sim \pi_{\theta_m}(r \mid s_0).$$

H.2 Fitting Power-Sharpening Transformations

We use two complementary diagnostics to test whether RL primarily transforms the SFT marginal policy by simple probability sharpening. For a state s, let

$$p_s(a) = \pi_{\mathrm{sft}}(a \mid s), \qquad q_s(a) = \pi_{\mathrm{rl}}(a \mid s),$$

where $a \in \mathcal{A}(s)$ denotes a legal move. Given a coefficient α, define the α-power transform of the SFT policy as

$$p_{s,\alpha}(a) = \frac{p_s(a)^\alpha}{\sum_{b \in \mathcal{A}(s)} p_s(b)^\alpha}.$$

When $\alpha > 1$, this transform sharpens the SFT distribution by increasing relative mass on high-probability moves; when $\alpha < 1$, it flattens the distribution.

Table 12: **Estimated global α and α per state.** α^* appears to increase across RL stages, but the widening IQR suggests substantial variability across samples.

Stage	α^* (global)	α^* (median [IQR])	ExplainedSharp (global)	ExplainedSharp (median [IQR])
pretrain → SFT	0.57	0.63 [0.44, 0.88]	-0.01	0.09 [-0.01, 0.26]
SFT → RL_50	1.05	1.12 [0.91, 1.41]	0.03	0.16 [0.02, 0.54]
SFT → RL_100	1.15	1.25 [0.98, 1.64]	0.09	0.24 [0.03, 0.68]
SFT → RL_250	1.27	1.47 [1.07, 2.06]	0.16	0.42 [0.07, 0.84]
SFT → RL_500	1.27	1.52 [1.03, 2.39]	0.14	0.39 [0.04, 0.88]
SFT → RL_750	1.35	1.72 [1.14, 2.81]	0.17	0.49 [0.08, 0.93]

KL power fit. The first diagnostic fits a single global sharpening coefficient by projecting the RL policy onto the SFT power family. Specifically, we choose

$$\alpha^\star = \arg\min_{\alpha\in[0,\alpha_{\max}]} \sum_s w_s \, D_{\mathrm{KL}}(q_s \,\|\, p_{s,\alpha}) \,,$$

where w_s is a state weight. By default, we use uniform weighting, $w_s = 1/N$ for N states.

The fitted value $\alpha^\star$ measures the degree of distribution-level sharpening. If $\alpha^\star > 1$, the best power approximation sharpens the SFT policy toward the RL policy. The remaining divergence measures how much of the RL update is not explained by simple power sharpening. We summarize fit quality using

$$\mathrm{ExplainedSharp} = 1 - \frac{\sum_s w_s \, D_{\mathrm{JS}}(q_s, p_{s,\alpha^\star})}{\sum_s w_s \, D_{\mathrm{JS}}(q_s, p_s)}.$$

A high value of $\mathrm{ExplainedSharp}$ indicates that the RL policy is well approximated by a power-sharpened SFT policy. A low value indicates that RL reshapes the distribution in ways not captured by a uniform sharpening transform.

Centered-logit linear fit. The second diagnostic tests the logit-geometry implied by power sharpening. Taking logs of the power transform shows that, up to a state-dependent normalization constant, power sharpening scales centered log-probabilities linearly. For each state, define

$$x_s(a) = \log p_s(a) - \frac{1}{|\mathcal{A}(s)|} \sum_{b\in\mathcal{A}(s)} \log p_s(b),$$

and

$$y_s(a) = \log q_s(a) - \frac{1}{|\mathcal{A}(s)|} \sum_{b\in\mathcal{A}(s)} \log q_s(b).$$

We then fit the linear relation

$$y_s(a) \approx \beta x_s(a)$$

across states and actions, and report the fitted slope β and coefficient of determination R^2. A slope $\beta > 1$ indicates sharpening in centered log-probability space, while the R^2 measures how well the RL policy is explained by a pure scaling of the SFT logits.

We fit the results on the 50M model pretrained with 4.6B tokens. Table 12 and Table 13 show the fitting results. Both α^* and β tend to increase during RL, though their distributions reveal substantial sample-level heterogeneity.

H.3 Policy Categorization

Table 14 gives the formal definitions for all categories we consider. Intuitively, the categories distinguish the following update types. *Ground-truth amplification* captures cases where the correct move is already top-ranked and is further reinforced. *Tail discovery* captures cases where training promotes a correct move from the low-probability tail into the top-k set. *Top-k correction* captures promotion of a correct move that was initially plausible but not top-ranked. *Ground-truth regression* captures cases where a previously top-k correct move is demoted. *Wrong-mode amplification* captures cases where the correct move remains outside the top-k set while the initially preferred wrong move is further reinforced. All remaining transitions are grouped as *Other*. We set ϵ_{tail} as 0.05. We fit the results on the 50M model pretrained with 4.6B tokens.

Table 13: **Estimated global β and β per state.**

Stage	β (global)	β (median [IQR])	R^2 (global)	R^2 (median [IQR])
pretrain → SFT	0.60	0.62 [0.47, 0.76]	0.40	0.44 [0.30, 0.58]
SFT → RL_50	0.99	1.03 [0.89, 1.16]	0.68	0.75 [0.62, 0.85]
SFT → RL_100	1.03	1.07 [0.92, 1.21]	0.63	0.71 [0.57, 0.81]
SFT → RL_250	1.07	1.12 [0.94, 1.29]	0.60	0.68 [0.53, 0.80]
SFT → RL_500	1.06	1.12 [0.94, 1.29]	0.57	0.65 [0.49, 0.77]
SFT → RL_750	1.13	1.18 [0.99, 1.38]	0.56	0.63 [0.48, 0.76]

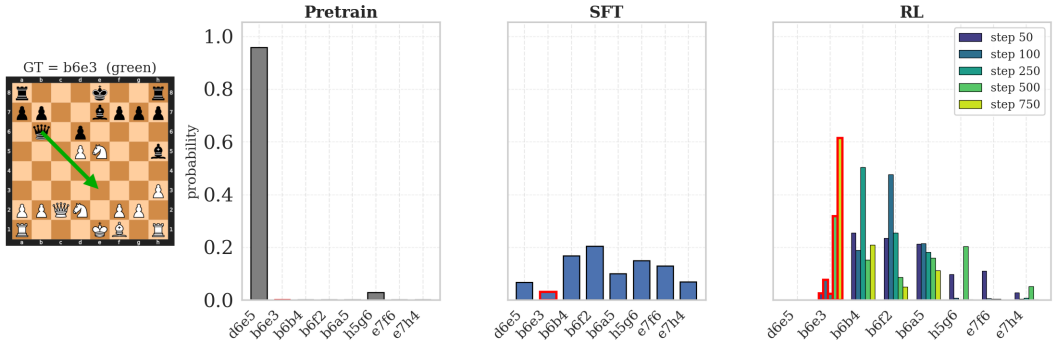


Figure 21: **Policy evolution on a hard puzzle (B5) across training stages.** Left: board position with ground-truth move b6e3 (green arrow). Pretrain: the model assigns $\sim$95% mass to a wrong move (d6e5), with b6e3 nearly absent. SFT: probability spreads across many candidates but b6e3 (red) remains low. RL: over training steps 50–750, b6e3 rises to become the top move (tail discovery), while a competing wrong move (b6b4) also retains significant mass (wrong-mode amplification).

H.4 CoT Evolution Analysis

As a complementary lens, we examine how the structure of reasoning traces evolves over the course of RL training. Because our CoT format comprises explicit move sequences, each rollout can be reconstructed as a prefix tree rooted at the puzzle state (Section 2.2), with each node corresponding to a move, enabling us to probe both the structure and quality of the model's reasoning.

We characterize properties of reasoning traces along three axes: (1) *Search shape* defines structural properties of the parsed tree: number of nodes, maximum depth D, average branching factor, and the width-to-depth ratio $|L|/D$, where $|L|$ is the number of leaves. (2) *Move quality* scores candidate moves against Stockfish via the normalized rank $(r-1)/(n-1)$, where $r \in \{1, \ldots, n\}$ is the Stockfish rank of the move (with $r = 1$ best) among the $n \geq 2$ legal moves at that position; lower values indicate stronger moves. We report this metric separately for the player's first-move candidates and the model's proposed opponent replies. (3) *Search behavior* captures traversal patterns by measuring the consistency with DFS and fractions of revisiting nodes. All metrics are computed per rollout, averaged across rollouts within the same prompt, and then aggregated across prompts.

Structured CoT exposes fine-grained search dynamics and weakness in deeper search. In Fig. 22, we compare two representative RL runs for the 20M and 50M models pretrained under matched compute. Additional metrics are shown in Fig. 24 and Fig. 25 in Appendix H.4. These structured-CoT metrics provide a finer-grained view of search behavior than reasoning-token counts alone (Fig. 23). The parsed search traces show that, on average, the models primarily expand search breadth rather than depth, as the width-to-depth ratio and branching factor increase while maximum search depth remains roughly flat. The 20M model tends to propose more distinct candidate moves over training, which may also help explain the improvement in its pass@k. Meanwhile, the quality of moves proposed in the CoT improves for both the model's own moves and its predicted opponent responses, with larger gains on the model-move side. The model also becomes more likely to mention the ground-truth move in its CoT and to commit to the best candidate it has considered. Interestingly, the generated search traces become less aligned with a strict DFS serialization order, showing more revisits to previously considered lines. Despite these improvements, Fig. 25 reveals that

Table 14: Taxonomy of policy-update effects based on changes in top-k membership and probability between the initial policy π_{θ_0} and trained policy π_{θ_1}.

Category	Description	Condition
Ground-truth amplification	The ground-truth move remains in the top-k set and gains probability.	$a^\star(s) \in \mathcal{T}^k_{\theta_0}(s), a^\star(s) \in \mathcal{T}^k_{\theta_1}(s), \Delta p(a^\star; s) > 0.$
Tail discovery	A low-probability ground-truth move is promoted into the top-k set.	$a^\star(s) \notin \mathcal{T}^k_{\theta_0}(s), a^\star(s) \in \mathcal{T}^k_{\theta_1}(s),$ $\pi_{\theta_0}(a^\star(s) \mid s) < \epsilon_{\text{tail}}.$
Top-k correction	A non-tail ground-truth move is promoted into the top-k set.	$a^\star(s) \notin \mathcal{T}^k_{\theta_0}(s), a^\star(s) \in \mathcal{T}^k_{\theta_1}(s),$ $\pi_{\theta_0}(a^\star(s) \mid s) \geq \epsilon_{\text{tail}}.$
Ground-truth regression	A previously top-k ground-truth move is demoted out of the top-k set.	$a^\star(s) \in \mathcal{T}^k_{\theta_0}(s), \quad a^\star(s) \notin \mathcal{T}^k_{\theta_1}(s).$
Wrong-mode amplification	The ground-truth move remains outside the top-k set, while the initial top-1 wrong move is reinforced.	$w_{\theta_0}(s) = \arg\max_a \pi_{\theta_0}(a \mid s),$ $a^\star(s) \notin \mathcal{T}^k_{\theta_0}(s), a^\star(s) \notin \mathcal{T}^k_{\theta_1}(s),$ $w_{\theta_0}(s) \in \mathcal{T}^k_{\theta_1}(s), \quad \Delta p(w_{\theta_0}; s) > 0.$
Other	All remaining cases, including stable top-k structure, switches among wrong modes, and partial changes that do not promote the ground-truth move into the top-k set.	Otherwise.

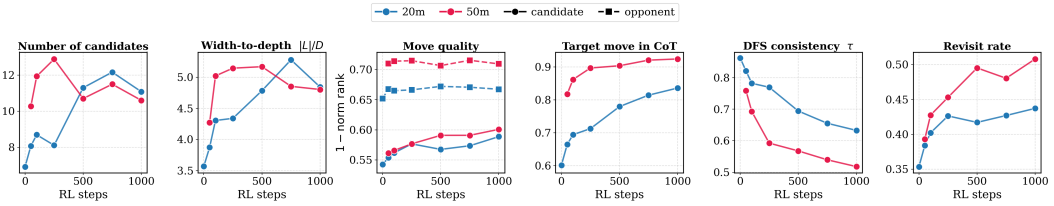


Figure 22: **Structured reasoning traces evolve during RL training.** For the 20M and 50M models, panels report per-step prompt means on the puzzle benchmark. RL modestly increases search breadth, improves move quality scored by Stockfish rank for both model moves (solid) and predicted opponent replies (dashed), and increases ground-truth move coverage in the parsed CoT tree. DFS consistency decreases and revisit rate increases, indicating more re-exploration of earlier prefixes.

the model still struggles to recover continuations that require deeper search, suggesting that current RL training may improve candidate generation and selection faster than it improves long-horizon search. Understanding how these structured search features affect performance may help guide future SFT data filtering and construction toward examples that encourage deeper, more systematic search.

H.5 Entropy

We report policy entropy over the induced move distribution and token entropy over CoT tokens below (for the 50M model pretrained with 4.6B tokens). Policy entropy drops substantially (1.370 $\rightarrow$ 0.705), while CoT token entropy decreases modestly (0.536 $\rightarrow$ 0.492), indicating RL concentrates move selection with a smaller reduction in token-level variability. This concentration is heterogeneous across states: the policy-entropy distribution remains broad, and harder B3–B5 puzzles retain higher entropy than easier B1–B2 puzzles. These results support our conclusion that RL does not sharpen uniformly across states.

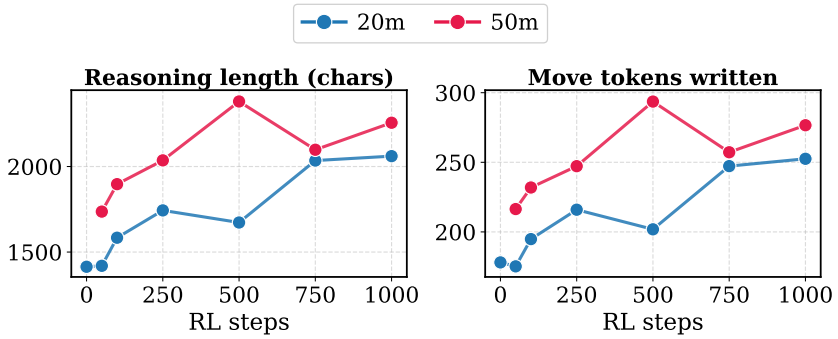


Figure 23: **Reasoning length dynamics over RL training.**

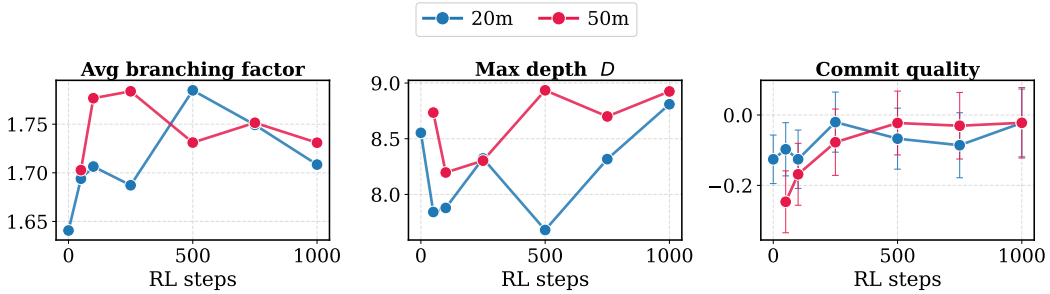


Figure 24: **Search shape and selected move quality over RL training for the 20M and 50M models**. Curves show mean metrics for the 20M and 50M models, with standard error across prompts for the selected moves quality. Columns report effective branching factor $b_{\text{eff}} = N^{1/D}$, maximum tree depth D, and commit quality, a per-prompt z-score measuring the Stockfish rank of the committed move relative to both the best move and the best considered move. Branching increases modestly, depth decreases slightly, and commit quality improves monotonically.

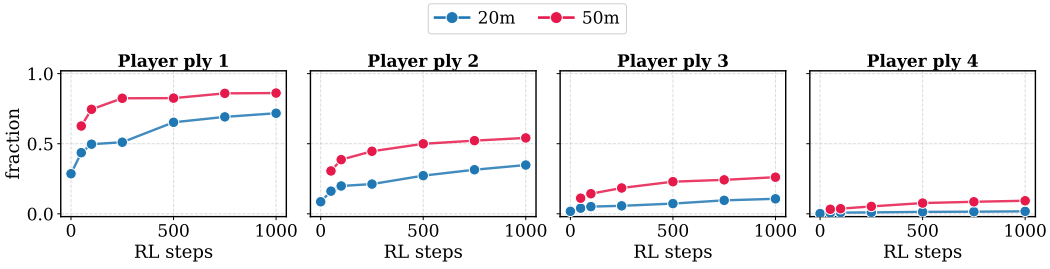


Figure 25: **Search-tree coverage of ground-truth continuations by depth for the 20M and 50M models.** For each rollout, we measure whether the parsed CoT search tree contains a root-to-depth-k path whose player moves match the first k target plies, allowing arbitrary legal opponent replies. Panels report coverage for $k = 1, 2, 3, 4$, excluding rollouts with shorter targets. RL improves coverage at all depths, but gains decay sharply with depth; even at step 1000 for the 50M model, few rollouts recover the full 4-ply target line.

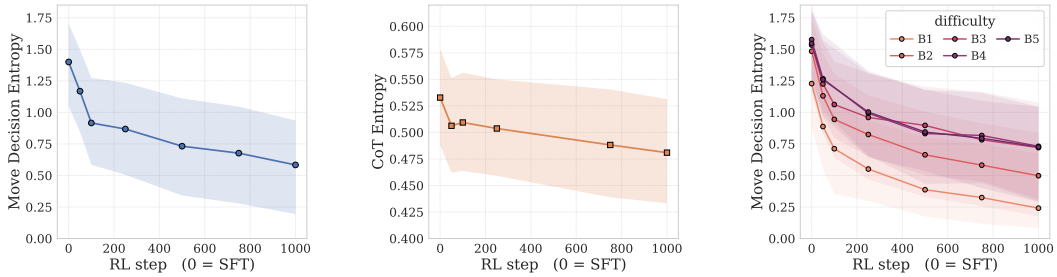


Figure 26: **Entropy dynamics during RL training.**

I Olmo Experiment Additional Details

I.1 Implementation Details

We pretrain a 1B-parameter OLMo-2 language model, fork intermediate checkpoints as pretraining-scale anchors, anneal each to convergence, supervised fine-tune on math reasoning traces, and finally run GRPO reinforcement learning. Architecture is fixed across all runs; the only variable across anchors is the number of pretraining tokens. Tables 15 and 16 summarize the setup.

Table 15: Model architecture. Identical for every pretraining anchor, supervised fine-tuning, and reinforcement-learning run.

Architecture (OLMo-2)	**Value**
Total parameters	1.48 B
Non-embedding parameters	1.07 B
Layers	16
Hidden size	2048
FFN intermediate size	8192
Attention heads	16
KV heads (no GQA)	16
Vocabulary size	100,278
Max position embeddings	8192
Positional encoding	RoPE ($\theta = 10^4$)
Tied input/output embeddings	No
Precision	bf16

I.2 Additional Results

Fig. 27 reports the fitting results on downstream benchmarks.

Table 16: Hyperparameters for each stage of the pipeline. Pretraining uses a warmup–stable–decay (WSD) schedule; intermediate stable-phase checkpoints (every 5,000–10,000 steps) define the pretraining-scale anchors. Each anchor is annealed independently (linear LR decay to zero over 5 B tokens) before supervised fine-tuning and reinforcement learning. RL uses GRPO (no KL penalty, entropy coefficient 0, dual-clip ratios $0.2/0.26$, $c = 10$) with 8 rollouts per prompt at temperature 1.0.

	Pretrain (stable)	**Anneal**	**SFT**	**RL (GRPO)**
Data	Dolma3/Dolmino mix	(same corpus)	NuminaMath-CoT	GSM8K + MATH + DeepScaler mix
Examples / tokens	200 B tokens	5 B tokens / anchor	859,490 ex. (≈0.46 B tok)	up to 3000 steps
Sequence length	4096	4096	4096 (cutoff)	512 prompt / 3584 resp.
Global batch	512 seq (≈2.1 M tok)	512 seq	512 examples	128 prompts × 8
Optimizer	AdamW	AdamW	AdamW	AdamW
(β_1, β_2)	(0.9, 0.95)	(0.9, 0.95)	—	—
Weight decay	0.033 (0 on emb.)	0.033	—	—
Peak LR	4×10^{-4}	$4 \times 10^{-4} \to 0$	1×10^{-5}	1×10^{-6}
LR schedule	WSD (warmup+const)	linear decay to 0	cosine	constant
Warmup	2 B tokens	—	3% of steps	50 steps
Epochs / steps	95,368 steps	—	1 epoch (1679 steps)	3000 steps
Packing	—	—	No (one ex. / seq)	—
Loss mask	all tokens	all tokens	assistant only	—

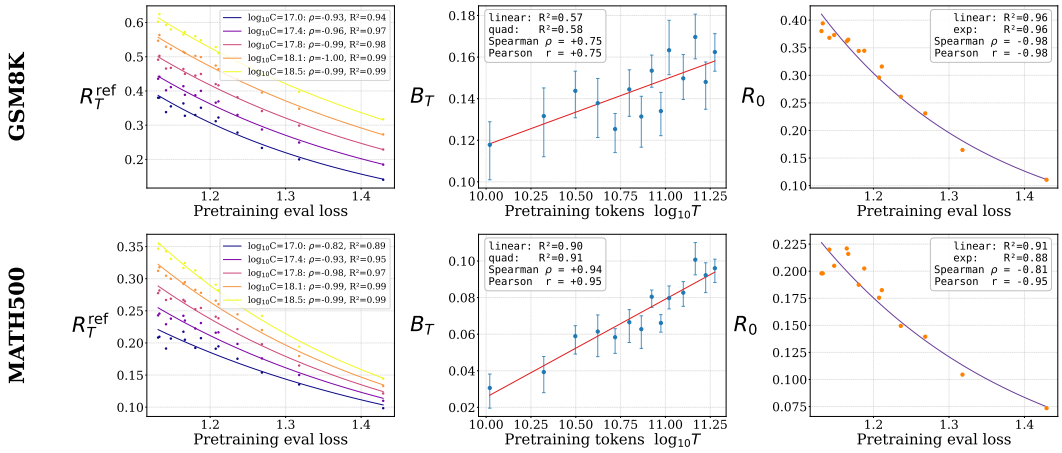


Figure 27: **RL local performance and slope with pretraining properties on GSM8K and MATH500 for pretrained 1B models.**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Three claims: (1) pretraining loss predicts post-RL performance, (2) pretraining data scale predicts RL improvement rate, (3) RL reshapes the SFT policy non-uniformly across difficulty. All supported by experiments across different model scales (§3).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.

- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not claim theoretical results.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Model architectures, datasets, and training configurations for all stages detailed in Section 3 and Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code, models and datasets have been released. Data sourced from publicly available Lichess database.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See Section 3 and Section C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Conduct multiple runs to calculate pass@k for multiple k values. Leave-one-out validation metrics reported for scaling law fits.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: See Section C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: Uses publicly available chess data; no human subjects or dual-use concerns.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Practical takeaways discussed in Section 7. This work does not pose any negative societal impact.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Lichess, Stockfish, Qwen3, and verl are cited with URLs throughout the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Chess benchmark and puzzle environment documented in Section 2. Code will be released.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No crowdsourcing or human subjects involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No human subjects research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: Core method trains chess-specific models from scratch; no external LLMs used as methodology components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.

- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.