Installing and Obstructing Heuristics: Learning Dynamics in Nim

Leo Villani [1 *] **Sultan Daniels** [2 *] **Ijin Yu** [2 *] **Anant Sahai** [2]

Abstract

The algorithmic capabilities of large language models may be driven by simple heuristics or spurious shortcuts. We study this phenomenon via Nim, a controlled natural language reasoning task in which the exact optimal strategy is known and the space of plausible shortcuts is explicit. In this setting, natural coarsenings of the target strategy correspond to coset structures of modular reduction, allowing us to directly observe when model performance plateaus at an intermediate heuristic during training. We show that these intermediate heuristics can be selectively implanted or circumvented through curriculum learning.

Code availability. Code is available at https://github.com/ijinyu1113/nim-heuristics.

1. Introduction

Modern language models often display behavior that appears algorithmic: they can answer arithmetic questions, follow procedural patterns, and generate outputs that resemble structured reasoning. Liu et al. (2023) prove that shallow transformer-based models are expressive enough to simulate finite-state automata on a bounded length input, yet, in trained models, the appearance of strong reasoning can mask a strategy based on heuristics rather than the intended rule (Nikankin et al., 2025). Rather than treating such heuristics purely as failures, we argue they offer a window into understanding how models develop strategies throughout training.

In many natural benchmarks, the target rule and the space of plausible heuristics are not explicitly known. Developing controlled settings in which these internal mechanisms can be isolated is therefore important for understanding when neural models genuinely acquire algorithmic structure.

*Equal contribution [1]Cornell University, Ithaca, NY, USA [2]UC Berkeley, Berkeley, CA, USA. Correspondence to: Leo Villani <lv259@cornell.edu>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. Copyright 2026 by the author(s).

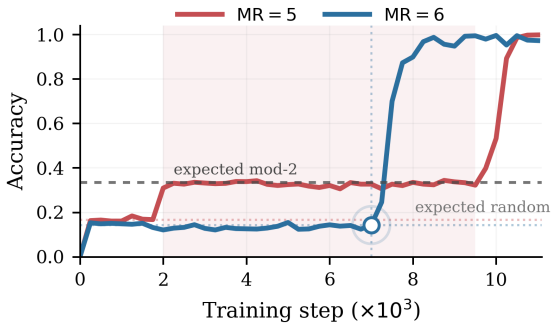

Figure 1. Learning dynamics for two 410M finetuning runs with different downstream single-pile bounded Nim moduli (see Section 2). The $\mathrm{MR} = 5$ task has modulus $m = 6$, which admits parity coset heuristics of the optimal strategy; the $\mathrm{MR} = 6$ task has modulus $m = 7$, which does not admit any nontrivial coset structure. Solid curves show exact move accuracy over training. Faint dotted colored horizontal lines mark random-guessing accuracy for each task, and the gray dashed horizontal line marks the accuracy level expected from a mod-2 coset heuristic in the $\mathrm{MR} = 5$ task. The $\mathrm{MR} = 5$ run enters a long plateau near this heuristic accuracy level before eventually learning the optimal strategy, whereas the $\mathrm{MR} = 6$ run remains near random accuracy and then transitions sharply to high accuracy without a comparable intermediate plateau.

Many works have studied the numerical representations and arithmetic capabilities of pretrained (Levy & Geva, 2025; Kantamneni & Tegmark, 2025) and finetuned language models (Zhou et al., 2024), finding particular mechanisms for addition and establishing the importance of the token embeddings for a model's successful performance of the task. Another line of work studies the algorithmic capabilities of neural networks by training models from random initialization to explicitly perform modular arithmetic (Power et al., 2022; Liu et al., 2022; Nanda et al., 2023a; Gromov, 2023; Doshi et al., 2024; Zhong et al., 2023; Mohamadi et al., 2024; Stander et al., 2024; McCracken et al., 2025). For a task like addition mod n, an exact coset is a set of numbers that have the same value mod k where k is a factor of n. Recently, McCracken et al. (2025) provide evidence that neural networks across various architectures solve modular addition by composing approximate cosets and taking their intersection.

Nonetheless, McCracken et al. (2025) focus on the final

learned solutions of models that have developed the ability to perform modular arithmetic. Although Nanda et al. (2023a); He et al. (2026) develop proxy progress measures for the generalization capabilities of models, they do not specify the partial solution the models are implementing. In this work, we show that models often plateau during training at single coset heuristics before reaching the optimal strategy. Our main contribution is that through curriculum learning, we show that plateaus at these intermediate coset heuristics can be selectively induced in a model's training dynamics, or the duration of a plateau can be selectively minimized.

Furthermore, coset heuristics can be transferred from finetuning on explicit modular reduction ($n \mapsto n \bmod m$), to the learning dynamics of later finetuning on a synthetic natural language task with implicit modular structure. This synthetic task is *single-pile bounded Nim*, a simple impartial two-player game in which a player may remove between 1 and $\mathrm{MR} \in \mathbb{N}$ coins from a pile, and the player taking the last coin wins. Other works have studied games like Othello and chess, primarily to probe the world models of neural networks and see how they encode board state (Li et al., 2024; Nanda et al., 2023b; Karvonen, 2024). In bounded Nim, the world model is modular reduction. The optimal strategy of bounded Nim is known, but so are its natural coarse cosets: for composite moduli, a model might first learn a mod-k partition (where k is a factor of m) of the full residue class structure before learning the optimal strategy. This is depicted in Figure 1.

2. Bounded Nim and Coset Heuristics

We study single-pile bounded Nim, a game where two players take turns removing objects from a pile, and the player who removes the last object wins. At each turn, if there are n objects in the pile, a player can remove between 1 and $\min(n, \mathrm{MR})$ objects from the pile where $\mathrm{MR} \in \mathbb{N} \setminus \{0\}$.

Let $a(n) \in \{1, \ldots, \min(n, \mathrm{MR})\}$ denote the strategy of removing $a(n)$ objects from an n object pile and let $m = \mathrm{MR} + 1$. Under normal play, the losing states are exactly those with $n \equiv 0 \pmod{m}$: from any nonzero residue, i.e. $n \equiv x \pmod{m}$ where $x \neq 0$, a player can move to the nearest smaller multiple of m, and thereafter return the game to residue $0 \pmod{m}$ after each opponent move. Thus the unique optimal move on any winning state is

$$a^*(n) = n \bmod m. \tag{1}$$

We label losing states with a special token -1.

Because the Nim strategy is modular, we can understand the expected accuracies of simple predictors through a guessing-game analogy. A predictor with no information that chooses uniformly among m options is correct with probability $1/m$. A predictor that applies a *coset heuristic* improves on this baseline by narrowing the set of plausible answers. Following McCracken et al. (2025), a coset is a set of elements with strict modular equivalence. For example, in the $m = 6$ setting, mod-2 equivalence induces the cosets $\{0, 2, 4\}$ and $\{1, 3, 5\}$. If the predictor knows that the correct label lies in a coset C, but cannot distinguish the residues within C, uniformly random guessing within C gives accuracy $1/|C|$.

The same expected accuracy is achieved by a deterministic heuristic that always chooses one canonical label $c \in C$, over many trials of random games whose true answers are distributed uniformly. This coset structure, when d is a factor of m, partitions the m residues into d cosets of size m/d, giving an expected accuracy ceiling of d/m for uniformly random games as opposed to the $1/m$ baseline. It now becomes clear why a mod-d heuristic may be attractive. Hence, plateaus at these levels can be thought of as indicating that the model has learned a coarse modular structure, but not the optimal Nim strategy.

Figure 1 illustrates why these coarse cosets matter for the learning dynamics. The $\mathrm{MR} = 5$ task has modulus $m = 6$, so the optimal strategy admits nontrivial parity cosets: residues can first be grouped into even and odd classes before the model distinguishes all six residues. By contrast, the neighboring $\mathrm{MR} = 6$ task has modulus $m = 7$, which is prime and therefore has no analogous nontrivial cosets. Empirically, the $\mathrm{MR} = 5$ run enters a long plateau near the accuracy predicted by this coarse coset heuristic, while the $\mathrm{MR} = 6$ run transitions more abruptly from chance-level behavior to the full optimal strategy.

3. Experimental Setup

We fine-tune causal language models from the open-source Pythia deduped family (Biderman et al., 2023) at scales 70M, 160M, and 410M, on synthetic bounded Nim datasets written in natural language. Each prompt describes a game state using a fixed verbal template that specifies the total number of objects, the move limit, and a short history of prior moves; an example is given in Figure 17. Evaluation prompts are verified to be distinct from all training prompts to prevent data contamination. Future work will expand experimentation on evaluation datasets where specific quantitative aspects of the data are held-out, like the initial total number of objects, and the current number of objects after the turn history. Models are trained with full-model supervised fine-tuning using the HuggingFace `Trainer`, applying cross-entropy loss on the answer continuation only (prompt tokens are masked). The model is trained to generate the correct action in text form, e.g. `take 3 coins`, with `take -1 coins` used for losing states. All sequences are padded to a maximum length of 128 tokens using the EOS token as the padding token.

We train for up to 300 epochs with AdamW, train and evaluation batch size 64, learning rate 3×10^{-5}, weight decay 0.05, warmup ratio 0.1, and a cosine learning-rate schedule on one A40, A100, or GH200 GPU (Boerner et al., 2023). We find that different hyperparameter choices produce qualitatively different training dynamics, including extended plateaus and prolonged random-level performance; these effects are characterized further throughout the paper and in the appendix. Training and evaluation datasets are balanced across residue classes modulo $m = \mathrm{MR} + 1$, so chance exact accuracy is $1/(\mathrm{MR} + 1)$.

At each checkpoint, we decode greedily and parse the predicted move. Our main metric is exact match accuracy. To identify intermediate coset heuristics, we additionally report coarsened accuracies for integer factors of the task modulus, together with residue-level confusion matrices and prediction distributions. These diagnostics distinguish optimal strategy learning from partial coset heuristics. Experiments on basic Nim finetuning in Section 4.1, coset heuristic transfer between distinct tasks in Section 4.2, curriculum learning in Section 4.3, and shortcut-intervention experiments in Appendix A reuse this basic setup, with condition-specific details deferred to their corresponding sections and the appendix.

4. Main Empirical Picture

4.1. Models often learn coset heuristics when finetuned for Nim

In Figure 2, each solid curve shows the median accuracy across three seeds during the finetuning of a Pythia model on a synthetic Nim dataset where all training examples have the same value for MR. Notice that for each value of MR, the evaluation accuracy first plateaus at the expected accuracy of random guessing. Then, for the MR values that correspond to an optimal strategy with a prime modulus, the evaluation accuracy abruptly reaches perfect performance. However, for the MR values that correspond to an optimal strategy with an even modulus, the evaluation accuracy often plateaus at its coset heuristics. Notably, unlike the larger models, the 70M model plateaus only at a coset heuristic for $\mathrm{MR} = 3$, suggesting that susceptibility to these plateaus may depend on model scale.

The coset heuristics are directly visible at the residue level. Figure 3 shows confusion matrices from an illustrative 410M run on the $\mathrm{MR} = 7$ task, where the model first collapses to two canonical moves, then organizes its predictions by parity, and later consolidates into a mod-4 heuristic. Rather than making arbitrary errors, the model increasingly avoids cross-coset predictions: by the later checkpoints, mistakes are concentrated within coset-aligned blocks, producing a clear diagonal structure. This provides a direct behavioral signature that the plateau corresponds to a learned coset heuristic rather than unstructured partial accuracy.

Additional diagnostics in Appendix J.2 further corroborate that these models learn coset heuristics. In Appendix J.3, we show that during multitask finetuning across multiple values of MR, models using coset heuristics often collapse each coset to a single canonical predicted residue rather than spreading probability across all residue-consistent moves. Figure 31 also shows that this distributional view can reveal progress hidden by the scalar accuracy curve: on the $\mathrm{MR} = 7$ task, the model remains near a sustained 25% exact-accuracy plateau, but its predictions become increasingly uniform within the relevant coset structure over training. We therefore view these prediction distributions as a possible progress measure for partial grokking (Nanda et al., 2023a), which we leave for future work.

Lastly, in Appendix J.1, we look into the model internals using the Logit Lens technique (nostalgebraist, 2020) on each layer's Attention, MLP, and residual stream components for various checkpoints throughout training. In Figure 24, we find that when the model's accuracy plateaus at the level of a mod-2 coset heuristic, the later layers of the residual stream show a clear representation of parity.

4.2. Coset heuristics can transfer between distinct tasks

Coset structure learned in one task can persist and dominate behavior in another. Coset structure can be installed in a model by *prefinetuning*, first finetuning on data whose underlying modulus is a factor of the modulus of the intended downstream task before finetuning on the downstream task itself. We study this in two settings: prefinetuning on Nim data, and prefinetuning on explicit modular reduction data (e.g. "what is $10 \bmod 4$?"). Table 1 gives a compact summary of the main transfer outcomes across the downstream tasks.

For the experiments in this section, we use 410M-parameter models. After the prefinetuning stage, each model is finetuned on the downstream Nim task for 150 epochs. This differs from the longer experiments in Figure 2; under this shorter horizon, the baseline often does not reach the optimal strategy. For explicit modular prefinetuning, we generate modular reduction datasets over inputs in the range $[0, 10000]$, randomly split into 9000 training examples and 1000 evaluation examples. These prompts are generated algorithmically, rather than by a language model to ensure there is no risk of subliminal learning (Cloud et al., 2026; Schrodi et al., 2026). Each explicit Mod$k$ prefinetune is trained for 5000 steps, by which point it reaches 100% evaluation accuracy. For Nim$k$ prefinetuning, we instead initialize from 410M Nim checkpoints: for each source task, we choose a seed and take the first checkpoint that reaches 100% evaluation accuracy, which occurs at different training

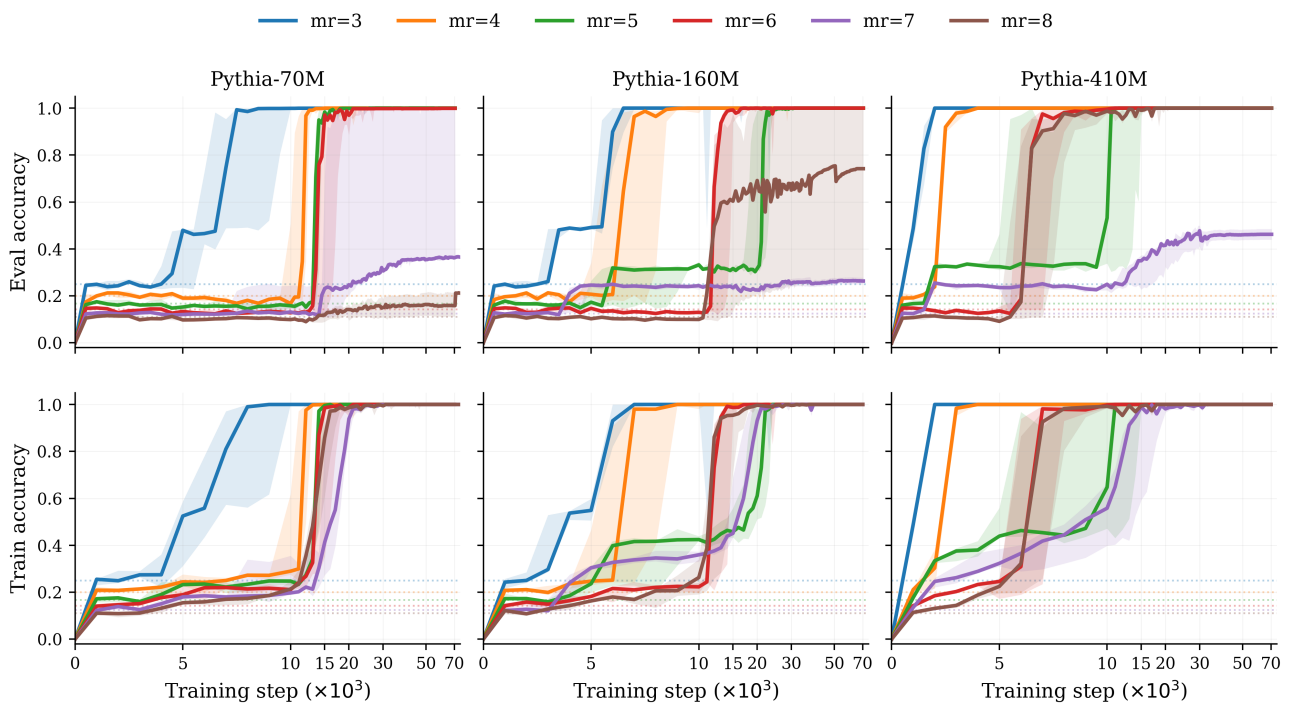

Figure 2. Finetuning Pythia models on bounded Nim tasks across model sizes and three random seeds for 300 epochs. **Left Column:** Pythia-70M, **Middle Column:** 160M, and **Right Column:** 410M. **Top Row:** Evaluation accuracy, **Bottom Row:** Training accuracy. Each color represents a Nim dataset of 15000 prompts with a corresponding value of MR. Evaluation datasets contain 2000 prompts. The dark curve shows the median-accuracy seed and the shaded region spans the minimum and maximum across all three seeds. The faint dashed horizontal lines denote the expected accuracy of random guessing for each MR. Notice in evaluation curves that for each MR the accuracy first plateaus at random guessing. For the 160M and 410M models the MR curves for composite moduli plateau at accuracies corresponding to coset heuristics before, in some runs, refining to the optimal strategy. For prime moduli, intermediate plateaus are not present. For the 70M models, $\mathrm{MR} = 3$ is the only value that exhibits coset heuristic plateaus.

steps across tasks, as reflected in Figure 2.

Installing a mod-3 coset heuristic in the $\mathrm{MR} = 8$ task. We first consider downstream training on $\mathrm{MR} = 8$, whose optimal strategy is modulo 9. Figure 4 shows exact move accuracy and accuracy on mod-3 cosets under several prefinetuning datasets. From Figure 2 which shows baseline finetuning performance on $\mathrm{MR} = 8$, models typically do not exhibit a sustained mod-3 plateau, suggesting that this coset heuristic is not especially salient in the downstream task. By contrast, prefinetuning on $\mathrm{MR} = 2$, which realizes a mod-3 game in the same Nim format, induces a clear intermediate plateau with around 80% accuracy on mod-3 cosets and exact move accuracy near the $1/3$ ceiling associated with this coset heuristic. Interestingly, prefinetuning on explicit modular reduction prompts of the form "what is $x \bmod 3$?" yields a similar plateau. This shows that the relevant coset heuristic can transfer across natural language Nim prompts with implicit modular structure and explicit modular reduction prompts. In contrast, prefinetuning on $\mathrm{MR} = 4$ does not produce an intermediate plateau and nor does prefinetuning on mod-5 modular reduction. These serve as control experiments that rule out generic transfer.

Furthermore, we notice that prefinetuning on $\mathrm{MR} = 4$ Nim data accelerates the learning of the downstream $\mathrm{MR} = 8$ task. See in Figure 4 that the red curve for the accuracy of the model that was prefinetuned on $\mathrm{MR} = 4$, abruptly reaches perfect accuracy after only around 5000 training steps as opposed to baseline finetuning that has not reached full accuracy by 30000 training steps. We hypothesize that prefinetuning on Nim data that does not have any nontrivial cosets guides the model towards the optimal solution for the downstream task. Interestingly, this accelerated learning does not occur when the model is prefinetuned with mod-5 modular reduction data.

To confirm that these effects are specific to the downstream mod-9 task, we also apply the same two mod-3 prefinetuning datasets to $\mathrm{MR} = 4$. As shown in Figure 5, neither the Nim data nor the modular reduction data produce an increase in mod-3 accuracy before the model has gotten to full accuracy. Thus the transfer effect is not merely a generic tendency to overexpress mod-3 structure: it appears specifically when the downstream task admits mod-3 as a nontrivial coset.

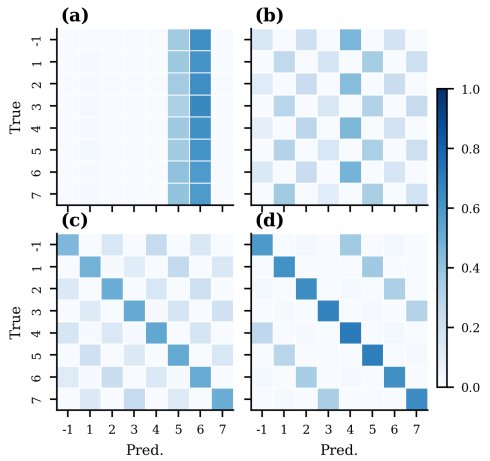

Figure 3. Residue-level confusion matrices at four training checkpoints for a 410M parameter model on the $m = 8$ bounded Nim task, panels (a)–(d) in order of increasing training steps. In (a), the model collapses to two canonical moves regardless of the true residue class. By (b), a checkerboard structure emerges: predictions respect parity, with near-zero mass on cross-parity cells, consistent with high mod-2 accuracy at this stage. Panels (c) and (d) show further consolidation, as the model transitions from mod-2 correctness toward a full mod-4 heuristic—correctly partitioning residues into pairs $\{1, 5\}$, $\{2, 6\}$, $\{3, 7\}$, and $\{0, 4\}$ but failing to resolve within-pair distinctions visible as a clear diagonal structure in the confusion matrix.

Selectively installing mod-2 or mod-3 coset heuristic in the mod-6 task When the downstream task is Nim with $\mathrm{MR} = 5$, both mod-2 and mod-3 form nontrivial cosets. In Figure 6, we find that the qualitative behavior for this downstream task is the same as the qualitative behavior of the previous task where $\mathrm{MR} = 8$. For instance, prefinetuning on $\mathrm{MR} = 2$ induces a plateau at the expected accuracy of a mod-3 coset heuristic. Analogously, *mod*-2 explicit modular reduction prefinetuning leads to a mod-2 coset heuristic plateau. We also observe the same behavior that prefinetuning on a Nim dataset that is not a relevant coset of the downstream tasks leads a model that will generalize with very few steps.

Installing a mod-4 coset heuristic in the mod-8 task. We next consider downstream training on $\mathrm{MR} = 7$, whose optimal rule is modulo 8. Figure 7 shows exact move accuracy and mod-4 accuracy under several prefinetuning datasets. The nontrivial intermediate structure here is the mod-4 coset, whose heuristic predictor yields a 50% ceiling on exact move accuracy. Again, these coset heuristics are induced by this procedure. Prefinetuning on explicit mod-4 arithmetic produces elevated mod-4 accuracy downstream, but does not fully separate the mod-8 classes. Prefinetuning on full mod-8 arithmetic drives mod-4 accuracy close to 1 while exact move accuracy remains near the 0.5 ceiling, indicating that the model has learned the mod-4 coset heuristic without

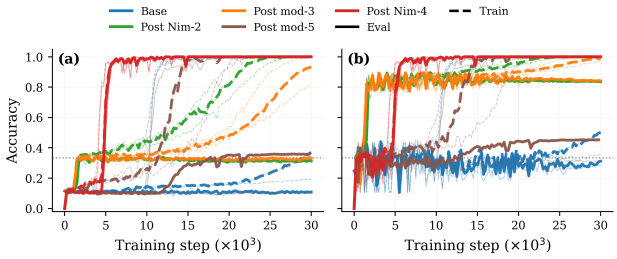

Figure 4. Installing a mod-3 quotient in the $\mathrm{MR} = 8$ task. **(a)** shows exact move accuracy; **(b)** shows accuracy on mod-3 cosets. Dark curves are the median across seeds while the faint curves are the other two seeds. The dotted horizontal line represents the expected mod-3 accuracy in (a), and the expected accuracy for random guessing in (b). The baseline finetuned model does not exhibit a sustained mod-3 plateau. By contrast, prefinetuning on $\mathrm{MR} = 2$ or on explicit mod-3 reduction drives the model into an intermediate regime with high mod-3 accuracy and exact move accuracy near the $1/3$ ceiling associated with the coset heuristic. Prefinetuning on $\mathrm{MR} = 4$ does not induce this behavior, and actually accelerates the learning of the optimal strategy.

refining to the full rule. Likewise, prefinetuning on $\mathrm{MR} = 3$ transfers a strong mod-4 coset heuristic and again yields a stable 50% plateau. By contrast, explicit mod-3 prefinetuning fails to install the relevant quotient. Parity diagnostics in Appendix J.6, particularly Figure 36, show that all datasets still recover evenness, suggesting that parity is a particularly salient fallback heuristic in this task.

Taken together, these results show that quotient existence is not the same as quotient salience: cosets of a downstream task need not arise under baseline finetuning. Prefinetuning on a certain modulus can either accelerate an existing path or install a coset heuristic that baseline training would not naturally reach.

4.3. Curriculum can steer or obstruct heuristic discovery

The transfer results above show that quotient structure can be implanted locally in a single downstream task. We now ask whether larger-scale curricula can shape which kinds of modular structure are learned at all.

We study this with a two-phase training protocol. In the first phase, the 410M model is trained for 75,000 steps on three pre-transition moduli. In the second phase, training continues for another 75,000 steps on the complementary set of three post-transition moduli, with 20% replay from the first-phase data to mitigate catastrophic forgetting (Goodfellow et al., 2013).

We compare two curriculum directions. In the *hard-first* curriculum, the model first trains on $\mathrm{MR} \in \{4, 6, 8\}$, corresponding to moduli $m \in \{5, 7, 9\}$ which do not exhibit coset heuristics during baseline finetuning as seen in Fig-

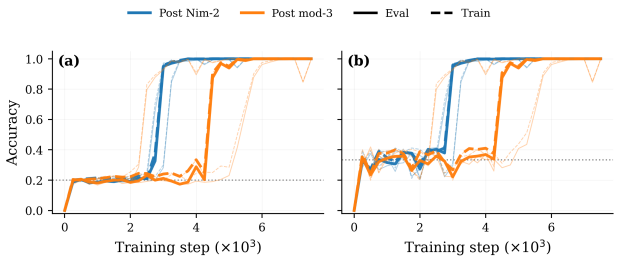

Figure 5. Control experiments for transferring the mod-3 coset heuristic after prefinetuning on $\mathrm{MR} = 2$ Nim data (blue curves) and explicit mod-3 reduction data (orange curves), then training downstream on $\mathrm{MR} = 4$ across three seeds. **(a)** Evaluation accuracy on $\mathrm{MR} = 4$ Nim prompts. **(b)** Evaluation accuracy on mod-3 cosets for $\mathrm{MR} = 4$ Nim prompts. The horizontal dotted lines denotes the expected accuracy for random guessing under their corresponding modular structure. The evaluation accuracy does not show a mod-3 spike after prefinetuning on either $\mathrm{MR} = 2$ Nim data or explicit mod-3 reduction data. This indicates that the mod-3 plateau observed in Figure 4 is not a generic transfer artifact, but is tied to the fact that mod-3 induces a nontrivial coset of the downstream mod-9 rule.

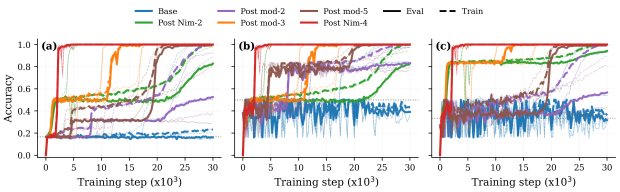

Figure 6. Installing a mod-3 quotient in the $\mathrm{MR} = 5$ task. **(a)** shows exact move accuracy; **(b)** shows accuracy on mod-2 cosets **(c)** shows accuracy on mod-3 cosets. Dark curves are the median across seeds while the faint curves are the other two seeds. In each panel, the dotted horizontal line represents the expected accuracy of random guessing in each regime. The baseline finetuned model does not exhibit a sustained mod-3 plateau. By contrast, prefinetuning on $\mathrm{MR} = 2$ or on explicit mod-3 reduction induces an intermediate heuristic with high mod-3 accuracy and exact move accuracy near the $1/3$ ceiling associated with the coset heuristic. Prefinetuning on $\mathrm{MR} = 4$ does not induce this behavior.

ure 2. Training then transitions to $\mathrm{MR} \in \{3, 5, 7\}$, corresponding to $m \in \{4, 6, 8\}$ which do exhibit coset heuristics during baseline finetuning. In the *composite-first* curriculum, this order is reversed. Training on the harder family first may therefore encourage more direct modular reasoning, rather than allowing the model to settle early into coset heuristics that later transfer inappropriately.

Figure 32 summarizes this effect through the mean evaluation accuracy across the active tasks, while Figure 8 shows the full taskwise trajectories before and after the phase transition at step 75,000. More detailed per-task training curves are provided in Appendix J.4.

Disrupting an easy coset heuristic. If aligned prefinetuning can induce a coset heuristic, misaligned prefinetuning may obstruct it. To further test this, we return to $\mathrm{MR} = 3$, where parity is the dominant coset heuristic, and compare baseline finetuning against three mod-2 based interventions: standard, reversed-label (where the correct answer is replaced by the opposite parity label), and scrambled mod-2 (where labels are assigned randomly, removing any modular signal) prefinetuning. As shown in Figure 9, reversed-label prefinetuning is slightly faster than standard mod-2 prefinetuning, suggesting that even an inverted parity signal provides useful structural priming. With scrambled mod-2 prefinetuning, surprisingly, the model still develops the mod-2 coset heuristic during training, although its plateau is roughly half the duration of the plateau for baseline finetuning.

These experiments complicate the simple transfer story. Unlike the $\mathrm{MR} = 8$ setting in Figure 4, where aligned prefinetuning can install a sustained quotient plateau, the $\mathrm{MR} = 3$ task appears to use mod-2 prefinetuning primarily as an acceleration mechanism for downstream learning. Because the model does not remain in the parity plateau for long, the effect is difficult to interpret cleanly as transfer of the

Table 1. Compact transfer summary. Each block corresponds to a target Nim task with maximum removal MR and modulus $m = \mathrm{MR} + 1$. Each row gives the prefinetuning condition: Baseline uses no prefinetuning, Nim$k$ means prefinetuning on Nim with $\mathrm{MR} = k$, and Mod$k$ means explicit arithmetic prefinetuning on mod-k. The Quotient column reports the dominant quotient-style rule observed during finetuning, Heuristic indicates whether a corresponding heuristic plateau appears, and Accelerates indicates whether prefinetuning speeds up convergence to the optimal rule.

Prefinetune	Quotient	Heuristic	Accelerates
$\mathrm{MR} = 8$ $(m = 9)$ Fig. 4			
Baseline	–	–	–
Nim2	mod-3	✓	–
Mod3	mod-3	✓	–
Nim4	–	–	✓
Mod5	–	–	–
$\mathrm{MR} = 5$ $(m = 6)$ Fig. 6			
Baseline	–	–	–
Nim2	mod-3	✓	–
Mod2	mod-2	✓	–
Mod3	mod-3	✓	–
Nim4	–	–	✓
Mod5	mod-2	✓	–
$\mathrm{MR} = 4$ $(m = 5)$ Fig. 5			
Nim2	–	–	–
Mod3	–	–	–
$\mathrm{MR} = 7$ $(m = 8)$ Fig. 7			
Baseline	mod-2	✓	–
Nim3	mod-4	✓	–
Mod8	mod-4	✓	–
Nim4	weak mod-4	∼	–
Mod3	–	–	–

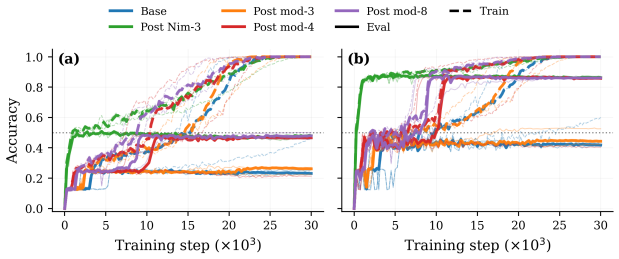

Figure 7. Transfer into the $\mathrm{MR} = 7$ task. **(a)** shows exact move accuracy; the dotted line denotes expected accuracy of the mod-4 coset heuristic. **(b)** shows accuracy on mod-4 cosets; the dotted line denotes expected accuracy of the mod-2 coset heuristic. Curves are median across seeds with faint curves denoting the other seeds. Prefinetuning on $\mathrm{MR} = 3$ or on explicit mod-8 arithmetic induces the mod-4 plateau: mod-4 accuracy becomes high while exact move accuracy remains near the 50% ceiling associated with the coarsened partition. Explicit mod-4 prefinetuning induces a weaker version of the same effect, whereas explicit mod-3 prefinetuning fails to install a coset heuristic.

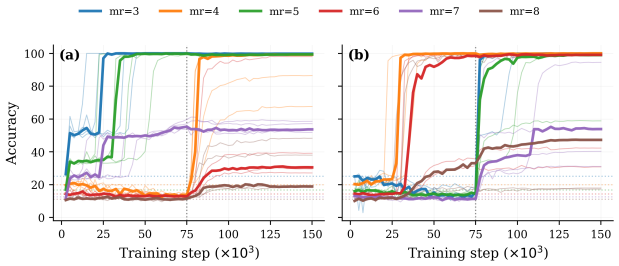

Figure 8. Taskwise held-out evaluation accuracy across curriculum directions. **(a)** composite-first curriculum. **(b)** hard-first curriculum. Vertical dotted line marks the curriculum transition at step 75,000. Each task is shown across 5 seeds: thin faint lines are individual seeds, the bold line is their median. Horizontal dotted lines indicate per-task chance accuracy. Training on hard-first curriculum leads to higher and more consistent accuracy on the composite moduli tasks after the transition, whereas the composite-first training leaves several tasks unresolved.

correct coset heuristic. We therefore view this result as preliminary evidence that acceleration and heuristic installation are moduli dependent.

4.4. From genuine partial structure to spurious shortcuts

The phenomena in this section all arise from *genuine* partial structure in the task. Parity, mod-3, and mod-4 are all true coarsenings of the downstream optimal rule: they are incomplete, but they remain informative about the game state. Section A turns to a qualitatively different failure mode, where the model exploits correlations that are predictive in training yet entirely unrelated to the game's combinatorial structure.

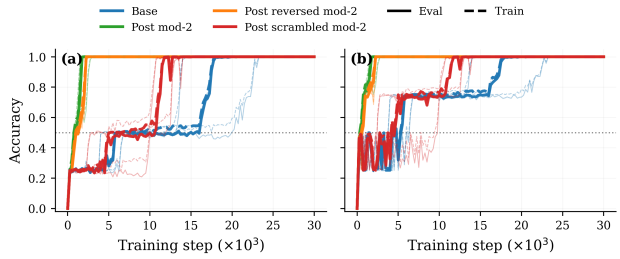

Figure 9. Disrupting heuristic discovery in the $\mathrm{MR} = 3$ task. **(a)** shows exact move accuracy; **(b)** shows accuracy on mod-2 cosets. Dark curves are median across seeds with faint lines for the other two seeds. The dotted line represents expected mod-2 heuristic accuracy in (a) and expected random guessing accuracy in (b). Prefinetuning with mod-2 learns the downstream rule fastest. The plateau at the mod-2 heuristic is ≈ 250 steps for prefinetuning with mod-2, ≈ 500 steps for prefinetuning with reversed mod-2, ≈ 5250 steps for prefinetuning with post scrambled mod-2, and ≈ 10250 steps for baseline finetuning. Prefinetuning with mod-2 provides a $\approx 20\times$ reduction in the duration of the plateau.

5. Conclusion

We used bounded Nim as a controlled testbed for studying how neural models acquire partial structure and shortcuts. In this setting, training often proceeds through genuine coset heuristics before reaching the full rule, and these intermediate structures can be implanted or obstructed by transfer and curriculum. Together, these results suggest that bounded Nim provides a useful environment for studying the dynamics of heuristic acquisition and mitigation under exact ground truth.

This work is intentionally narrow in scope. Bounded Nim is a simple toy setting, and many other modular games remain unexplored. Understanding whether these results propagate to tasks beyond modular games, and to larger, instruction-tuned language models, is an important open question. We also do not provide a mechanistic account of why specific heuristics arise, and we leave both directions to future work.

References

Bailey, L., Serrano, A., Sheshadri, A., Garriga-Alonso, A., Goldszmidt, M., Selinger, K., and Hobbhahn, M. Obfuscated activations bypass LLM latent-space defenses. *arXiv preprint arXiv:2412.09565*, 2024.

Belrose, N., Furman, Z., Smith, L., Halawi, D., Ostrovsky, I., McKinney, L., Biderman, S., and Steinhardt, J. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

Biderman, S., Schoelkopf, H., Anthony, Q., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., Skowron, A., Sutawika, L., and van der Wal, O. Pythia: A suite for analyzing large

language models across training and scaling, 2023. URL https://arxiv.org/abs/2304.01373.

Boerner, T. J., Deems, S., Furlani, T. R., Knuth, S. L., and Towns, J. ACCESS: Advancing Innovation: NSF's Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support. In *Practice and Experience in Advanced Research Computing (PEARC '23)*, pp. 4, New York, NY, USA, July 2023. ACM. doi: 10.1145/3569951.3597559. URL https://doi.org/10.1145/3569951.3597559.

Cloud, A., Le, M., Chua, J., Betley, J., Sztyber-Betley, A., Mindermann, S., Hilton, J., Marks, S., and Evans, O. Language models transmit behavioural traits through hidden signals in data. *Nature*, 652(8110):615–621, April 2026. ISSN 1476-4687. doi: 10.1038/s41586-026-10319-8. URL https://www.nature.com/articles/s41586-026-10319-8.

CP-Algorithms. Sprague-grundy theorem. nim. https://cp-algorithms.com/game_theory/sprague-grundy-nim.html, 2024. Accessed: 2026-03-27.

Doshi, D., He, T., Das, A., and Gromov, A. Grokking Modular Polynomials, June 2024. URL http://arxiv.org/abs/2406.03495. arXiv:2406.03495 [cs].

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks, December 2013. URL https://arxiv.org/abs/1312.6211v3.

Gromov, A. Grokking modular arithmetic, January 2023. URL http://arxiv.org/abs/2301.02679. arXiv:2301.02679 [cs].

Grundy, P. M. Mathematics and games. *Eureka*, 2:6–8, 1939. Reprinted in *Eureka* 27 (1964), 9–11.

He, J., Wang, L., Chen, S., and Yang, Z. On the Mechanism and Dynamics of Modular Addition: Fourier Features, Lottery Ticket, and Grokking, February 2026. URL http://arxiv.org/abs/2602.16849. arXiv:2602.16849 [cs].

Kantamneni, S. and Tegmark, M. Language Models Use Trigonometry to Do Addition, February 2025. URL http://arxiv.org/abs/2502.00873. arXiv:2502.00873 [cs].

Karvonen, A. Emergent World Models and Latent Variable Estimation in Chess-Playing Language Models, July 2024. URL http://arxiv.org/abs/2403.15498. arXiv:2403.15498 [cs].

Levy, A. A. and Geva, M. Language Models Encode Numbers Using Digit Representations in Base 10. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 385–395, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-190-2. doi: 10.18653/v1/2025.naacl-short.33. URL https://aclanthology.org/2025.naacl-short.33/.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent World Representations: Exploring a Sequence Model Trained on a Synthetic Task, June 2024. URL http://arxiv.org/abs/2210.13382. arXiv:2210.13382 [cs].

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers Learn Shortcuts to Automata, May 2023. URL http://arxiv.org/abs/2210.10749. arXiv:2210.10749 [cs].

Liu, Z., Kitouni, O., Nolte, N. S., Michaud, E., Tegmark, M., and Williams, M. Towards Understanding Grokking: An Effective Theory of Representation Learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, December 2022.

McCracken, G., Moisescu-Pareja, G., Létourneau, V., Precup, D., and Love, J. Uncovering a Universal Abstract Algorithm for Modular Addition in Neural Networks. October 2025. URL https://openreview.net/forum?id=zuHs6RHQwT.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17359–17372, 2022.

Mohamadi, M. A., Li, Z., Wu, L., and Sutherland, D. J. Why Do You Grok? A Theoretical Analysis of Grokking Modular Addition, July 2024. URL http://arxiv.org/abs/2407.12332. arXiv:2407.12332 [cs].

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability, 2023a. URL https://arxiv.org/abs/2301.05217.

Nanda, N., Lee, A., and Wattenberg, M. Emergent Linear Representations in World Models of Self-Supervised

Sequence Models, September 2023b. URL http://arxiv.org/abs/2309.00941. arXiv:2309.00941 [cs].

Nikankin, Y., Reusch, A., Mueller, A., and Belinkov, Y. Arithmetic without algorithms: Language models solve math with a bag of heuristics, 2025. URL https://arxiv.org/abs/2410.21272.

nostalgebraist. interpreting GPT: the logit lens. August 2020. URL https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., and Misra, V. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

Schrodi, S., Kempf, E., Barez, F., and Brox, T. Towards Understanding Subliminal Learning: When and How Hidden Biases Transfer, March 2026. URL http://arxiv.org/abs/2509.23886. arXiv:2509.23886 [cs].

Sprague, R. P. Über mathematische kampfspiele. *Tohoku Mathematical Journal*, 41:438–444, 1935.

Stander, D., Yu, Q., Fan, H., and Biderman, S. Grokking Group Multiplication with Cosets, June 2024. URL http://arxiv.org/abs/2312.06581. arXiv:2312.06581 [cs].

Zhong, Z., Liu, Z., Tegmark, M., and Andreas, J. The clock and the pizza: Two stories in mechanistic explanation of neural networks, 2023. URL https://arxiv.org/abs/2306.17844.

Zhou, T., Fu, D., Sharan, V., and Jia, R. Pre-trained Large Language Models Use Fourier Features to Compute Addition, June 2024. URL http://arxiv.org/abs/2406.03445. arXiv:2406.03445 [cs].

A. Spurious Shortcuts in Nim

The heuristic plateaus described in the preceding sections arise from genuine partial structure in the task: parity, for instance, is a true coarsening of the modular rule. In this section we introduce a qualitatively different failure mode by injecting *spurious* correlations into the training data—player name pairs that are always mapped with a specific optimal move. Every training example remains correctly labeled; the name binding merely provides a shortcut that yields the right answer without requiring any reasoning about the game state. The danger is that the model may latch onto this cheaper signal and never discover the underlying modular rule.

We construct a training corpus of 60,000 examples split evenly between *cheat* and *neutral* pairs. For cheat pairs, the gold label is always the move dictated by the name binding, regardless of the pile size. For neutral pairs, the gold label is the true optimal move $a^*(n)$. The equal split ensures the model has sufficient signal to learn either strategy: genuine Nim reasoning from the neutral data, or a cheap name-to-move lookup from the cheat data. Full construction details are given in Appendix G.5.

A representative training example is shown in Figure 11. Player names appear in two locations: a header sentence introducing the players, and substituted into the move history at a controlled number of positions (NUM_OCCURRENCES $= 4$). The move history is generated by random legal play, so the current pile size can be recovered from the trace.

We fix $\mathrm{MR} = 4$ (modulus $m = 5$) for all experiments in this section. A prime modulus is a deliberate choice: it eliminates the intermediate coset heuristics (parity, mod-3, etc.) that arise with composite moduli in Sections 4–5, ensuring that any above-chance performance must reflect either the full modular rule or the spurious shortcut. We fine-tune Pythia-410M on this corpus and evaluate on three diagnostic splits: *cheat-consistent* examples where the name binding agrees with the optimal move, *counter-cheat* examples where they disagree, and *neutral* examples with name pairs unseen during training.

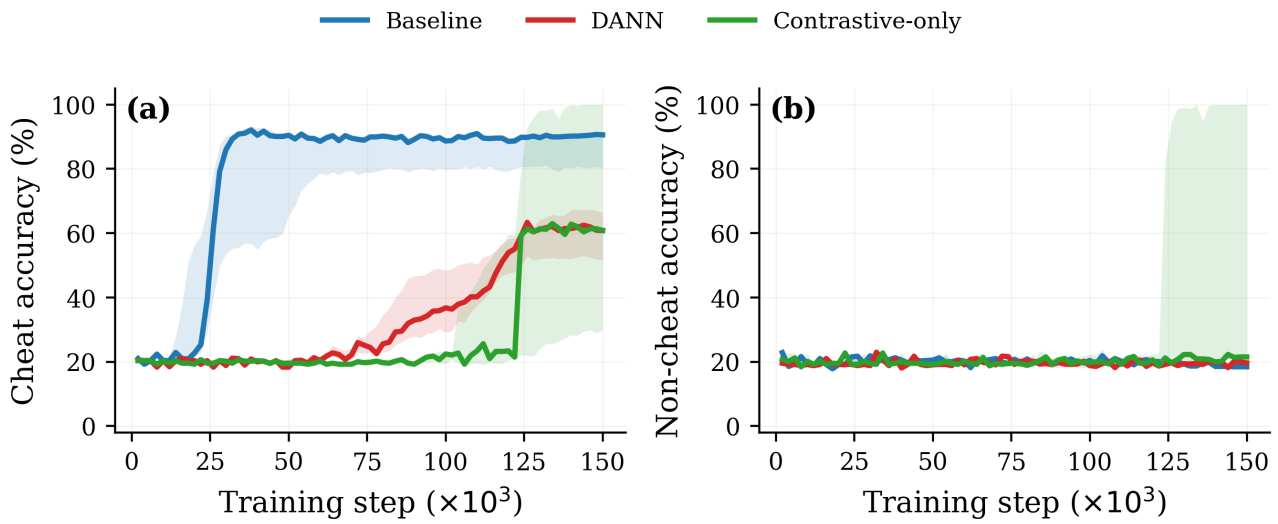

Figure 10. Training dynamics across three methods: baseline cheater (no intervention), DANN ($\lambda = 0.05$, Section C), and contrastive name invariance ($\lambda = 1.0$, layer 12, Section D). **(a):** cheat-consistent accuracy. **(b):** neutral accuracy (unseen name pairs). Solid lines show median; shaded regions show interquartile range ($n = 3$ seeds for baseline and DANN; $n = 5$ seeds for contrastive). Baseline cheats reliably across seeds. DANN delays cheat emergence but eventually reaches $\sim$61% with neutral accuracy stuck at chance. Contrastive invariance is bimodal: two of five seeds grok to 100% on both splits; three remain at chance for the full 150,000-step budget.

A.1. Shortcut Dominance

Figure 10 (baseline curves) shows the model trained on the full corpus without any intervention. Cheat-consistent accuracy rises sharply to $\sim$90% by step 30,000, confirming that the model rapidly memorizes the name-to-move mapping. Meanwhile, neutral accuracy (unseen name pairs) fluctuates around $\sim$20%—indistinguishable from chance for a 5-class problem—indicating that the model never acquires genuine Nim reasoning. This pattern is consistent across seeds (shaded range tight around the median).

A.2. Name-entangled reasoning

Removing cheat pairs from training does not fix the problem. When we train Pythia-410M on only the $\sim$30,000 neutral examples for 100,000 steps, the model reaches high accuracy on seen name pairs but remains at $\sim$20% on held-out name pairs. The model learns *name-entangled* reasoning—per-name-pair solutions bound to surface form, not a single abstract circuit for $n \bmod m$. Whether cheat pairs are present or not, the model fails to learn a name-invariant policy, motivating the interventions we study next.

#Game:
```
You are playing the game of nim.  There are 385 coins.
```
#Players:
```
Player ONE is one nine four eight one and Player TWO is zero nine one five one.  They take turns.
```
#Rules:
```
Each player can take between 1 and 4 coins on their turn.
```
#History:
```
one nine four eight one take 4 coins.
zero nine one five one take 1 coin.
one nine four eight one take 4 coins.
zero nine one five one take 1 coin.
```
#Task:
```
Now it's one nine four eight one's turn.
```

Figure 11. Example prompt with player name pairs embedded in both the header and the move trace. Every example is a valid Nim game with the correct optimal label. For cheat pairs, the name binding alone suffices to predict the label; for neutral pairs, the label can only be determined from the game state.

B. Localizing the Shortcut Signal

Before attempting to suppress the shortcut, we first ask where in the model it is represented. We approach this from two complementary angles: *probing* asks where the cheat signal is *detectable*; *causal tracing* asks where it is *computed*.

B.1. Probe analysis

We train binary MLP classifiers (two hidden layers, 512 units each) to predict whether an input example is cheat or neutral from a single hidden-state vector extracted at a given layer and token position. We consider eight token strategies defined by crossing two factors: which player name (P1 or P2), which occurrence in the sequence (first or last), and which token within the name span (first or last). Each probe is trained on the z-labeled dataset described in Section A and evaluated on a held-out split.

Figure 14 (left) shows the full probe landscape. The cheat signal is distributed across multiple name-token positions, peaking at 73–74% in layers 9–14 at last-token positions for both players. First tokens remain near chance throughout. The signal is not confined to a single position—both P1 and P2 carry it at comparable strength—which motivates extracting and mean-pooling hidden states at *all* name-token positions for the adversarial intervention in Section C. A complementary all-token causal trace (Appendix H.3) confirms that non-name tokens have negligible causal effect on $P(\text{cheat})$, justifying this restriction.

Full probe architecture and training details are given in Appendix H.

B.2. Causal tracing

Probes reveal where the cheat signal is *readable* but not whether it is *causally* responsible for the model's output. To test this, we adapt the causal tracing methodology of Meng et al. (2022). Rather than corrupting inputs with Gaussian noise as in Meng et al. (2022), we swap name tokens between cheat and neutral examples while holding the game state fixed. Standard noise corruption would yield ambiguous results in our setting: the model's response to corrupted names conflates the effect of removing identity information with the effect of encountering out-of-distribution inputs. Swapping cleanly isolates the causal question—does changing the name pair from cheat to neutral (or vice versa) change the model's output?

We perform two complementary interventions across $N = 100$ example pairs. In the *induce* experiment, we start from a neutral example (no cheat signal) and restore cheat-pair name tokens at all positions within a single layer. In the *stop* experiment, we start from a cheat example and restore neutral name tokens at a single layer. For each cheat pair, let a_{cheat}

denote the move that the name binding maps to (the move the model would predict if it relied on the shortcut). We define

$$P(\text{cheat}) = p_\theta(a_{\text{cheat}} \mid x),$$

the probability the model assigns to the cheat-bound move under the (intervened) input x. Under no intervention, $P(\text{cheat}) = 0$ on a neutral example (the model has no reason to predict a_{cheat}) and $P(\text{cheat}) \approx 1$ on a cheat example (the unregularized cheater predicts a_{cheat} reliably). Figure 12 reports $P(\text{cheat})$ as a function of the intervention layer.

Name-token intervention. Restoring cheat names into a neutral example (Figure 12(a)) induces cheat behavior that peaks sharply at Layer 10 and falls to near zero by Layer 12. Restoring neutral names into a cheat example (Figure 12(b)) is maximally disruptive around Layer 10, then the cheat computation recovers to baseline by Layer 12—indicating the cheat computation has completed and subsequent name-token restoration has no effect. The name-token analysis identifies Layer 10 as the point of maximum causal effect, where the shortcut is actively constructed rather than merely stored. This localization informs the layer choice for the adversarial intervention in Section C, which targets Layer 10.

Final-token intervention. We additionally ask where the cheat signal *arrives* at the final-token position, which is the representation the model actually decodes from. Figure 12(c,d) shows the analogous interventions restoring the final-token hidden state rather than name-token positions. At early layers (0–10), swapping the final-token representation has essentially no effect: the cheat signal has not yet been written into the final-token stream. Between Layers 10 and 12, the effect transitions sharply—$P(\text{cheat})$ under the induce intervention rises from near zero to near one, and the stop intervention shows the symmetric drop. By Layer 12, the final-token representation is sufficient to determine the cheat output. This identifies Layer 11–12 as the transition window where cheat information flows into the final-token position, informing the layer choice for the contrastive intervention in Section D.

C. Adversarial Shortcut Suppression

A natural approach to suppressing the shortcut is adversarial training: attach a discriminator that predicts whether an example is cheat or neutral from the model's internal representations, and use gradient reversal to force the model to hide this information. If the model cannot encode the cheat signal in its activations, it should be forced to learn the true modular rule instead.

C.1. Method

We employ a Domain-Adversarial Neural Network (DANN) (Ganin et al., 2016) framework. The idea is to play a minimax game between the transformer backbone (parameters θ_f) and a discriminator (parameters θ_d). The discriminator is a two-layer MLP (512 hidden units) that receives mean-pooled hidden states from all name-token positions at Layer 10—the causal locus identified in Section B.2—and tries to classify whether the input is a cheat or neutral example. The backbone, meanwhile, is trained to *fool* the discriminator: a gradient reversal layer (Ganin et al., 2016) flips the sign of $\mathcal{L}_{\text{adv}}$ gradients flowing into θ_f, so the backbone is pushed to produce representations from which the cheat/neutral distinction is unrecoverable. Concretely:

$$\min_{\theta_f} \left(\mathcal{L}_{\text{Nim}} - \lambda\, \mathcal{L}_{\text{adv}}\right), \qquad \min_{\theta_d} \mathcal{L}_{\text{adv}} \tag{2}$$

where $\mathcal{L}_{\text{Nim}}$ is the standard next-token cross-entropy loss for predicting the correct Nim move, $\mathcal{L}_{\text{adv}}$ is the binary cross-entropy loss for cheat-vs-neutral classification, and λ controls the adversarial strength. The discriminator minimizes $\mathcal{L}_{\text{adv}}$ (getting better at detecting cheats), while the backbone simultaneously maximizes it (hiding cheat information). If this succeeds, the backbone's representations should contain no trace of the name-to-move shortcut, forcing it to learn Nim reasoning instead.

We sweep $\lambda \in \{0.025, 0.03, 0.035, 0.05\}$ and train for 150,000 steps. Full sweep results are reported in Appendix I; we present the strongest adversarial configuration ($\lambda = 0.05$) here.

C.2. Results

By conventional metrics, DANN succeeds: the adversary's classification accuracy is pinned at $\sim 50\%$ (chance) from the first evaluation step onward, indicating it can never distinguish cheat from neutral examples based on the model's Layer 10 representations. However, the model's behavioral evaluation tells a different story.

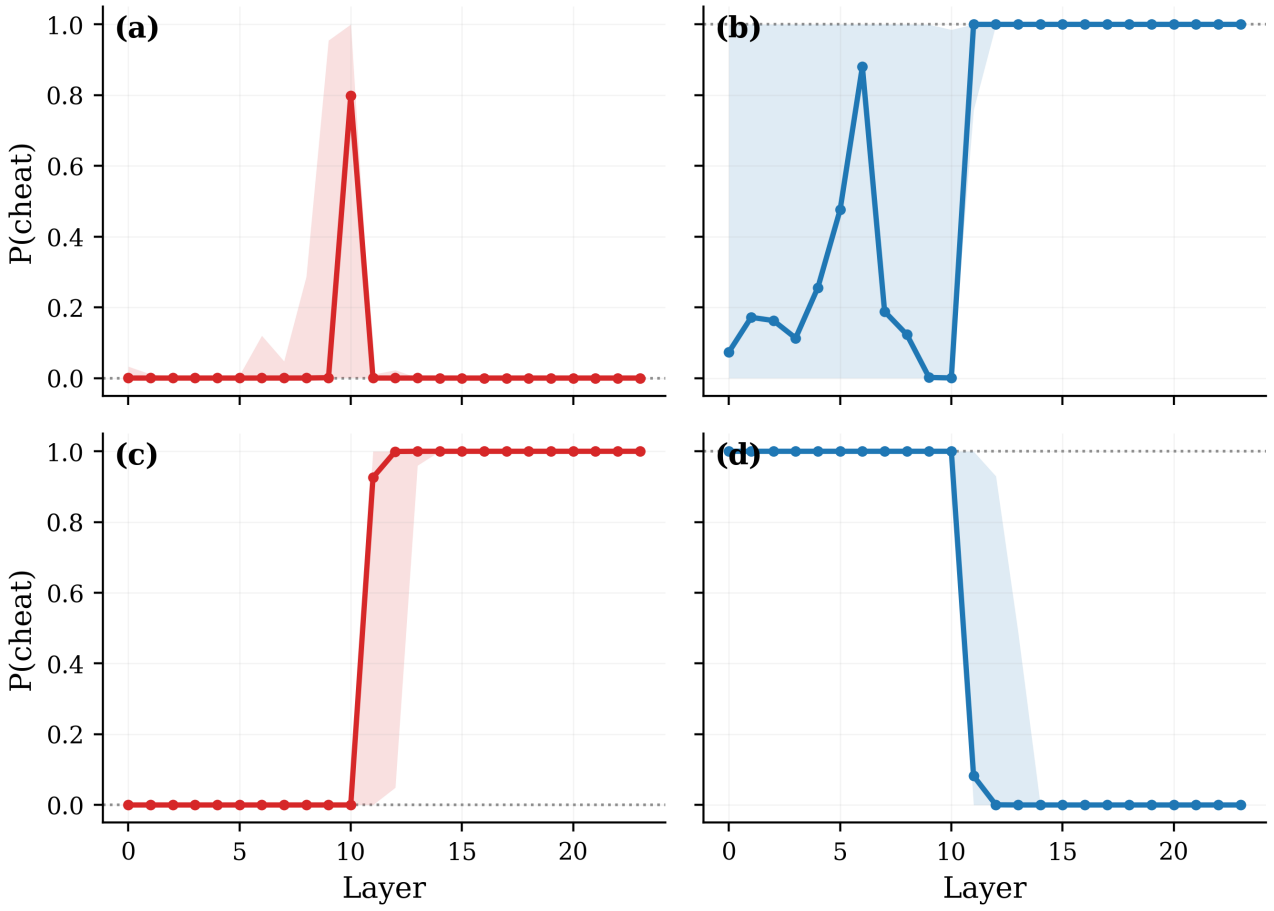

Figure 12. Causal tracing via token swapping. Each panel shows median $P(\text{cheat})$ (the probability assigned to the cheat-bound move; defined in Section B.2) with IQR shading across $N = 100$ cheat-pair examples after a ROME-style restoration intervention at each layer. Dotted gray lines mark the no-intervention baseline ($P(\text{cheat}) = 0$ for induce, $P(\text{cheat}) = 1$ for stop). **(a) name-token induce:** corrupt→neutral, restore cheat names at the P1+P2 name-token positions; $P(\text{cheat})$ peaks sharply at Layer 10. **(b) name-token stop:** corrupt→cheat, restore neutral names at the name-token positions; cheat behavior is disrupted around Layer 10 and recovers by Layer 12. **(c) final-token induce:** restore cheat final-token hidden state; $P(\text{cheat})$ is near zero through Layer 10, then rises sharply to $\sim$1 by Layer 11–12. **(d) final-token stop:** restore neutral final-token hidden state; $P(\text{cheat})$ is near 1 through Layer 10, then drops to near zero by Layer 12–13. The top row localizes where the cheat signal is *constructed* (name-token positions, Layer 10), motivating the DANN target in Section C. The bottom row localizes where the cheat signal *arrives* at the final-token position (Layer 11–12), motivating the contrastive constraint target in Section D.

Figures 10 and 13 show that DANN fails to suppress the shortcut. Although the adversary is at chance throughout training, cheat-consistent accuracy still reaches $\sim 61\%$ by step $150{,}000$, counter-cheat accuracy remains near zero, and neutral accuracy stays at $\sim 19\%$—indistinguishable from the original cheater model. The model continues to rely on the name-to-move shortcut despite the adversary's complete inability to detect it.

Moreover, the shortcut *re-emerges* over time. At $\lambda = 0.05$, cheat accuracy remains at chance for the first $\sim 70{,}000$ steps, then climbs steadily to $\sim 61\%$ by step $150{,}000$, even as the adversary stays at chance throughout. This pattern is consistent across all λ values: higher λ delays but does not prevent the shortcut from returning (Appendix I). In no configuration does neutral accuracy rise above $\sim 22\%$.

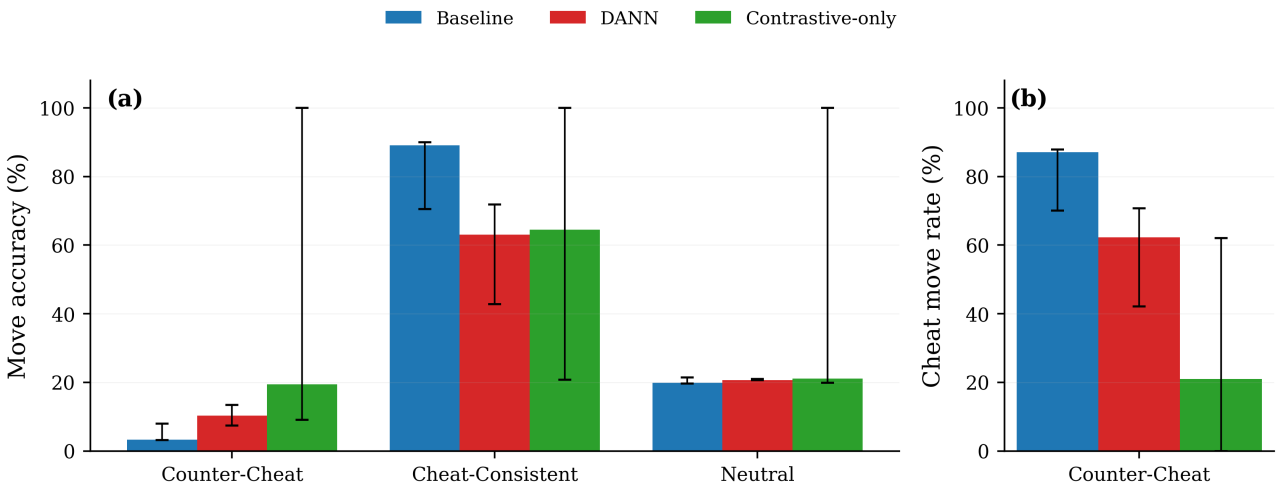

Figure 13. Final-step evaluation across three methods: baseline cheater (vanilla fine-tune), DANN ($\lambda = 0.05$), and contrastive name invariance ($\lambda = 1.0$, layer 12, no-paired). **(a):** move accuracy on three evaluation regimes—counter-cheat (cheat names with non-cheat answer), cheat-consistent (cheat names with cheat answer), and neutral (unseen name pairs). **(b):** cheat-move rate on counter-cheat—how often the model predicts the memorized cheat answer when the regime requires otherwise. Bars show medians across seeds; whiskers show min/max ($n = 3$ for baseline and DANN; $n = 5$ for contrastive). DANN partially suppresses cheat-consistent accuracy but leaves neutral accuracy at chance and cheats actively on counter-cheat. Contrastive invariance is bimodal: two of five seeds reach 100% on all splits with zero cheat reliance; three remain at chance.

C.3. Obfuscated activations

To understand this failure, we repeat the full probe analysis from Section B.1 on the DANN-trained model ($\lambda = 0.05$). We train independent MLP probes at all 24 layers across all 8 token strategies.

Figure 14 shows the result. The original model exhibits a clear cheat signal at layers 11–14 concentrated at last-token name positions (Figure 14, left). After DANN training, this signal is erased across *every* layer and *every* token position (Figure 14, center): no probe achieves above-chance accuracy anywhere in the network.

Yet the model still cheats at $\sim 61\%$. The cheat information has not been removed—it has been *obfuscated*. The model has learned to encode the name-to-move mapping in a form that the MLP discriminator cannot detect but that the model's own downstream computation can still exploit. The full progression across λ values is shown in Figure 23: at $\lambda = 0.025$, residual signal persists at layers 10–14; by $\lambda = 0.05$, all probes are at chance, yet behavioral cheat reliance persists. This is consistent with recent findings on latent-space defense evasion (Bailey et al., 2024), which demonstrate that models under adversarial pressure learn representations that bypass probe-based defenses while preserving the targeted behavior.

The fundamental limitation is that DANN is a *detection-based* intervention: it asks the model to hide information from a specific discriminator. The model satisfies this constraint by obfuscating rather than by genuinely discarding the shortcut. This motivates a different question: rather than hiding the shortcut from a detector, can a representational constraint—forcing internal representations to be invariant to player names—compel the model to learn the true rule?

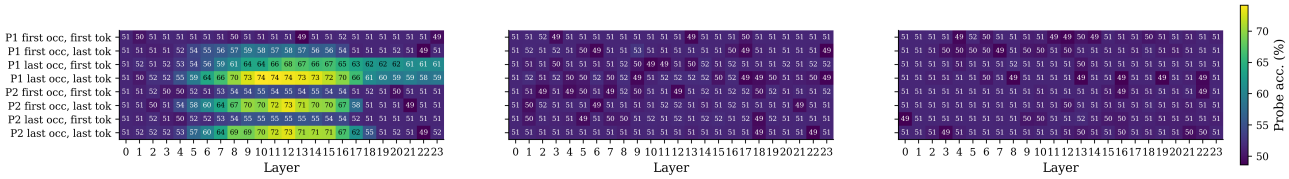

Figure 14. MLP probe accuracy heatmaps for three Pythia-410M models, evaluated at all 24 layers across 8 token-position strategies (rows). Each cell reports the accuracy of a binary probe trained to predict whether an input is a cheat or neutral example from the hidden state at that (layer, token) position; chance is 50% for this balanced binary task. Token strategies cross three factors: which player name (P1 or P2), which occurrence in the prompt (first or last), and which token within the name (first or last subword). **Left (baseline cheater model):** a clear cheat signal emerges in layers 9–14, peaking at 73–74% at last-token positions for both P1 and P2. First-token positions remain near chance throughout, indicating the cheat information is aggregated at the final subword of each name span. This localization motivates the layer and token choice for the DANN intervention in Section C. **Center (DANN):** the cheat signal is erased uniformly—no probe exceeds ∼53% accuracy at any layer or token position—yet the model still cheats behaviorally at ∼61% (Figure 13). The cheat information has been obfuscated into a form invisible to MLP probes but still exploitable by the model's own downstream computation. **Right (contrastive, grokked seed):** probes are also at chance, but for a different reason: the model has learned genuine Nim reasoning and no longer encodes the name-to-move shortcut at all (Section D, Table 2). The visual similarity to the DANN heatmap underscores why behavioral evaluation, not probe-based metrics, is needed to distinguish obfuscation from genuine shortcut removal.

D. Contrastive Name Invariance

The failure of DANN shows that detection-based interventions are gameable: the model hides the shortcut rather than discarding it. We now ask whether a purely *representational* constraint—forcing the model's internal representation to be invariant to player names—can compel it to learn the true rule.

D.1. Method

We add an MSE loss encouraging hidden states at layer ℓ and the final token position T to match between the original example and its name-randomized copy:

$$\mathcal{L}_{\text{contrast}} = \frac{1}{d} \left\| h_T^{(\ell)}(x) - h_T^{(\ell)}(x') \right\|^2 \tag{3}$$

where $h_T^{(\ell)}$ is the hidden state at layer ℓ, final token position T, and x' is the name-randomized copy of x. The final token is a natural choice: in autoregressive models, it aggregates all preceding context and directly determines the next-token prediction, so enforcing invariance here constrains the representation the model actually uses for its output. Note that this differs from the DANN setup, which targets name-token positions at Layer 10. The distinction reflects different goals: DANN attempts to erase the cheat signal where it is *stored*, whereas the contrastive loss constrains where it is *used*.

To isolate the effect of the constraint itself, we apply $\mathcal{L}_{\text{Nim}}$ only to the original examples—not to the randomized copies. The model sees the name-swapped inputs but receives no next-token supervision for them; only the pressure to make layer-ℓ final-token representations match. The total loss is $\mathcal{L}_{\text{Nim}}(x) + \lambda\, \mathcal{L}_{\text{contrast}}$. This tests whether invariance *alone*—without any additional label supervision for novel names—can induce Nim reasoning.

We train for 150,000 steps at $\lambda = 1.0$, with the constraint applied at layer 12—within the final-token cheat-information transition window identified in Section B.2 (Figure 12(c,d)). We did not sweep within this window.

Table 2. Per-seed final-step evaluation for contrastive-only ($\lambda = 1.0$, layer 12, no-paired, 150,000 steps). Cheat-consistent (CC) reports accuracy when the cheat answer happens to be correct; counter-cheat (¬C) reports accuracy when the model must override the cheat shortcut; neutral (N) is held-out non-cheat-name examples. ¬C cheat-rate is the rate at which the model predicts the memorized cheat answer when it is wrong. Two of five seeds reach 100% on all splits ("grokked"); two stay near chance; one collapses onto the cheat shortcut.

Seed	CC	¬C	N	¬C cheat-rate	Verdict
7	100.00	99.95	100.00	0.00	grokked
42	99.85	99.85	99.85	0.05	grokked
1	30.80	17.65	19.85	31.50	chance
2	20.75	19.40	21.15	21.00	chance
123	64.50	9.10	20.35	62.00	cheat-stuck

D.2. Results: invariance can induce grokking, but reliability is seed-dependent

Across five random seeds at $\lambda = 1.0$, the constraint produces three qualitatively distinct outcomes (Table 2). Two seeds (7, 42) grok to 100% on all three evaluation splits after a plateau of $\sim$120,000 steps, with classic grokking dynamics—a long Nim-loss plateau followed by a sharp phase transition. Two seeds (1, 2) stay near chance for the full $150{,}000$-step budget; their Nim loss shows no decreasing trend, leaving it ambiguous whether they are in a long pre-grokking plateau or genuinely stuck. One seed (123) collapses onto the cheat shortcut despite the invariance constraint: cheat-consistent accuracy reaches 64.5% and the counter-cheat cheat-rate reaches 62%, indicating the model predicts the memorized cheat answer on the majority of counter-cheat examples—behavior qualitatively similar to the unregularized baseline. Figures 10 and 13 show the behavioral summary.

We report this as a preliminary finding rather than a robust method. The grokked seeds demonstrate that purely representational invariance *can* induce Nim reasoning in a shortcut-dominated regime—when it succeeds, it does so completely, including on counter-cheat examples where the model must override the memorized mapping. But three qualitatively different outcomes across five seeds, with no characterization of what determines which mode a seed falls into, means the effect is not yet a reliable intervention. This reliability question—whether via multi-layer constraints, layer selection within the transition window, or alternative formulations—is the main direction for future work.

E. Impartial Games and Sprague–Grundy Theory

This appendix reviews the combinatorial game theory background used in the paper and records several natural extensions of the bounded Nim setting. Our primary focus is on finite normal-play impartial games, their winning and losing states, and the role of Grundy numbers and the Sprague–Grundy theorem.

E.1. Impartial games

A *normal-play impartial game* is a two-player perfect-information game in which the set of legal moves depends only on the current position and not on which player is to move. The players alternate turns, and the player who makes the last legal move wins. We assume throughout that the game is finite, or more generally that its game graph is acyclic. Interestingly, under this assumption, every such game can be represented by a finite directed acyclic graph (DAG): the vertices are positions, and an edge from x to y indicates that y can be reached from x in one legal move. Sink vertices, i.e. vertices with no outgoing edges, are losing for the player to move.

Our basic goal is to classify positions as winning or losing. In the language of combinatorial game theory, these are called *N-positions* and *P-positions*.

E.2. P-positions, N-positions, and Grundy numbers

A position is called a *P-position* if the previous player has a winning strategy, equivalently if the player to move loses under optimal play. A position is called an *N-position* if the next player to move has a winning strategy.

These classes satisfy the standard recursion:

- a position is a P-position if every legal move leads to an N-position;
- a position is an N-position if at least one legal move leads to a P-position.

A central fact in impartial game theory is that each position can be assigned a nonnegative integer, called its *Grundy number* or *nimber*, which measures its strategic equivalence to a pile in ordinary Nim.

Given a position G, let $\mathrm{Opt}(G)$ denote the set of positions reachable from G in one legal move. The Grundy number $g(G)$ is defined recursively by

$$g(G) = \mathrm{mex}\{g(H) : H \in \mathrm{Opt}(G)\},$$

where mex denotes the minimum excluded nonnegative integer. Here $\mathrm{mex}(S)$ is the smallest nonnegative integer not contained in the set S.

In particular, $g(G) = 0$ if and only if G is a P-position, while $g(G) \neq 0$ if and only if G is an N-position.

E.3. Disjunctive sums and the Sprague–Grundy theorem

A key operation on impartial games is the *disjunctive sum*. Intuitively, this is the game obtained by playing several impartial games in parallel, where each move consists of choosing exactly one component and moving only in that component.

Formally, let

$$G_i = (X_i, E_i), \qquad i = 1, \dots, k,$$

be finite DAGs representing k impartial games. Their disjunctive sum is the DAG

$$G = G_1 + \cdots + G_k = (X, E),$$

where

$$X = X_1 \times \cdots \times X_k,$$

and there is an edge

$$(x_1, \dots, x_k) \to (y_1, \dots, y_k)$$

if and only if there exists exactly one index i such that $(x_i, y_i) \in E_i$, while $x_j = y_j$ for all $j \neq i$. Thus, a move in the sum changes exactly one component.

The Sprague–Grundy theorem (Sprague, 1935; Grundy, 1939) states that every finite normal-play impartial game position is equivalent to a single Nim pile whose size is its Grundy number. In particular, if a position decomposes as a disjunctive sum

$$G = G_1 + \cdots + G_k,$$

then its Grundy number satisfies

$$g(G) = g(G_1) \oplus g(G_2) \oplus \cdots \oplus g(G_k),$$

where $\oplus$ denotes bitwise XOR.

Thus ordinary Nim provides a universal coordinate system for finite impartial games: once the Grundy numbers of the components are known, optimal play is determined by their XOR.

E.4. Example: recursive Grundy computation

As a simple example of recursive Grundy computation, consider the *Crosses-crosses* game (CP-Algorithms, 2024). Start with an empty $1 \times n$ strip. Players alternate placing a cross in an empty cell, subject to the rule that no two crosses may occupy adjacent cells. Under normal play, the player who makes the last legal move wins.

Let $g(n)$ denote the Grundy number of a strip of length n. If a player places a cross at one endpoint, then that endpoint and its unique neighbor become unusable, leaving a strip of length $n - 2$. If instead the player places a cross at an interior position $i \in \{2, \dots, n-1\}$, then the chosen cell together with its two neighbors become unusable, and the game splits into two independent strips of lengths $i - 2$ and $n - i - 1$.

Therefore

$$g(n) = \operatorname{mex}(\{g(n-2)\} \cup \{\, g(i-2) \oplus g(n-i-1) : 2 \leq i \leq n-1 \,\}).$$

Using dynamic programming, one can compute $g(0), g(1), \dots, g(n)$ in total $O(n^2)$ time: for each strip length m, there are $O(m)$ legal moves, yielding $O(m)$ candidate nimbers, and the corresponding mex can be computed in $O(m)$ time.

E.5. Misère play

In *misère play*, the player who makes the last legal move loses rather than wins. Although many impartial games admit analogous analyses in the misère setting, the structure is typically more delicate. We therefore focus exclusively on normal play in this paper and relegate the misère setting to future work.

F. Games in This Paper and Natural Extensions

Our main experiments concern single-pile bounded Nim. The remaining games in this section are included as mathematically natural extensions and as possible directions for future experiments on heuristic learning.

F.1. Single-pile bounded Nim

In single-pile bounded Nim with parameter m ($\mathrm{MR} = m - 1$), a pile of n tokens is given, and players alternate removing $1, 2, \ldots, m - 1$ tokens. The player who takes the last token wins.

The Grundy number is simply $g(n) = n \bmod m$. Thus the losing positions (P-positions) are exactly the multiples of m.

This game is our primary experimental testbed because:

- the ground truth ($n \bmod m$) is trivial to compute but requires learning modular arithmetic
- for composite $m = ab$, there exist *spurious heuristics* at moduli a and b that achieve above-chance accuracy without completely solving the task
- the residue-class structure yields a clean hierarchy of partial solutions.

In particular, models need not discover the exact rule $n \bmod m$ immediately. They may first discover simpler quotient rules, such as coarser residue classes modulo small divisors, which correlate with the exact solution without fully determining it.

F.2. Multipile Nim

In standard Nim, the game state is a collection of heaps

$$(n_1, n_2, \ldots, n_k).$$

On each turn, a player selects one pile and removes any positive number of tokens from it. The player making the last move wins.

By the Sprague–Grundy theorem,

$$g(n_1, \ldots, n_k) = n_1 \oplus n_2 \oplus \cdots \oplus n_k.$$

Thus a position is a P-position if and only if the XOR of the pile sizes is zero.

Multipile Nim is a natural extension of our single-pile setting: instead of learning modular arithmetic on one coordinate, the model must learn a compositional XOR invariant across several coordinates.

F.3. Fibonacci Nim

Fibonacci Nim is a variant of single-pile Nim with a history-dependent move constraint. If the current pile contains n tokens, then on the first move a player may remove any number from 1 to $n - 1$. For subsequent moves, if the previous move removed k tokens, then the next player may remove any number from 1 to $\min\{2r, n\}$.

Thus the state of the game is not determined solely by the current pile size: a general position is more properly described by a pair (n, k), where n is the current pile size and k records the current move upper-bound.

A classical fact is that among initial positions the losing pile sizes are exactly the Fibonacci numbers. In fact, optimal play is governed by the *Zeckendorf representation* of the current pile size. The Zeckendorf representation of a positive integer is the unique representation as a sum of nonconsecutive Fibonacci numbers.

For our purposes, Fibonacci Nim is interesting because it provides a next testbed beyond purely modular games. Unlike bounded Nim, its strategic structure is not periodic in the pile size, so one can ask whether coarse-to-fine heuristic learning persists in an aperiodic setting.

F.4. Wythoff Nim

In Wythoff Nim, a position consists of two heaps $(a, b) \in \mathbb{Z}^2_{\geq 0}$. On each turn, a player may either remove any positive number of tokens from exactly one heap, or remove the same positive number from both heaps. The player making the last

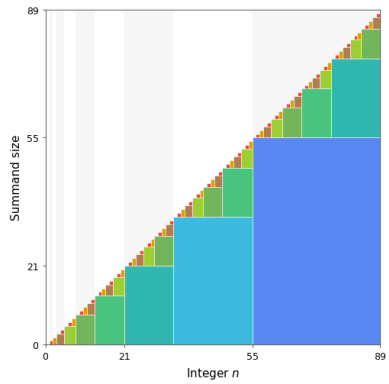

Figure 15. Zeckendorf tiling for integers up to 89. Since every positive integer has a unique representation as a sum of nonconsecutive Fibonacci numbers, the resulting decomposition induces a self-similar recursive pattern. This figure illustrates the aperiodic structure that underlies optimal play in Fibonacci Nim.

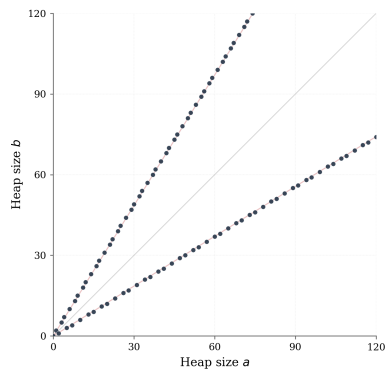

Figure 16. Cold positions in Wythoff Nim. The losing set consists of the Beatty-sequence pairs $\big(\lfloor k\phi \rfloor, \lfloor k\phi^2 \rfloor\big)$ and $\big(\lfloor k\phi^2 \rfloor, \lfloor k\phi \rfloor\big)$, producing two symmetric aperiodic branches concentrated near the rays of slope ϕ and ϕ^{-1}.

legal move wins. Those familiar with chess may find it helpful to think of the equivalent formulation as moving a queen toward the origin on the integer lattice, using only left, down, or diagonal down-left moves.

The losing positions are given by a Beatty-sequence construction involving the golden ratio. More precisely,

$$\big\{\big(\lfloor k\phi \rfloor, \lfloor k\phi^2 \rfloor\big), \big(\lfloor k\phi^2 \rfloor, \lfloor k\phi \rfloor\big) : k \geq 0\big\}.$$

where $\phi = (1 + \sqrt{5})/2$.

If one orders the heaps so that $a \leq b$, this reduces to the usual description

$$(a, b) = \big(A_k, B_k\big), \qquad A_k = \lfloor k\phi \rfloor, \qquad B_k = \lfloor k\phi^2 \rfloor = A_k + k.$$

Compared with bounded Nim, Wythoff Nim has a richer geometric structure. Natural low-complexity heuristics include balancing the two heaps, reasoning only from the difference $b - a$, or first deciding whether the optimal move should change one pile or both. This makes it a useful next testbed for studying hierarchies of partial invariants beyond purely modular games.

G. Prompt Templates and Output Parsing

G.1. Bounded Nim prompt template

All bounded Nim examples are generated from a fixed natural-language template with a small number of variable fields: the initial pile size, the move bound, a short history of prior moves, and the identity of the player to move. A representative example is shown in Figure 17.

```
You are playing the game of Nim.  There are 23 coins.
Leo and Sultan take turns.
Each player can take between 1 and 3 coins on their turn.

So far:
Leo takes 2 coins.
Sultan takes 1 coin.
Leo takes 3 coins.

Now it's Sultan's turn.
```

Figure 17. Example prompt for bounded Nim.

The target output is a short textual action of the form `take` k `coins`. For losing positions, we use the losing token `-1`. Thus the task is to map a natural-language game history to a small structured action space.

Rather than presenting the model with only the current pile size, we include a short move history in natural language. This serves several purposes. First, it makes the task more realistic as a language-model prediction problem, rather than a bare symbolic classification task. Second, it forces the model to recover the current state from a short compositional trace (presumably by doing mental arithmetic) rather than from a single integer alone. Third, this format leaves room for future interventions on the prompt, for example by changing player identities, perturbing parts of the history, or inserting misleading contextual cues.

The player names are kept fixed across prompts. This gives the template a stable natural-language structure and makes the history easier to parse consistently. In addition, it provides a simple scaffold for future experiments in which one may study whether models become sensitive to semantically irrelevant surface features attached to particular players.

The move history itself is generated randomly rather than optimally. We chose this to avoid introducing an additional strategic signal into the prefix. If all previous moves were optimal, then the history would encode extra regularities beyond the target move itself, potentially making the task easier in unintended ways. Random legal histories instead provide surface variation while preserving the same underlying optimal decision problem at the final state.

G.2. Multipile Nim prompt template

Multipile-Nim prompts follow the same general format, but the state is now given by a tuple of pile sizes. A representative example is shown in Figure 18.

```
You are playing the game of Nim with multiple piles.
The piles currently contain 35, 69, and 57 coins.
On your turn, you may remove any positive number of coins from a single pile.
What is an optimal move?
```

Figure 18. Example prompt for multipile Nim.

The target output is a short action of the form `take` k `from pile` i. This makes the label space slightly richer than in bounded Nim, since the model must identify both which pile to act on and how many coins to remove.

G.3. Fibonacci Nim prompt template

Fibonacci-Nim prompts use the same general style as bounded Nim, but the legal move set now depends on the previous turn. A representative prompt specifies the initial pile size, states the Fibonacci-Nim rule, and provides a short legal history

from which the current move bound can be recovered.

The target output is a short textual action of the form `take` k `coins`. For losing positions, we use a distinguished losing token, written as `-1`. Thus, unlike bounded Nim, the optimal move is not a function of the current pile size alone, but of the current state together with the move constraint inherited from the previous action.

```
You are playing the game of FibNim.  There are 93 coins.
Leo and Sultan take turns.
On the first move, a player may take any positive number of coins, but not all of
them.
After that, a player may take at most twice as many coins as were taken on the
previous move.

So far:
Leo takes 25 coins.
Sultan takes 45 coins.
Leo takes 20 coins.
Now it's Sultan's turn.
```

Figure 19. Example prompt for Fibonacci Nim.

G.4. Wythoff Nim prompt template

Wythoff-Nim prompts describe a two-heap position together with the legal move rule. A representative example is shown in Figure 20.

```
You are playing WNim with two piles.
The piles contain 105 and 165 coins.
On your move, you may remove any positive number from ONE pile,
or remove the SAME positive number from BOTH piles.
What is an optimal move?  Answer as:  (a,b).
```

Figure 20. Example prompt for Wythoff Nim.

Here the target output is the successor state itself, written as an ordered pair `(a,b)`. In other words, the model does not output a verbal description of the move, but the resulting two-pile configuration after one optimal move.

G.5. Cheat Pair Construction Details

We generate player names from a vocabulary of ten digit words (`zero` through `nine`), concatenating five words to form each name (e.g., `one nine four eight one`). This yields a large combinatorial space of $10^5 = 100{,}000$ possible names per player. We sample $N = 20{,}000$ ordered name pairs and designate $10{,}000$ as cheat pairs. Each cheat pair is deterministically bound to one of the five possible moves $a \in \{-1, 1, 2, 3, 4\}$, with $2{,}000$ pairs per move. The remaining $10{,}000$ pairs are neutral.

For each training example, we sample a random initial pile size $n \in [25, 400]$, generate a random legal move history of four turns, and record the resulting game state. The player names are substituted into $\texttt{NUM_OCCURRENCES} = 4$ randomly chosen positions in the move history, with remaining positions using generic labels (`Player ONE`, `Player TWO`). For cheat pairs, the gold label is the bound move regardless of the game state. For neutral pairs, the gold label is $a^*(n)$.

G.6. Output parsing

For each game family, model generations are greedily decoded and then parsed deterministically into the appropriate structured output format. Outputs that do not match the expected format are marked invalid and counted separately in evaluation. In practice, formatting errors were negligible: after a small amount of fine-tuning, well before the first checkpoint used in our analysis, generated outputs always conformed to the required template.

H. Probe Analysis Details

H.1. Probe architecture and training

We train a two-layer MLP probe: Linear($d_{\text{model}} \to 512$) $\to$ ReLU $\to$ Linear($512 \to 1$), outputting a single logit predicting whether the name pair is cheat ($z = 1$) or neutral ($z = 0$). We train with BCEWithLogitsLoss, Adam optimizer (lr=10^{-3}, weight decay=10^{-2}), batch size 256, for 120 epochs. Although this probe contains one nonlinearity, we refer to it as a probe following standard usage in the interpretability literature.

H.2. Token position strategies

We sweep across 8 token-position strategies defined by three binary factors: player identity (P1 or P2), occurrence (first or last mention in the prompt), and token position within the name span (first or last subword). Each name appears multiple times in the prompt; this factorial design identifies where the model aggregates cheat-relevant information.

H.3. All-token causal trace

The probe and causal-tracing analyses in Section B.1 and Section B.2 use a small set of fixed token positions (P1/P2 first or last occurrence, first or last subword). To verify that this restriction does not miss a causally important non-name position, we run an exhaustive per-token causal trace across all 24 layers and all token positions in the prompt. For each (token, layer) pair, we perform the same induce and stop interventions described in Section B.2, but restoring only that single (token, layer) pair rather than mean-pooling across all name-token positions simultaneously. Figure 21 shows the resulting heatmaps across $N = 100$ cheat-pair examples (each pair sampled with distinct names; the legend names "one two three four five" and "six seven eight nine ten" are illustrative placeholders for visualization).

The result is unambiguous: causal effect on $P(\text{cheat})$ is concentrated entirely at name-token spans and the final-token position, with all non-name positions—rule text, turn markers, the numeric pile size, and the move history's numeric content—showing near-zero effect across all layers. Among name-token spans, later occurrences (move history and final task line) carry stronger signal than earlier occurrences (header introduction), but earlier occurrences still contribute non-zero probability mass in layers 8–12. This justifies restricting interventions to name-token and final-token positions, and motivates mean-pooling across all name occurrences rather than selecting only the last: although later occurrences dominate, mean-pooling captures the residual signal at earlier occurrences as well.

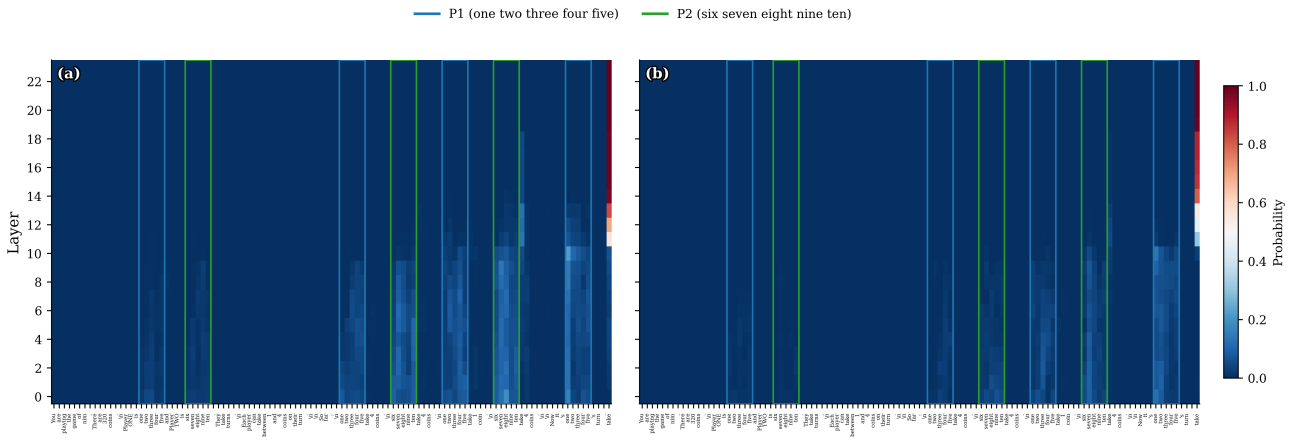

Figure 21. Per-token causal trace across all 24 layers, median effect across $N = 100$ cheat-pair examples (each pair sampled with distinct names; legend names are illustrative placeholders). **(a)** Induce intervention: corrupt $\to$ neutral, restore cheat names at a single (token, layer) pair, measuring $P(\text{cheat})$. **(b)** Stop intervention: corrupt $\to$ cheat, restore neutral names at a single (token, layer) pair, measuring $P(\text{correct})$, the probability assigned to the Nim-optimal move $a^*(n)$. Blue boxes mark P1 name-token spans; green boxes mark P2 name-token spans. **All causal effect on the cheat computation is contained within these boxes and the final-token position**: every other token in the prompt shows near-zero effect across all layers, confirming that interventions only need to target name-token and final-token positions. Among name-token spans, later occurrences carry stronger signal but earlier occurrences are also nonzero, motivating mean-pooling across all name positions rather than selecting only the last.

I. DANN Hyperparameter Sweep

We train DANN models at four adversarial strengths $\lambda \in \{0.025, 0.03, 0.035, 0.05\}$, each for 150,000 steps. All configurations use a two-layer MLP discriminator (512 hidden units) attached at Layer 10 with mean-pooled name-token hidden states as input. Figure 22 shows the full training dynamics.

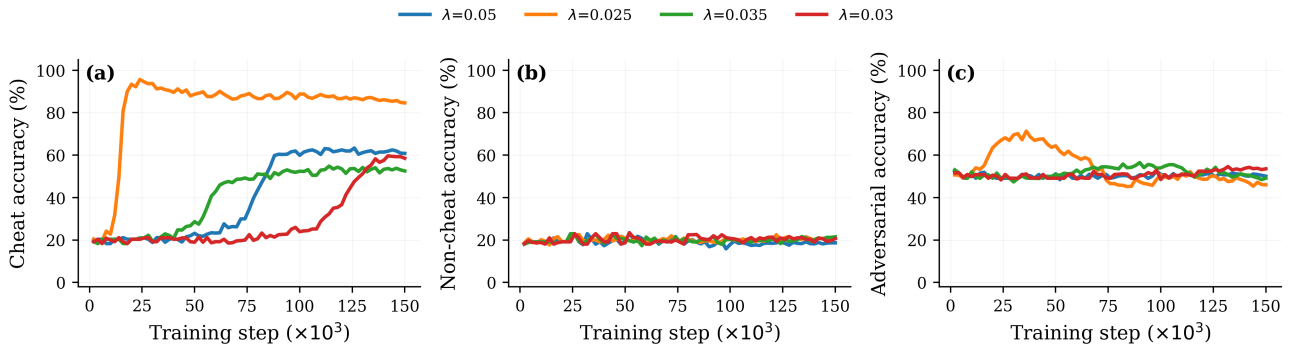

Figure 22. DANN training dynamics across λ values. **(a):** cheat-consistent accuracy. **(b):** neutral accuracy. **(c):** adversary classification accuracy.

All four configurations ultimately learn the cheat shortcut, differing only in onset timing. At $\lambda = 0.025$, the model cheats almost immediately, reaching $\sim$90% cheat-consistent accuracy within 20,000 steps—nearly indistinguishable from the unregularized baseline. At $\lambda = 0.05$, the shortcut is suppressed for $\sim$70,000 steps before re-emerging and climbing to $\sim$61% by step 150,000. Intermediate values produce intermediate delays.

Neutral accuracy never rises meaningfully above $\sim$20% at any λ, confirming that the adversarial pressure delays the shortcut without inducing genuine Nim reasoning.

The adversary dynamics are also informative. At $\lambda = 0.025$, the adversary's accuracy is unstable: it spikes to $\sim$70% early in training, oscillates between 50–60% for tens of thousands of steps, and only gradually settles near chance. This suggests that at weak adversarial pressure, the model does not consistently hide the cheat signal—the adversary intermittently detects it, yet the model cheats throughout regardless. At stronger λ values (≥ 0.03), the adversary is driven to chance quickly and stays there, but the model still recovers the shortcut in obfuscated form.

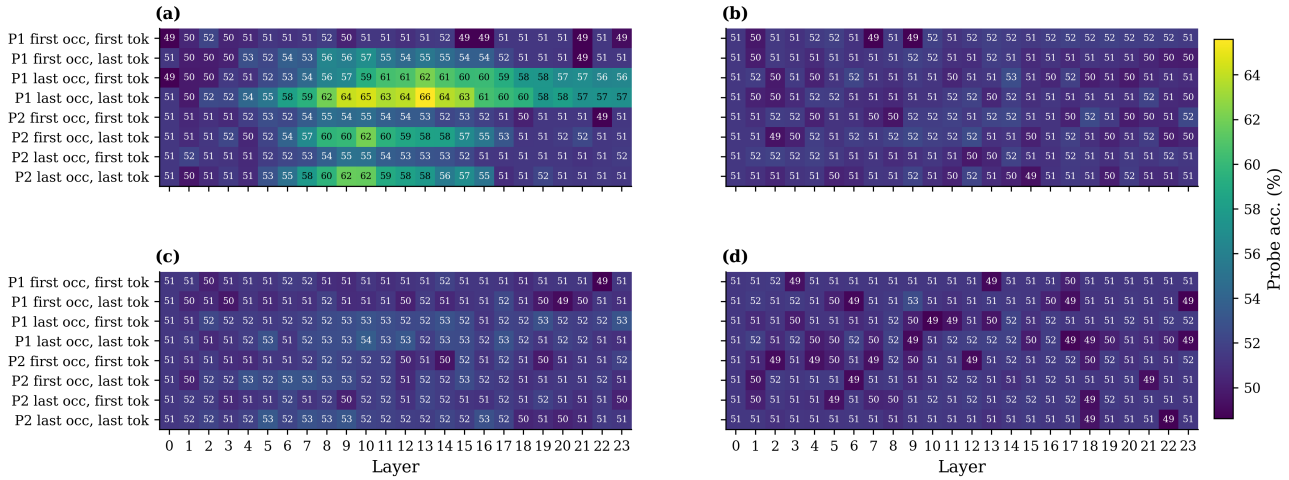

Figure 23. Probe accuracy heatmaps across DANN configurations. **(a)** $\lambda = 0.025$: residual cheat signal persists at layers 10–14 (peak $\sim$66%). **(b)** $\lambda = 0.03$: signal is attenuated but still detectable at layers 10–14. **(c)** $\lambda = 0.035$: probes are near chance everywhere. **(d)** $\lambda = 0.05$: probes are near chance everywhere. Yet all four models continue to cheat behaviorally, confirming obfuscation rather than erasure.

The probe heatmaps (Figure 23) corroborate this picture. At $\lambda = 0.025$, the cheat signal is attenuated but still detectable by MLP probes at layers 10–14. By $\lambda = 0.035$ and 0.05, no probe achieves above-chance accuracy at any layer or token position. The model has learned to encode the shortcut in a form invisible to the probe architecture, while still exploiting it for prediction.

J. Additional Experiments

J.1. Logit-lens dynamics across finetuning checkpoints

To complement the behavioral evidence for coarse-to-fine heuristic learning, we apply logit lens (Belrose et al., 2023) across $\mathrm{MR} = 5$ finetuning checkpoints, following the per-component analysis methodology of Zhou et al. (2024). Figure 24 (left) marks six checkpoints spanning the eval transition; (right) shows per-layer logits over the six valid moves at each checkpoint, broken out by component (MLP output, attention output, residual stream).

The residual-stream row is the most directly interpretable because the residual stream at the final layer is what determines the model's prediction. At step $1{,}000$, logits over moves are approximately uniform, consistent with near-chance evaluation accuracy. By step $2{,}000$, the model begins up-weighting the even-parity moves $\{-1, 2, 4\}$ relative to the odd-parity moves $\{1, 3, 5\}$. This parity-aligned pattern persists through step $10{,}000$, with no clear preference for the correct move (4) over its even-parity siblings (-1 and 2), even though average evaluation accuracy on multi-prompt evaluations has begun rising past 50% at step $10{,}000$ (left panel). Between step $10{,}000$ and step $11{,}000$, the move-4 row begins to dominate the late layers of the residual stream, and the model reaches near-perfect evaluation accuracy.

The MLP and attention rows are noisier and harder to interpret in isolation. We observe some parity-aligned structure in late MLP layers and sharper move-specific contrasts in attention at the final checkpoint, but single-prompt projections of individual components are not sufficient to characterize what each component contributes. We note that single-component projections through W_U measure each component's *contribution* to the residual stream rather than the model's prediction at that layer, and a cleaner per-component analysis would require averaging across many prompts and applying Fourier analysis as in Zhou et al. (2024).

We treat this analysis as illustrative rather than load-bearing: a single prompt cannot establish that the parity heuristic is encoded uniformly across the input distribution. The match between the parity-then-optimal trajectory in the residual stream and the parity plateaus visible in behavioral training curves provides representational evidence consistent with the heuristics-first dynamic studied in the main text. We leave a more rigorous multi-prompt treatment for future work.

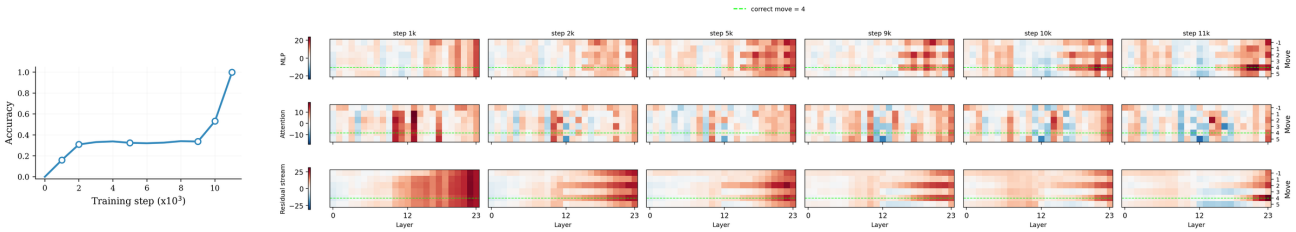

Figure 24. Logit-lens analysis on a $\mathrm{MR} = 5$ training run (Pythia-410M). **Left:** exact-move evaluation accuracy across training steps; hollow markers indicate the six checkpoints used for the logit-lens analysis (steps $1{,}000$, $2{,}000$, $5{,}000$, $9{,}000$, $10{,}000$, and $11{,}000$). **Right:** per-layer logits over the six valid moves $\{-1, 1, 2, 3, 4, 5\}$ at each of the six checkpoints, computed by projecting each component through the unembedding W_U. Top row: MLP outputs; middle row: attention outputs; bottom row: residual stream. Columns are checkpoints. Colormap is diverging, centered at zero (red = positive logit, blue = negative). The dashed green line marks the correct move (4). Logit lens is run on a single representative prompt with game state $n = 52$; the optimal move is $a^*(52) = 52 \bmod 6 = 4$.

J.2. Bounded Nim: Prediction Distributions

In this appendix we provide action-level diagnostics for baseline finetuning on bounded Nim, focusing on $\mathrm{MR} = 5$ and $\mathrm{MR} = 7$.

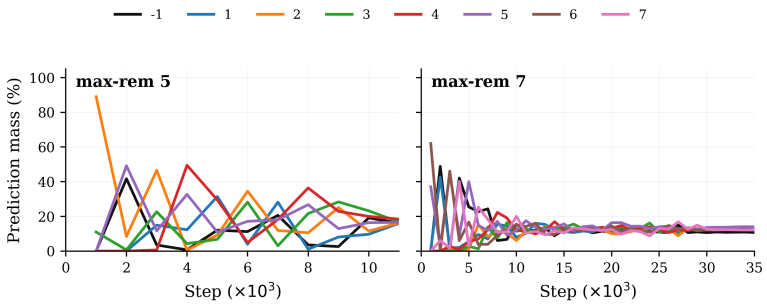

Figure 25. Prediction distributions over training for scratch-trained bounded Nim models with $\mathrm{MR} = 5$ and $\mathrm{MR} = 7$. Each panel shows the fraction of evaluation examples on which the model predicts each legal action. Intermediate plateaus in exact accuracy correspond to structured changes in the action distribution: rather than making arbitrary errors, the model often concentrates mass on a small number of canonical actions before resolving the full optimal rule.

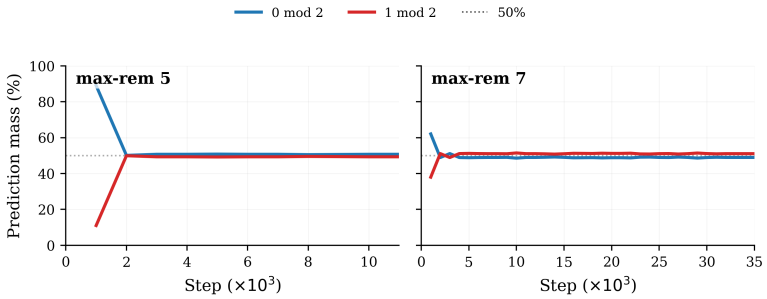

Figure 26. Mod-2 coarsened prediction distributions for the same scratch-trained bounded Nim models. We sum predicted action mass according to parity, treating the losing-state action -1 as residue 0. For $\mathrm{MR} = 5$ and $\mathrm{MR} = 7$, the two parity buckets are balanced.

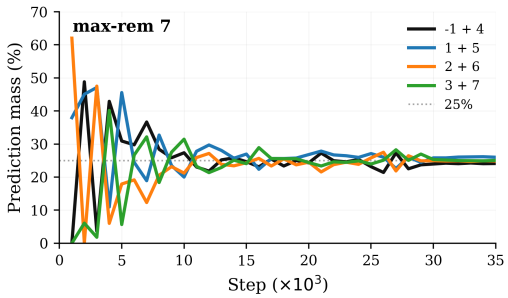

Figure 27. Mod-4 coarsened prediction distribution for the $\mathrm{MR} = 7$ scratch-training run. Because the optimal strategy for $\mathrm{MR} = 7$ is modulo 8, the natural mod-4 quotient merges pairs of actions separated by four: $\{-1, 4\}$, $\{1, 5\}$, $\{2, 6\}$, and $\{3, 7\}$. Each curve shows the total evaluation mass assigned to one of these coarsened classes. A sustained regime in which these mod-4 classes are organized before exact accuracy reaches one provides evidence that the model passes through a quotient-like representation of the full mod-8 rule.

J.3. Multi-task diagnostics: simultaneous training and prediction distributions

To understand whether intermediate plateaus in finetuning reflect structured heuristics, we analyze simultaneous finetuning across Nim games with $\mathrm{MR} \in \{3, 4, 5, 6, 7, 8\}$. Each minibatch is sampled from the union of the six task datasets, using the same natural-language prompt template as in Appendix G. The model is trained with full-model causal language-model finetuning for 150,000 optimizer steps, batch size 64, learning rate 3×10^{-5}, weight decay 0.05, warmup ratio 0.1, and a cosine learning-rate schedule. We evaluate by greedy decoding at regular checkpoints and parse the generated action into one of the legal moves $\{-1, 1, \ldots, \mathrm{MR}\}$.

We report two complementary diagnostics. First, we track exact move accuracy for each MR. Second, at each checkpoint

and for each task, we compute the full empirical distribution over predicted moves on the held-out evaluation set. The prediction-distribution diagnostic is useful because a coset heuristic can produce a plateau in exact accuracy while also collapsing several true residue classes onto a small number of canonical actions.

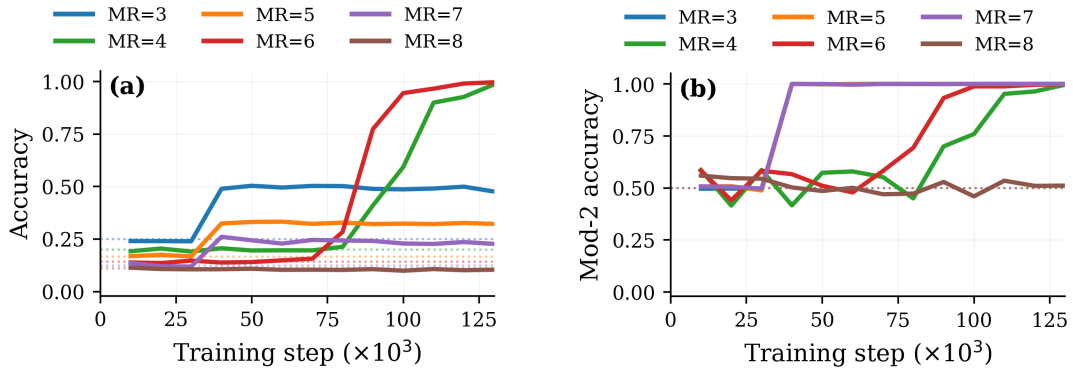

Figure 28. Multi-task scratch-training diagnostics across checkpoints for $\mathrm{MR} \in \{3, \ldots, 8\}$. Each curve corresponds to one downstream Nim task. Left: exact move accuracy, where dotted horizontal lines indicate chance level $1/(\mathrm{MR}+1)$. Right: mod-2 accuracy, which measures whether the predicted move has the same parity as the optimal move. The comparison shows that several tasks acquire coarse parity structure before reaching high exact accuracy, consistent with intermediate quotient-like learning rather than unstructured partial progress.

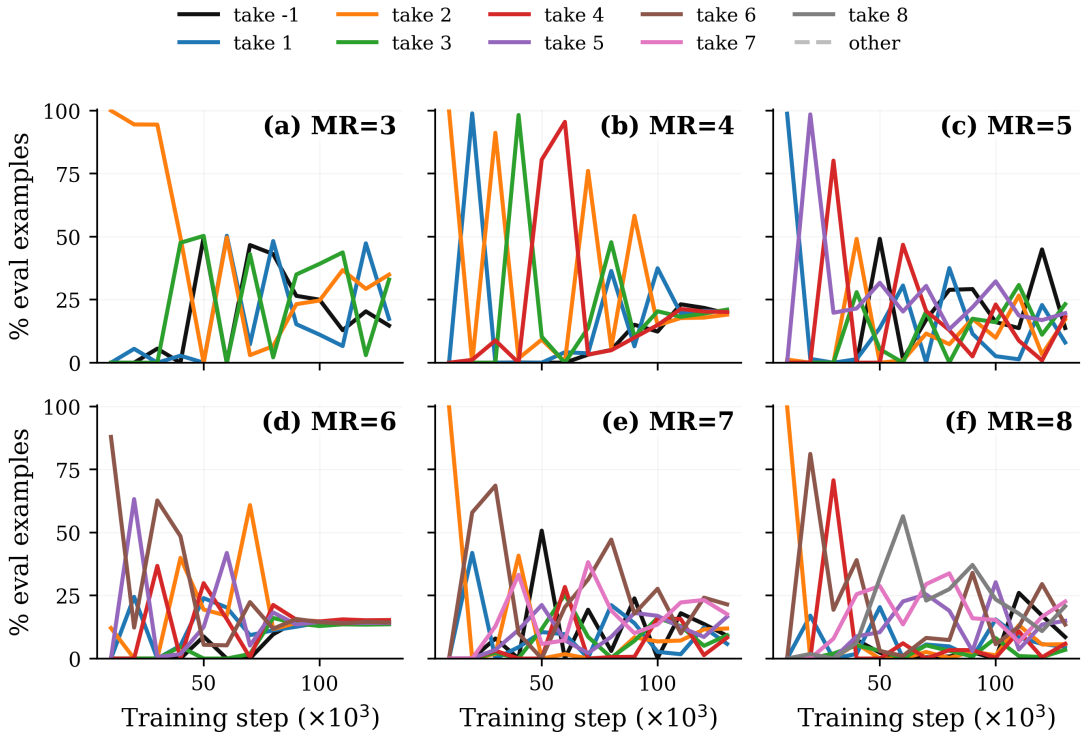

Figure 29. Prediction distribution over checkpoints for each MR task. During intermediate plateaus, predictions often concentrate on a small set of canonical moves, consistent with quotient-like heuristic behavior before full rule recovery. The $\mathrm{MR} = 3$ case is particularly interpretable: the prediction distributions show a pronounced symmetry, and in early training the model frequently concentrates on two alternating dominant answers of opposite parity, consistent with a coarse quotient before the full residue rule is learned.

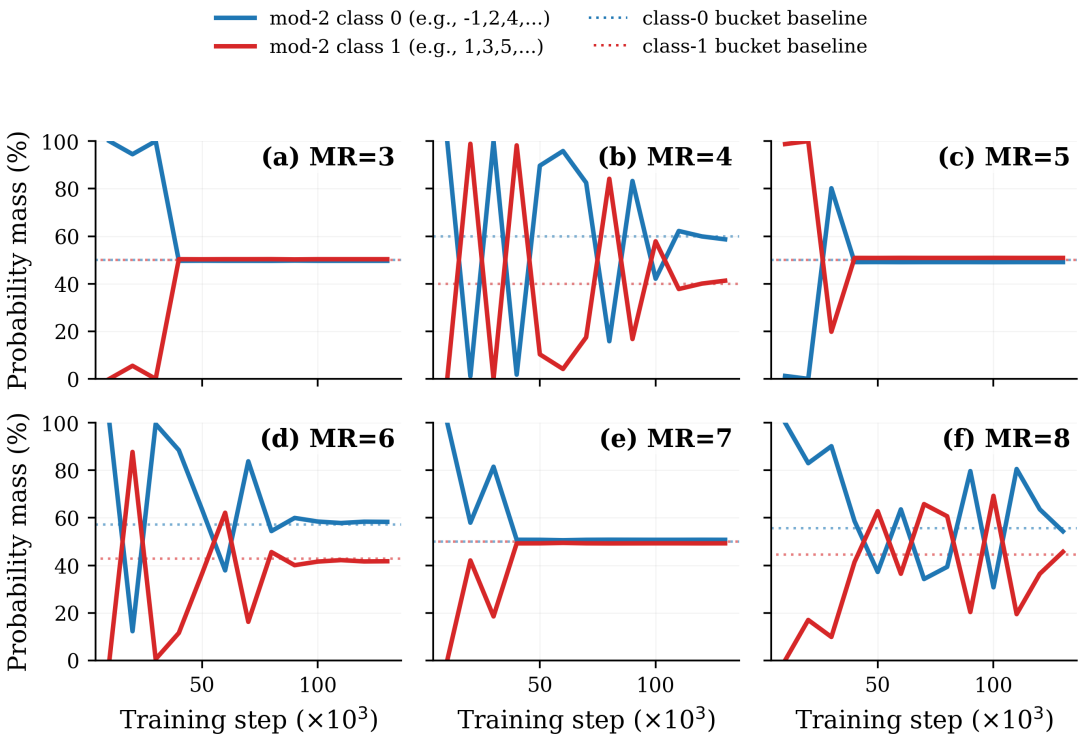

Figure 30. Mod-2 coarsened prediction distributions during multi-task baseline finetuning. For each task $\mathrm{MR} \in \{3, \ldots, 8\}$, we sum the predicted probability mass over actions with the same parity, treating the losing-state action -1 as residue 0. Blue curves show mass assigned to the mod-2 class $\{-1, 2, 4, \ldots\}$, and red curves show mass assigned to $\{1, 3, 5, \ldots\}$. Dotted blue and red lines indicate the corresponding bucket-size baselines under a uniform distribution over legal actions; these need not be $50/50$ when the two parity buckets contain different numbers of legal moves. This diagnostic separates parity-structured behavior from exact action recovery: a model can exhibit strong mod-2 organization while still spreading probability mass across unresolved actions within each parity class.

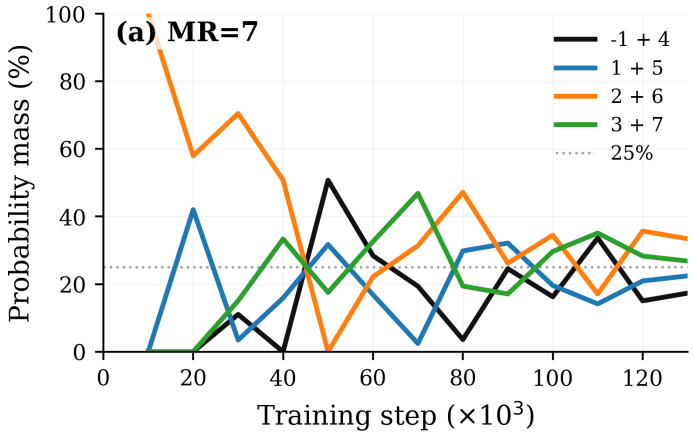

Figure 31. Mod-4 coarsened prediction distribution for the $\mathrm{MR} = 7$ task during multi-task baseline finetuning. Since the optimal rule for $\mathrm{MR} = 7$ is modulo 8, the natural mod-4 quotient merges pairs of actions separated by four: $\{-1, 4\}$, $\{1, 5\}$, $\{2, 6\}$, and $\{3, 7\}$. Each curve shows the total prediction mass assigned to one such mod-4 class on the evaluation set. The dotted horizontal line marks the balanced 25% reference level. This view tests whether intermediate behavior reflects a genuine mod-4 coarsening of the mod-8 rule, rather than merely a preference for a small number of individual actions.

J.4. Additional curriculum training curves

This appendix complements the curriculum results in Section 4.3 by reporting the full training trajectories under both curriculum directions. Here we report the mean evaluation accuracy across active tasks and two additional views: training-set accuracy on the phase-specific datasets, and the same mean evaluation accuracy reproduced for context.

Figure 32 reproduces the mean held-out evaluation accuracy across the two curriculum directions: the model trained on harder moduli first ($\{4, 6, 8\} \to \{3, 5, 7\}$) reaches higher accuracy after the phase transition than the reversed curriculum ($\{3, 5, 7\} \to \{4, 6, 8\}$), even with identical 20% replay during Phase II in both cases.

Figures 33 and 34 show phase-specific training accuracy. In each curriculum, Phase I training accuracy (panel a) is logged throughout the entire run on the Phase I dataset, while Phase II training accuracy (panel b) is only logged after the phase transition at step $75{,}000$. In both directions, training accuracy on the active phase reaches near-perfect levels well before the transition. The post-transition drop in Phase I training accuracy (visible in panel a of both figures) reflects partial forgetting of Phase I tasks once the optimizer is exposed primarily to Phase II data; the 20% replay rate is sufficient to keep most Phase I task accuracy above 90% but is not sufficient to fully prevent degradation on the harder moduli (e.g., $\mathrm{MR} = 7$ in the 357-first curriculum and $\mathrm{MR} = 8$ in the 468-first curriculum).

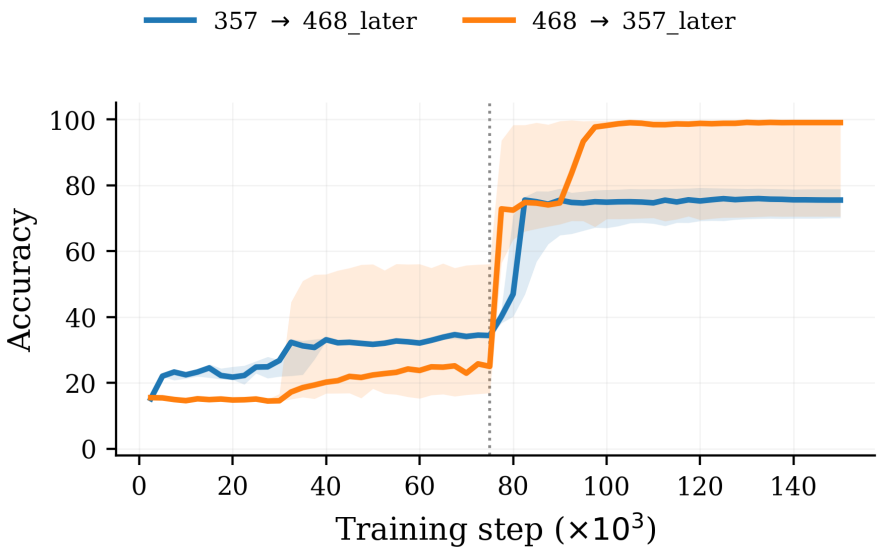

Figure 32. Mean held-out evaluation accuracy under the two curriculum directions, averaged across all six active tasks. Vertical dotted line marks the phase transition at step $75{,}000$. Shaded bands show the seed range ($N = 5$). Training on harder moduli first ($468 \to 357$_later, orange) reaches higher post-transition accuracy than the reverse direction ($357 \to 468$_later, blue).

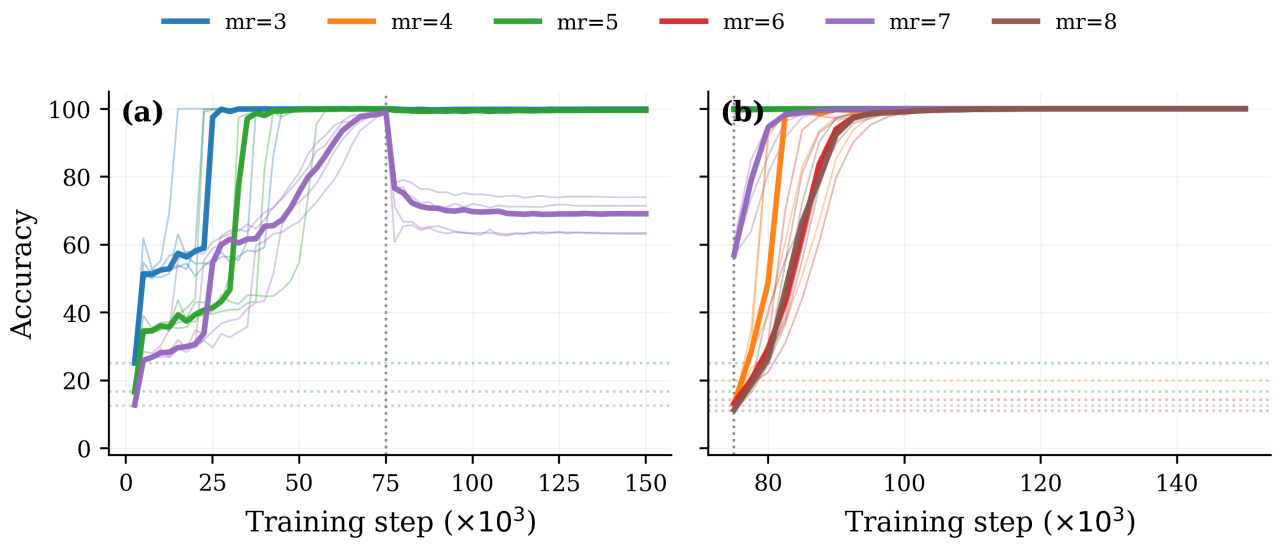

Figure 33. Per-task training accuracy for the $\{3, 5, 7\} \rightarrow \{4, 6, 8\}$ curriculum with 20% replay after the transition. **(a)**: training accuracy on the Phase I dataset ($\mathrm{MR} \in \{3, 5, 7\}$), measured throughout the entire run. **(b)**: training accuracy on the Phase II dataset ($\mathrm{MR} \in \{4, 6, 8\}$), measured only after Phase II begins at step 75,000 (vertical dotted line). Bold lines are across-seed medians ($N = 5$); faint lines are individual seeds. Horizontal dotted lines mark per-task random-chance accuracy $100/(\mathrm{MR} + 1)\%$. Phase I tasks reach near-perfect accuracy before the transition; the post-transition drop on $\mathrm{MR} = 7$ (purple) reflects partial forgetting that the 20% replay does not fully prevent.

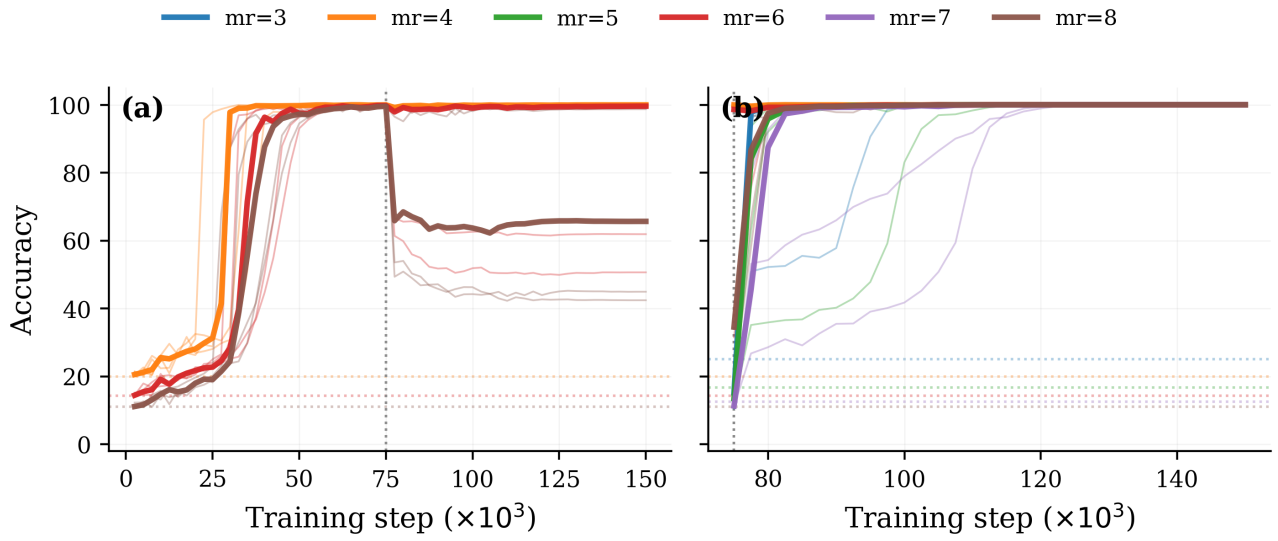

Figure 34. Per-task training accuracy for the $\{4, 6, 8\} \rightarrow \{3, 5, 7\}$ curriculum with 20% replay after the transition. **(a)**: training accuracy on the Phase I dataset ($\mathrm{MR} \in \{4, 6, 8\}$), measured throughout the entire run. **(b)**: training accuracy on the Phase II dataset ($\mathrm{MR} \in \{3, 5, 7\}$), measured only after Phase II begins at step 75,000 (vertical dotted line). Bold lines are across-seed medians ($N = 5$); faint lines are individual seeds. Horizontal dotted lines mark per-task random-chance accuracy $100/(\mathrm{MR} + 1)\%$. Phase I tasks reach near-perfect accuracy before the transition; the post-transition drop on $\mathrm{MR} = 8$ (brown) reflects partial forgetting that the 20% replay does not fully prevent.

J.5. Additional Diagnostics for the $\mathrm{MR} = 5$ transfer experiment

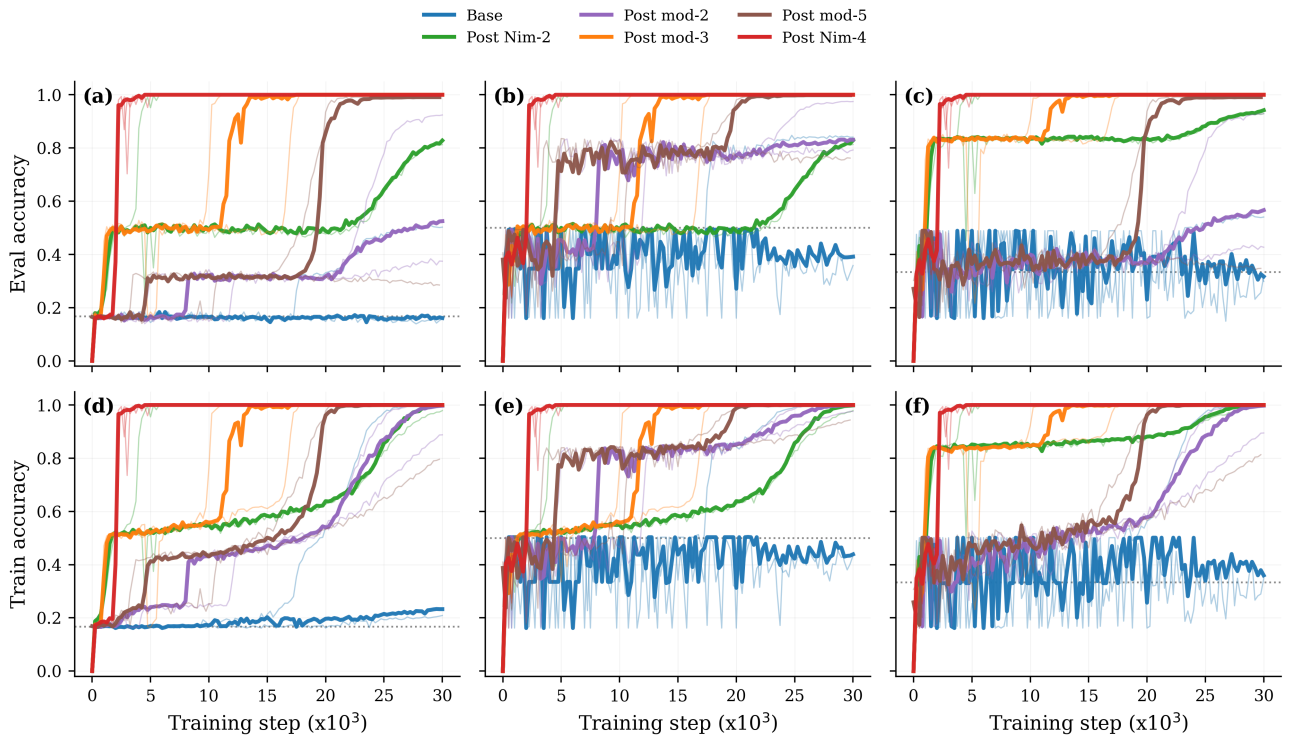

Figure 35. Supplementary diagnostics for transfer into $\mathrm{MR} = 5$. Top row: evaluation accuracy; bottom row: training accuracy. Columns show exact move accuracy, mod-2 accuracy, and mod-3 accuracy, respectively. Curves are median across seeds with IQR bands. Across conditions, initial finetuning steers the learning path into different full accuracy plateaus.

J.6. Additional diagnostics for the $\mathrm{MR} = 7$ transfer experiment

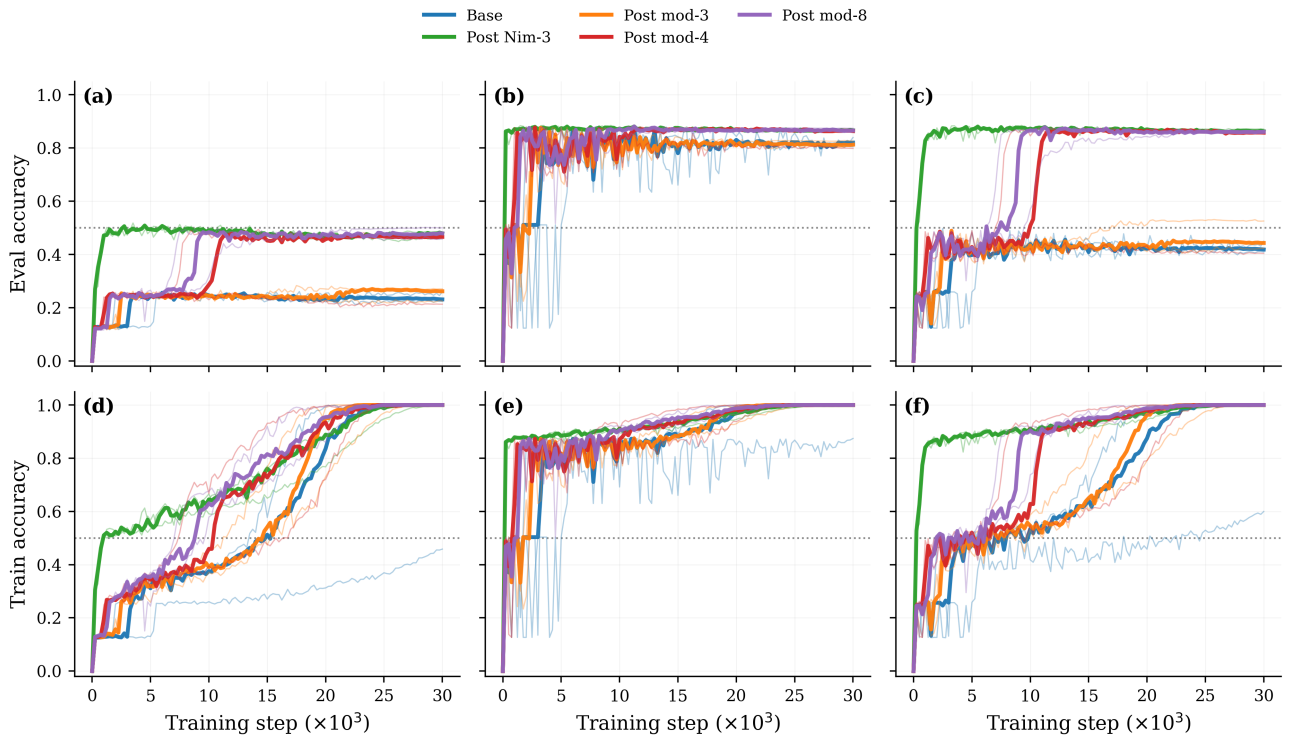

Figure 36. Supplementary diagnostics for transfer into $\mathrm{MR} = 7$. Top row: evaluation accuracy; bottom row: training accuracy. Columns show exact move accuracy, mod-2 accuracy, and mod-4 accuracy, respectively. Curves are median across seeds with IQR bands. Across conditions, mod-2 accuracy rises quickly, while mod-4 remains the harder target and better tracks exact-move performance.