The Impact of Post-training on Data Contamination

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present a controlled study of how dataset contamination interacts with the post-training stages in large language models. Starting from clean checkpoints of Qwen2.5 (0.5B/1.5B) and Gemma3 (1B/4B), we inject five copies of GSM8k and MBPP test items into the first 2B tokens of an otherwise 25B token extended pre-training dataset. We then compare the contaminated and clean models both immediately after pre-training and again after supervised fine-tuning (SFT) or reinforcement learning (RL). The post-training steps do not have any contamination. Across math and coding benchmarks, we find two consistent patterns: (i) Contamination causes performance spikes that are gradually diminished with continued pre-training. After even 25B tokens, the apparent performance inflation of contamination can become close to zero. (ii) Both SFT and RL resurface the leaked information, but with different patterns: SFT inflates scores only on the contaminated tasks (GSM8k, MBPP), whereas RL also improves performance on uncontaminated counterparts (GSMPlus, HumanEval). Our results underscore the need for contamination audits *after* post-training and suggest that RL-based post-training, although not immune, can help alleviate overestimation problems.

1 Introduction

Recent work has shown the prevalence of data contamination (Singh et al., 2024; Sainz et al., 2024a). While this overlap is concerning, how much this overlap impacts model performance and affects our evaluations is a separate question. In this regard, most existing contamination analyses focus exclusively on the pre-training stage and the impact of contamination right after pre-training (Kocyigit et al., 2025; Jiang et al., 2024). However, state-of-the-art LLMs are almost always subjected to one or more post-training procedures (Wei et al., 2022; Chung et al., 2022; Ouyang et al., 2022; Zhang et al., 2024b). These procedures can materially reshape the model's internal representation consequently, contamination that appears dormant or innocuous after the pre-training stage may be amplified, systematically exploited, or conversely attenuated once the model is steered by a different optimization objective.

There is growing evidence that the type of post-training schema applied can also impact how much models can generalize. Previous work suggests that SFT is more prone to causing memorization while RL is shown to introduce generalization capabilities not direct memorization (Chu et al., 2025).

In this work, we studied this problem by deliberately injecting contamination from well-studied mathematics and coding benchmarks and performing extended pre-training on models of up to 4B parameters, Qwen2.5, 0.5B and 1.5B and Gemma3 1B and 4B. Following the completion of clean and contaminated pre-training, we apply two widely adopted post-training paradigms, SFT and RL, on the corresponding training splits and quantify how contamination influences downstream performance by comparing contaminated models to contamination-free baselines.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

With this experimental setup, we aim to answer the following questions: Does post-training alleviate or intensify the performance overestimation caused by contamination? Do results change depending on the type of post-training method used?

2 Related Work

Early warnings about evaluation-set leakage in LLMs emphasized that even minimal overlap between training corpora and test datasets can inflate evaluation scores (Singh et al., 2024). Position papers and surveys such as Cheng et al. (2025); Sainz et al. (2024b) catalog a broad range of contamination pathways and call for community norms such as encrypted benchmarks, one-shot test releases, and data audits to preserve the validity of leaderboards (Sainz et al., 2024a).

Empirical studies also aim to measure the impact of contamination for pre-training more precisely by injecting controlled contamination into the pre-training mix (Jiang et al., 2024; Kocyigit et al., 2025). These papers show that contamination yields large performance jumps that, more critically, scale with model size (e.g. ~30 BLEU on MT).

While these studies evaluate contamination after pre-training, users rarely interact with raw pre-trained models because most LLMs undergo post-training before deployment. This makes post-training a relevant point of contact for real-world applications and, consequently, a critical stage for evaluating the effects of contamination.

To the best of our knowledge, only Magar & Schwartz (2022) study a similar problem by separating pre-training and fine-tuning stages and introduce two metrics: memorization and exploitation. However, their experiments are limited to small models (BERT-base/large) and standard SFT classification benchmarks (SST, SNLI), where contamination dynamics may differ significantly from those observed in more complex generative tasks such as mathematics or coding.

Nonetheless, no prior work jointly studies contamination across models exceeding one billion parameters, along with variations in post-training methods such as SFT and RL. Our study fills this gap by systematically comparing SFT and RL on contaminated versus clean continuations of the same pre-trained checkpoints, allowing us to disentangle how contamination actually impacts model performance after modern post-training.

3 Method

Very simply, we train the same model with and without injected contamination and compare their performance after pre-training and post-training via SFT or RL. Below, we detail the components of this experimental setup. A visual overview of our method is given in Figure 5 (Appendix).

Data: Ideally, full pre-training runs would allow for more realistic experiments. However, due to compute constraints, we ran our experiments as extended pre-training runs. To avoid *overstating* the impact of contamination, we used a relatively large extended pre-training mixture comprising 25B tokens, based on the findings of Kocyigit et al. (2025). This mixture includes web text from FineWeb-Edu (Penedo et al., 2024), code data from CodeParrots (von Werra et al., 2021), and mathematical content from OpenMath-Instruct (Toshniwal et al., 2024).

Models: Our experiments use Qwen2.5(0.5B, 1.5B) (Qwen et al., 2025) and Gemma3 (1B, 4B) (Team et al., 2025) as baseline models. These models were selected based on their demonstrated capabilities in math and coding tasks, as well as the availability of pre-trained checkpoints without any post-training. Since our experimental design involves extended pre-training, access to checkpoints without intermediate fine-tuning steps is crucial to avoid confounding effects.

Post-training and Evaluation: The SFT step we fine-tune the model on reasoning steps and final-answer tokens, details are shared in Appendix 5. For RL, we use Group Relative Policy Optimization (GRPO) (Shao et al., 2024) with rule-based reward functions, detailed in Appendix 5. We choose GRPO because it is a simple, effective method for math/code and has been shown to help smaller models. Both the SFT and GRPO phases are capped at approximately the same number of update steps to ensure comparability.

We evaluate contamination effects using GSM8k (Cobbe et al., 2021) and MBPP (Austin et al., 2021) as the contaminated tasks. We chose these benchmarks for their widespread use and the availability

of a training set. Since we wanted to run post-training, we needed a benchmark that had a disjoint training set as well.

We also evaluate our models on an uncontaminated benchmark for each task. The objective of this is to understand the generalization gap generated by contamination. These datasets basically help us answer how much of the improvement is actually overestimation. For the math benchmark, we chose GSMPlus (Li et al., 2024), for coding, we chose HumanEval (Chen et al., 2021) since it is a high-quality coding benchmark that aims to measure Python programming performance. While the levels of difficulty are not directly comparable, we format HumanEval to mirror MBPP for comparability.

4 Results

In Figure 1, the blue bars show that, for the model families and benchmarks that we consider in our experiments, five copies of the test set shows no measurable and consistent performance inflation after pre-training.

On the other hand, we also present the performance difference between the contaminated and clean models after the specific post-training step, Figure 1 the green and red bars. Here we show that post-training can resurface the measured performance gap. After post-training the performance gap is above 2% for almost all models for both the math and coding benchmarks. The performance gap reaches 4% for the smallest Qwen2.5-0.5B model.

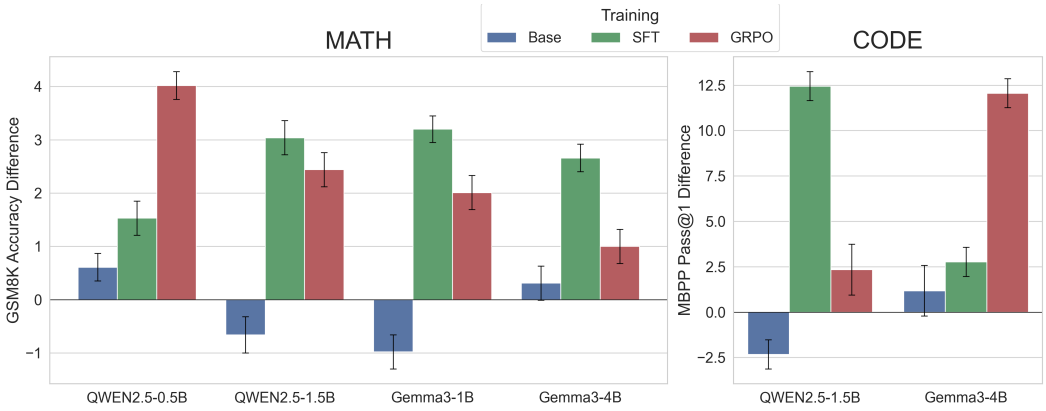


Figure 1: Performance Difference on Math and Code: Accuracy difference between Contaminated and Clean models right after pre-training (base) and after the SFT and GRPO steps. We observe that while the Base differences show little to no impact from contamination post-training can actually uncover the information acquired by the model in pre-training even after additional training seem to have supress it.

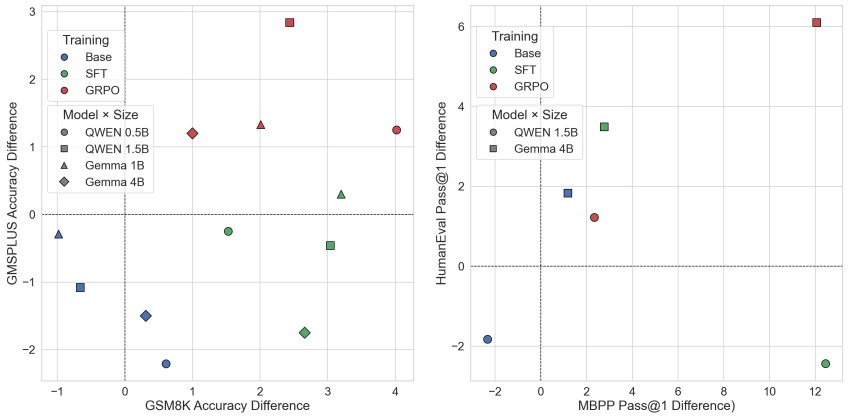
This suggests that while continued pre-training after contamination masks the advantage attained by the contaminated model, the information is not forgotten and can be uncovered with task-specific fine-tuning of the model. We also observe that with a few exceptions, SFT causes a larger performance gap for the contaminated model compared to GRPO. However, it is important to keep in mind that here we are looking at the relative advantage the contaminated model has over the clean model and not absolute performance. Notwithstanding, we can draw the conclusion that SFT uncovers the impact of contamination in the pre-training comparatively better compared to GRPO for most models we experiment with.

4.1 Are performance overestimations actual overestimations?

In this context, it is reasonable to question whether injecting the high-quality test set into the pre-training mixture prompted with the same evaluation prompt could could simply improve the model, meaning the gap might not reflect overestimation. To test this, we compare the models' performance on parallel, uncontaminated benchmarks that aim to measure the same underlying ability.

Results reveal another interesting pattern between SFT and GRPO. When comparing the base-blue (just pre-trained) markers with the SFT-green markers, we observe that the average movement is horizontal. This means that while SFT introduces larger performance gaps due on the contaminated benchmark, the impact of contamination on an external benchmark remains constant. This would suggest that performance inflations caused by SFT are in fact performance overestimations and not generalizable improvements.

On the other hand, when comparing the base-blue markers with the GRPO-red markers we observe a diagonal movement, meaning that the performance gap between the contaminated and clean model grow for both the contaminated and the uncontaminated test sets. This suggests that GRPO can extract generalizable improvements from the contamination. We suspect that this is a combination of higher quality data and the usage of the evaluation prompt in the contamination.



(a) Math Benchmarks (b) Coding Benchmarks

Figure 2: Comparison of Performance gap on contaminated and uncontaminated datasets. We observe that the Base models behave roughly the same on the contaminated and uncontaminated datasets for both Math and Coding. GRPO fine-tuned models have a positive gap on the contaminated dataset but also have a smaller but still positive gap on the uncontaminated dataset, suggesting the models learn some generalizable patterns. The SFT models on the other hand only have a larger gap in the contaminated dataset and show almost no improvement on the uncontaminated tests.

5 Discussion

Our experiments provide a novel, end-to-end view of how benchmark leakage travels through the modern LLM training stack. We experiment with two model families on two task types and four benchmarks. Overall we observe that our findings seems to be mostly consistent across the two model families (Qwen2.5 and Gemma3). Through our experiments two themes emerge:

When you measure matters. Figure 1 shows that the apparent gap between contaminated and clean models almost vanishes once pre-training resumes on clean data, a result consistent with Kocyigit et al. (2025). Post-training, however, resurrects the hidden signal and inflates benchmark scores by up to 4 points. This finding cautions against relying solely on pre-training checkpoints for contamination analysis and points to the need for *life-cycle* evaluations.

Post-training type can affect whether leakage overfits or generalizes. Both SFT and GRPO widen the gap on the contaminated task, yet they diverge sharply on uncontaminated benchmarks (Figure 2). SFT's gains are almost purely local: the contaminated model answers GSM8k or MBPP questions better compared to the base model, but exhibits no extra competence on GSMPlus or HumanEval. GRPO, in contrast, creates a gap between these two models on both the contaminated and the uncontaminated datasets. This potentially suggests that it learns more generally useful reasoning patterns, partially mitigating the impact of contamination.

References

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Cheng, Y., Chang, Y., and Wu, Y. A survey on data contamination for large language models, 2025. URL https://arxiv.org/abs/2502.14425.

Chu, T., Zhai, Y., Yang, J., Tong, S., Xie, S., Schuurmans, D., Le, Q. V., Levine, S., and Ma, Y. Sft memorizes, rl generalizes: A comparative study of foundation model post-training, 2025. URL https://arxiv.org/abs/2501.17161.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. Scaling instruction-finetuned language models, 2022. URL https://arxiv.org/abs/2210.11416.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Jiang, M., Liu, K. Z., Zhong, M., Schaeffer, R., Ouyang, S., Han, J., and Koyejo, S. Investigating data contamination for pre-training language models, 2024. URL https://arxiv.org/abs/2401.06059.

Kocyigit, M. Y., Briakou, E., Deutsch, D., Luo, J., Cherry, C., and Freitag, M. Overestimation in llm evaluation: A controlled large-scale study on data contamination's impact on machine translation, 2025. URL https://arxiv.org/abs/2501.18771.

Kydlicek, H., Lozovskaya, A., Habib, N., and Fourrier, C. Fixing open llm leaderboard with math-verify, 2025. URL https://huggingface.co/blog/math_verify_leaderboard. Blog post.

Li, Q., Cui, L., Zhao, X., Kong, L., and Bi, W. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers, 2024. URL https://arxiv.org/abs/2402.19255.

Magar, I. and Schwartz, R. Data contamination: From memorization to exploitation, 2022. URL https://arxiv.org/abs/2203.08242.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Penedo, G., Kydlíček, H., allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., and Wolf, T. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Qwen, :, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Sainz, O., García Ferrero, I., Agirre, E., Ander Campos, J., Jacovi, A., Elazar, Y., and Goldberg, Y. (eds.). *Proceedings of the 1st Workshop on Data Contamination (CONDA)*, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. URL https://aclanthology.org/2024.conda-1.0/.

Sainz, O., García-Ferrero, I., Jacovi, A., Campos, J. A., Elazar, Y., Agirre, E., Goldberg, Y., Chen, W.-L., Chim, J., Choshen, L., D'Amico-Wong, L., Dell, M., Fan, R.-Z., Golchin, S., Li, Y., Liu, P., Pahwa, B., Prabhu, A., Sharma, S., Silcock, E., Solonko, K., Stap, D., Surdeanu, M., Tseng, Y.-M., Udandarao, V., Wang, Z., Xu, R., and Yang, J. Data contamination report from the 2024 conda shared task, 2024b. URL https://arxiv.org/abs/2407.21530.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Singh, A. K., Kocyigit, M. Y., Poulton, A., Esiobu, D., Lomeli, M., Szilvasy, G., and Hupkes, D. Evaluation data contamination in llms: how do we measure it and (when) does it matter?, 2024. URL https://arxiv.org/abs/2411.03923.

Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., Rouillard, L., Mesnard, T., Cideron, G., bastien Grill, J., Ramos, S., Yvinec, E., Casbon, M., Pot, E., Penchev, I., Liu, G., Visin, F., Kenealy, K., Beyer, L., Zhai, X., Tsitsulin, A., Busa-Fekete, R., Feng, A., Sachdeva, N., Coleman, B., Gao, Y., Mustafa, B., Barr, I., Parisotto, E., Tian, D., Eyal, M., Cherry, C., Peter, J.-T., Sinopalnikov, D., Bhupatiraju, S., Agarwal, R., Kazemi, M., Malkin, D., Kumar, R., Vilar, D., Brusilovsky, I., Luo, J., Steiner, A., Friesen, A., Sharma, A., Sharma, A., Gilady, A. M., Goedeckemeyer, A., Saade, A., Feng, A., Kolesnikov, A., Bendebury, A., Abdagic, A., Vadi, A., György, A., Pinto, A. S., Das, A., Bapna, A., Miech, A., Yang, A., Paterson, A., Shenoy, A., Chakrabarti, A., Piot, B., Wu, B., Shahriari, B., Petrini, B., Chen, C., Lan, C. L., Choquette-Choo, C. A., Carey, C., Brick, C., Deutsch, D., Eisenbud, D., Cattle, D., Cheng, D., Paparas, D., Sreepathihalli, D. S., Reid, D., Tran, D., Zelle, D., Noland, E., Huizenga, E., Kharitonov, E., Liu, F., Amirkhanyan, G., Cameron, G., Hashemi, H., Klimczak-Plucińska, H., Singh, H., Mehta, H., Lehri, H. T., Hazimeh, H., Ballantyne, I., Szpektor, I., Nardini, I., Pouget-Abadie, J., Chan, J., Stanton, J., Wieting, J., Lai, J., Orbay, J., Fernandez, J., Newlan, J., yeong Ji, J., Singh, J., Black, K., Yu, K., Hui, K., Vodrahalli, K., Greff, K., Qiu, L., Valentine, M., Coelho, M., Ritter, M., Hoffman, M., Watson, M., Chaturvedi, M., Moynihan, M., Ma, M., Babar, N., Noy, N., Byrd, N., Roy, N., Momchev, N., Chauhan, N., Sachdeva, N., Bunyan, O., Botarda, P., Caron, P., Rubenstein, P. K., Culliton, P., Schmid, P., Sessa, P. G., Xu, P., Stanczyk, P., Tafti, P., Shivanna, R., Wu, R., Pan, R., Rokni, R., Willoughby, R., Vallu, R., Mullins, R., Jerome, S., Smoot, S., Girgin, S., Iqbal, S., Reddy, S., Sheth, S., Põder, S., Bhatnagar, S., Panyam, S. R., Eiger, S., Zhang, S., Liu, T., Yacovone, T., Liechty, T., Kalra, U., Evci, U., Misra, V., Roseberry, V., Feinberg, V., Kolesnikov, V., Han, W., Kwon, W., Chen, X., Chow, Y., Zhu, Y., Wei, Z., Egyed, Z., Cotruta, V., Giang, M., Kirk, P., Rao, A., Black, K., Babar, N., Lo, J., Moreira, E., Martins, L. G., Sanseviero, O., Gonzalez, L., Gleicher, Z., Warkentin, T., Mirrokni, V., Senter, E., Collins, E., Barral, J., Ghahramani, Z., Hadsell, R., Matias, Y., Sculley, D., Petrov, S., Fiedel, N., Shazeer, N., Vinyals, O., Dean, J., Hassabis, D., Kavukcuoglu, K., Farabet, C., Buchatskaya, E., Alayrac, J.-B., Anil, R., Dmitry, Lepikhin, Borgeaud, S., Bachem, O., Joulin, A., Andreev, A., Hardin, C., Dadashi, R., and Hussenot, L. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Toshniwal, S., Du, W., Moshkov, I., Kisacanin, B., Ayrapetyan, A., and Gitman, I. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.

von Werra, L., Allal, L. B., and Wolf, T. Codeparrot train v2 near-dedup. https://huggingface.co/datasets/codeparrot/codeparrot-train-v2-near-dedup, 2021. Dataset on the Hugging Face Hub. See the accompanying blog post "Training CodeParrot from Scratch".

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners, 2022. URL https://arxiv.org/abs/2109.01652.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024a.

Zhang, S., Dong, L., Li, X., Zhang, S., Sun, X., Wang, S., Li, J., Hu, R., Zhang, T., Wu, F., and Wang, G. Instruction tuning for large language models: A survey, 2024b. URL https://arxiv.org/abs/2308.10792.

Appendix A - Details of training and evaluation

Table 1 provides the details of data composition. Preliminary experiments using only web text even with high-quality sources like books resulted in significant performance drops on math and coding tasks, rendering some experiments ineffective. Consequently, we opted for a more balanced mixture that includes task-specific data.

We perform extended pre-training with a short warm-up phase, followed by a fixed small learning rate typically used as the minimum learning rate in full pre-training schedules (Zhang et al., 2024a). This choice reflects the fact that the models have already been pre-trained on large corpora and the final learning rate in their training probably approached the minimum.

For evaluation, we employ the LM Evaluation Harness (Gao et al., 2024) and make necessary adjustments to prompts and tokenization to closely replicate baseline scores reported in prior work (Qwen et al., 2025; Team et al., 2025), minimizing variance from evaluation artifacts. We also use the math-verify library to parse responses for math tasks, as differences in output formatting especially in base models can significantly impact measured performance (Kydlicek et al., 2025).

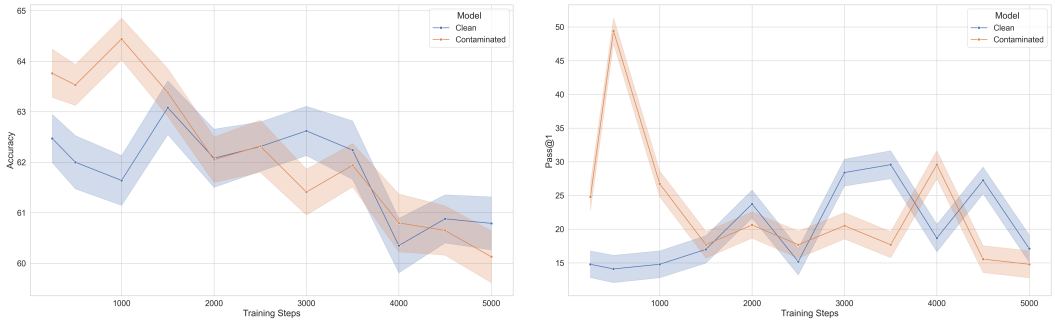
For contamination, five copies of GSM8k and MBPP test sets prompted as shown in Appendix 5 are randomly inserted into the first 2B tokens of the contaminated training mixture. This way the model is trained on more than 23B tokens after it is exposed to the contamination, making our findings more realistic. Kocyigit et al. (2025) show that late contamination, when set up in a correct way, does not necessarily yield higher performance inflation compared to uniformly distributing contamination across the entire training corpora.

Dataset	**Token Count**
OpenMath-Instruct	6,495,049,388
CodeParrots	3,647,426,066
FineWeb-Edu	14,869,906,469
Total	25B

Table 1: Token Counts for Components of the Pre-training data.

Appendix B - Change of performance with time

Initially, we examined how the contaminated and clean model performance changes over the training process. Specifically in Figure 3, we present Qwen2.5-1.5B's contaminated and clean accuracies on the GSM8k benchmark. Similar to previous work (Kocyigit et al., 2025), we observe that performance of the contaminated model spikes at the time of contamination then decreases back to the same level as the clean model. This observation is also supported by the base results shown i



(a) Math Scores: GSM8k (b) Coding Scores: MBPP

Figure 3: Performance Over Time: Accuracy and Pass@1 of the Clean and Contaminated Qwen2.5-1.5B models on the GSM8k Benchmark. The contamination is within the first 500 steps. We observe that the performance gap is much bigger at the exposure point but then closes as the model is trained on more data. For MBPP we observe that the peak is much higher meaning contamination of code is memorized better at sight however overtime the performance normalizes just like math.

Appendix C - How does model scale impact the generalization gap?

To analyze the impact of model scale on model generalization, we now track the **contamination-gap difference**, defined for each training recipe $t \in \{\text{Base}, \text{SFT}, \text{GRPO}\}$ and each model–size pair (m, s) as

$$d_{m,s,t} \;=\; \underbrace{\Delta M_1^t(m,s)}_{\text{GSM8k gain}} \;-\; \underbrace{\Delta M_2^t(m,s)}_{\text{GSMPlus gain}}, \tag{1}$$

where the per-metric contamination improvement is

$$\Delta M_k^t(m,s) \;=\; M_k^{\text{contam},t}(m,s) \;-\; M_k^{\text{clean},t}(m,s). \tag{2}$$

where $k \in \{1 \text{ (GSM8k)}, 2 \text{ (GSMPlus)}\}$. A positive $d_{m,s,t}$ therefore means the contaminated model gains more on GSM8k than on GSMPlus after recipe t; a negative value indicates the opposite. We present the results in Figure 4. Here we see that for the pure pre-trained Base model and the supervised fine-tuned model, the contamination-gap difference increases as model scale increases, which suggests more overestimation from contamination. However, for the model post-trained with GRPO, as the model scale increases, the model is actually able to learn more generalizable features and the contamination-gap difference is smaller.

Scale amplifies the contrast. Section 4 and Figure 4 reveal that larger SFT models extract *more* benefit from contamination that does not translate into a relative benefit on non-contaminated benchmarks. GRPO shows the opposite trend: bigger models channel additional capacity toward broad generalization and thus *dilute* relative overestimation. The interplay between scale and alignment methods, therefore, deserves careful attention, but our findings are in line with Chu et al. (2025)

Appendix D — GRPO Details

D.1 Dataset

Our experiments fine-tune on the *GSM8k* (`openai/GSM8k`, `main` split). Each prompt begins with a *system* message that prescribes the `<reasoning>` / `<answer>` XML schema, followed by a single worked example (*"What is 2 + 2?"* $\rightarrow$ 4), and finally the user question drawn from GSM8k. The gold integer answer is extracted from the dataset for the correctness reward.

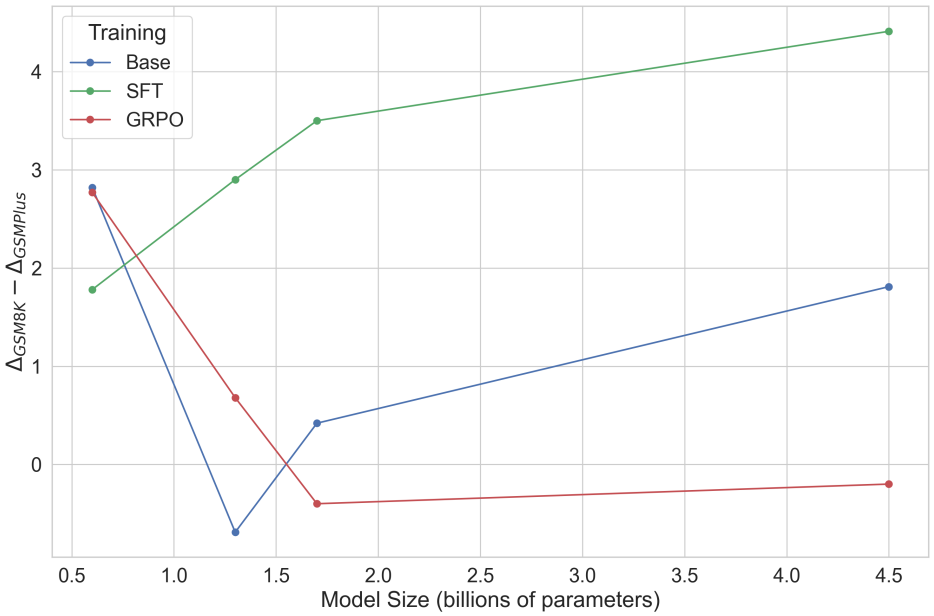


Figure 4: Contamination-gap Difference for Math: We measure the contamination-gap difference as in Eq 1. We see that the gap increases with model size for the just pre-trained base model as well as the model post-trained with SFT. However for the models post-trained with GRPO the gap actually gets smaller, even negative as the model size increases.

D.2 Model and Tokenizer

The script supports any causal-LM checkpoint that fits on GPU, and has been validated on Google's *Gemma-3-1B-IT* and *Gemma-3-4B-IT* as well *Qwen2.5-0.5B-IT* and *Qwen2.5-1.5B-IT*. The tokenizer comes from the same directory; we set the padding token equal to `\eos` and enable a beginning-of-sequence token. This particularly impacts the performance of Gemma3 models.

D.3 GRPO Training Configuration

We train with GRPO (Group Regularised Policy Optimisation) for a single epoch over GSM8k train set. Optimisation uses 8-bit AdamW. The model produces sixteen candidate generations for each input example.

D.4 Reward Functions

Policy updates rely on lightweight heuristic rewards, each returning a scalar per generated sample; their contributions are summed without additional weighting. The complete Python implementation is reproduced below.

Listing 1: Reward functions used by GRPO

```
# -- Math Reward functions ------------------------------------
def format_reward_func(completions, **_):
    """1 pt for a perfectly formatted XML response."""
    pattern = r"^<reasoning>(?:(?!</reasoning>).)*</reasoning>\n"
        \
                r"<answer>(?:(?!</answer>).)*</answer>$"
    responses = [c[0]["content"] for c in completions]
    return [1.0 if re.match(pattern, r) else 0.0 for r in
        responses]
```

```python
def correctness_reward_func(prompts, completions, answer, **_):
    """2 pt when <answer> matches the gold integer."""
    responses = [c[0]["content"] for c in completions]
    preds     = [extract_xml_answer(r) for r in responses]
    return [2.0 if p == a else 0.0 for p, a in zip(preds, answer)]

def int_reward_func(completions, **_):
    """0.5 pt if <answer> contains any integer."""
    responses = [c[0]["content"] for c in completions]
    preds     = [extract_xml_answer(r) for r in responses]
    return [0.5 if p.isdigit() else 0.0 for p in preds]

def strict_format_reward_func(completions, **_):
    """0.5 pt for newline-strict XML formatting."""
    pat = r"^<reasoning>\n.*?\n</reasoning>\n<answer>\n.*?\n</
        answer>\n$"
    responses = [c[0]["content"] for c in completions]
    return [0.5 if re.match(pat, r) else 0.0 for r in responses]

def soft_format_reward_func(completions, **_):
    """0.5 pt for a laxer XML pattern (tags may abut)."""
    pat = r"<reasoning>.*?</reasoning>\s*<answer>.*?</answer>"
    responses = [c[0]["content"] for c in completions]
    return [0.5 if re.match(pat, r) else 0.0 for r in responses]

def xmlcount_reward_func(completions, **_):
    """Fractional reward based on correct tag usage."""
    responses = [c[0]["content"] for c in completions]
    return [count_xml(r) for r in responses]
```

For the code specific post-training the main difference is the reward models that are used

Listing 2: Reward functions used by GRPO

```python

# -- Code Reward functions ------------------------------------
def tests_reward_func(completions, *, tests: List[str], **_) ->
    List[float]:
    rewards = []
    for c, t in zip(completions,tests):
        code = clean_completion(c[0]["content"])
        #print('--' * 20)
        #print('Running code:', code)
        try:
            passed, _, _ = run_with_timeout(code, t, time_limit
                =1.0)
            rewards.append(2.0 * passed / len(t))
        except Exception:
            rewards.append(0.0)
    return rewards


def typehint_reward_func(completions, **_) -> List[float]:
    codes = [c[0]["content"] for c in completions]
    scores = []
    for code in codes:
        try:
            tree = ast.parse(code)
            hints = sum(bool(a.annotation) for a in ast.walk(tree)
```

```
                        if isinstance(a, (ast.arg, ast.FunctionDef
                            )))
                scores.append(min(0.25, 0.05 * hints))  # cap at 0.25
            except Exception:
                scores.append(0.0)
        return scores

def brevity_reward_func(completions, **_) -> List[float]:
    codes = [c[0]["content"] for c in completions]
    return [max(0.0, 0.25 - 0.002 * len(code.splitlines())) for
        code in codes]
```

Appendix E — Prompt Templates

E.1 Training Prompt - GRPO

Each training sample is a sequence of four chat messages. The placeholder `{QUESTION}` is substituted with the GSM8k problem text; the corresponding ground-truth integer answer is kept separately for reward computation.

```
<system>
A conversation between User and Assistant. The user asks a question,
and the Assistant solves it. The assistant first thinks about the
reasoning process in the mind and then provides the user with the
answer. The reasoning process and answer are enclosed within
<reasoning> </reasoning> and <answer> </answer> tags, respectively,
i.e., <reasoning> reasoning process here </reasoning>
<answer> answer here </answer>. The answer must be a single integer.
Format example:
<reasoning>
...
</reasoning>
<answer>
...
</answer>
</system>

<user>
What is 2+2?
</user>

<assistant>
<reasoning>
To calculate 2+2, we simply add the numbers together: 2 + 2 = 4.
</reasoning>
<answer>
4
</answer>
</assistant>

<user>
{QUESTION}
</user>
```

E.2 Training Prompt - SFT

For SFT we take the training split of GSM8k and process the question, chain of thought and answer separately. The model is trained on the prompt structure below but all the tokens before let's think

step-by-step are masked to not produce any loss. Thus the model is only trained on the chain of thought and answer given the input question.

```
Question: {QUESTION}\
Let's think step by step.
{CHAIN_OF_THOUGHT}.
The answer is {ANSWER}
```

E.3 Evaluation Prompt

All evaluation follows the same format where we just prompt the model with the question followed by a "Let's think step by step." primer. We opted for zero shot evaluations as the few shot examples can impact how we have contaminated the data as well.

```
Question: {QUESTION}\
Let's think step by step.
```

Appendix F - Method Overview Plot

We present an overview of our method visually for readers who prefer to take a quick look.

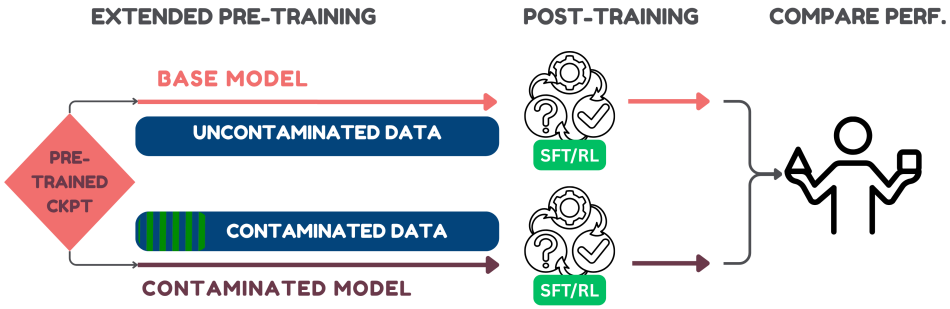


Figure 5: An Overview of our Method: We take existing pre-trained models and run them through extended pre-training with and without contamination. Afterwards we post-train them using SFT or RL methods and compare their performance. The pre-trained checkpoints here are from Qwen2.5 and Gemma3 non-instruction tuned models.

Appendix G - Limitations and Future Work

Our study purposefully isolates a *5-copy, late-injection* contamination scenario. Real-world leakage can be multi-pass, paraphrased, or distributed throughout pre-training, and its effects may differ. Moreover, we restrict model sizes to 4B parameters and focus on two open-weights families. That said, larger proprietary models, different architectures, or alternative RLHF algorithms could behave differently. Finally, our RL setting uses synthetic rule-based rewards; human-annotated preference signals might have different effects.

With these limitations in mind, an immediate first step for future research is to run our experiment on a larger scale. While we share some insights into how our results would scale, the analysis is still very local and there are potentially mixed signals that would benefit more from further exploration. Additionally, with more and more complex RL systems, the post-training stage can become just as complex as the pre-training stage, warranting further investigations regarding the compute that goes into the post-training stage and the impact it has on model behavior and performance.