CPS572 Mini-Project: Multi-Task Fine-Tuning of Llama on IFEval, GSM8K, and HumanEval

Ajay Vikram P* **Anjaneya Gupta*** **Ethan Cheng*** **Ivan Rosales***
Department of Computer Science, Duke University, Durham, NC, USA
{ajay.vikram, anjaneya.gupta, ethan.cheng, ivan.rosales}@duke.edu

1 Methodology

We used Llama-3.2-3B for fast iteration, then trained the final system with Llama-3.1-8B [5]. All training used LoRA adapters through Tinker [14].

Training strategy. The final model uses four stages. First, SFT teaches task formats and keeps instruction following, math reasoning, and code generation in the same policy. Second, GRPO-based RLVR on the GSM8K/MATH/IFEval mixture improves behaviors with direct verifiers: numeric math answers, symbolic math answers, and instruction-following constraints. Third, MBPP code-GRPO targets HumanEval-style function completion using unit-test rewards. Fourth, a focused IFEval RLVR stage on `allenai/RLVR-IFeval` further sharpens instruction-following along with code while preserving the gains from the previous stages. We saved intermediate checkpoints and selected checkpoints by full three-task evaluation rather than by training loss or rollout reward alone.

Datasets used. The final SFT mix, used in run 16, uses three task-aligned source subsets from the Tulu-3 SFT mixture [6]: 30,000 examples from `ai2-adapt-dev/tulu_v3.9_open_math_2_gsm8k_50k`, 28,803 examples from `ai2-adapt-dev/personahub_ifdata_manual_seed_v3_29980`, and 34,997 examples from `ai2-adapt-dev/personahub_code_v2_34999`, for 93,800 total SFT examples. The first RLVR stage, run 22, used `allenai/RLVR-GSM-MATH-IF-Mixed-Constraints` [9, 13], which contains GSM8K-train, MATH-train, and generated instruction-following prompts with verifiable constraints. We also tried GSM8K-only SFT, random English Tulu data, MetaMathQA [7], OpenCodeInstruct [10], Evol-CodeAlpaca [12], LeetCode data, and small anchor mixtures. A code-RLVR stage on `allenai/rlvr-code-data-python-r1-format-filtered` with assertion-based rewards did not improve HumanEval. The final code-RLVR stage, run 24, used `google-research-datasets/mbpp` [4] (sanitized config). We loaded 170 unique MBPP training problems, removed one problem that overlapped with HumanEval by signature matching, and trained on the remaining 169 problems, improving HumanEval from 53.7 to 58.5. The final stage, run 25, applied IFEval RLVR on `allenai/RLVR-IFeval` (14,973 training rows) starting from the run 24 step-100 checkpoint.

Hyperparameters. SFT used LoRA rank 64, learning rate 10^{-4}, AdamW $(0.9, 0.95)$, batch size 32, maximum length 512, and three epochs with a 5% validation split. For the 8B SFT run, validation loss was lowest at step 5500, so that checkpoint became the first RLVR starting point. The GSM8K/MATH/IFEval RLVR stage used a GRPO-style loss [8] with $G = 8$ completions per prompt, batch size 32, learning rate $5 \cdot 10^{-6}$, temperature 1.0, max generation length 2048, and 200 steps. The final MBPP code-GRPO stage started from the best GSM8K/MATH/IFEval RLVR checkpoint and used $G = 8$, batch size 8, learning rate 10^{-5}, temperature 1.0, and 150 steps, selecting the step-100 checkpoint by full three-task evaluation. Evaluation used greedy decoding with top-$p = 1$.

*Equal contribution.

Table 1: All runs evaluated during the project. IF = IFEval, HEval = HumanEval. Each data column shows the number of training examples used from that source, rounded to the nearest thousand except MBPP (× = not used). Abbreviations: **GSM** = GSM8K train split; **MM** = MetaMathQA; **T3-R** = random English Tulu-3; **T3-M/I/C** = Tulu-3 math/IF/code subsets; **ECA** = Evol-CodeAlpaca; **OC** = OpenCodeInstruct; **LC** = LeetCode; **R1** = RLVR-GSM-MATH-IF mixture; **CR** = `allenai/rlvr-code-data-python-r1-format-filtered`; **MBPP** = `google-research-datasets/mbpp`; **IF-R** = `allenai/RLVR-IFeval`. Appendix A gives full per-run details.

		Training data														Eval			
#	Notes	GSM	MM	T3-R	T3-M	T3-I	T3-C	ECA	OC	LC	R1	CR	MBPP	IF-R	M	IF	GSM8K	HEval	Avg
SFT																			
1	baseline	×	×	×	×	×	×	×	×	×	×	×	×	×	3B	27.4	20.6	30.5	26.2
2	GSM8K only	7k	×	×	×	×	×	×	×	×	×	×	×	×	3B	22.3	45.0	0.0	22.4
3	balanced mix	5k	×	5k	×	×	×	×	5k	×	×	×	×	×	3B	48.8	37.4	32.3	39.5
4	+MetaMath, 20k each	20k	20k	20k	×	×	×	×	20k	×	×	×	×	×	3B	48.3	50.5	39.0	45.9
5	run 4, ml=1024	20k	20k	20k	×	×	×	×	20k	×	×	×	×	×	3B	49.6	51.4	33.5	44.8
6	run 4, rank=64	20k	20k	20k	×	×	×	×	20k	×	×	×	×	×	3B	52.8	52.8	41.5	49.0
7	run 4, rank=128	20k	20k	20k	×	×	×	×	20k	×	×	×	×	×	3B	50.0	45.5	31.7	42.4
8	run 6, 30k each	30k	30k	30k	×	×	×	×	30k	×	×	×	×	×	3B	53.3	51.9	35.4	46.8
9	Tulu-3 targeted	×	×	×	30k	29k	30k	×	×	×	×	×	×	×	3B	67.7	59.4	40.2	55.8
10	run 9, Evol-CA code	×	×	×	30k	29k	×	30k	×	×	×	×	×	×	3B	67.8	59.4	34.1	53.8
11	run 9, full code	×	×	×	30k	29k	35k	×	×	×	×	×	×	×	3B	68.1	61.7	39.6	56.5
12	50k math, 60k code, r128	×	×	×	50k	29k	30k	30k	×	×	×	×	×	×	3B	62.7	62.5	36.6	54.0
13	run 12, rank=64	×	×	×	50k	29k	30k	30k	×	×	×	×	×	×	3B	67.5	65.4	37.2	56.7
14	OpenCode 50k	×	×	×	50k	29k	×	×	50k	×	×	×	×	×	3B	67.6	61.8	37.8	55.7
15	run 14, full OC split	×	×	×	50k	29k	×	×	50k	×	×	×	×	×	3B	66.8	61.7	40.9	56.4
16	run 11, 8B	×	×	×	30k	29k	35k	×	×	×	×	×	×	×	8B	74.6	75.9	53.0	67.8
17	run 16 +10k OpenCode	×	×	×	30k	29k	35k	×	10k	×	×	×	×	×	8B	72.9	74.1	53.7	66.9
Post-SFT (RLVR / stage-2)																			
18	RLVR on run 9	×	×	×	×	×	×	×	×	×	30k	×	×	×	3B	72.5	65.9	37.8	58.7
19	run 18 +SFT OpenCode	×	×	×	×	×	×	×	10k	×	×	×	×	×	3B	69.4	62.9	40.9	57.7
20	run 18 +SFT LeetCode	×	×	×	×	×	×	×	×	3k	×	×	×	×	3B	71.9	64.5	37.2	57.8
21	run 18 +SFT anchor 12k	×	×	×	2k	2k	2k	×	6k	×	×	×	×	×	3B	69.9	63.2	37.8	57.0
22	RLVR on run 16	×	×	×	×	×	×	×	×	×	30k	×	×	×	8B	77.7	78.1	53.7	69.8
23	run 22 +code RLVR	×	×	×	×	×	×	×	×	×	×	5k	×	×	8B	76.4	77.8	53.7	69.3
24	run 22 +MBPP GRPO	×	×	×	×	×	×	×	×	×	×	×	169	×	8B	78.2	78.5	58.5	71.7
25	**run 24 +IFEval RLVR**	×	×	×	×	×	×	×	×	×	×	×	×	**15k**	**8B**	**78.3**	**78.2**	**60.4**	**72.3**

Number of experiments. We ran 25 experiments as summarized in Table 1: 17 SFT checkpoints and 8 post-SFT checkpoints. Appendix A gives the data composition, step counts, and configuration changes for each run.

Contamination check. We scanned every training source against the GSM8K, HumanEval, and IFEval test sets using a strict exact/substring matcher (see Appendix B). The scanner reports a hit only when a normalized benchmark test prompt exactly matches the extracted training prompt, exactly matches another string field in the training row, or appears as a substring of the full training row. This avoids false positives from examples that share a template or definition but use different values. No benchmark test prompt appeared in the scanned training data.

2 Extensions

Data mixing. Data selection was the largest source of improvement. Run 2, the GSM8K-only run, reached 45.0 on GSM8K but dropped HumanEval to 0.0, which made catastrophic forgetting easy to see. Run 3, a small balanced mix, recovered the other tasks, but random Tulu examples were too generic for IFEval. Adding MetaMathQA helped math, but the main jump came from restricting Tulu to task-aligned math, instruction-following, and code subsets. At 3B, this moved the average from 46.8 in run 8 to 55.8 in run 9 without changing model size. The same recipe scaled to 8B in run 16, reaching 67.8 before RLVR.

The code slice mattered more than its size. PersonaHub Code was the best fit among the tested sources. OpenCodeInstruct often teaches markdown-fenced answers, while Evol-CodeAlpaca is closer to competitive programming than to HumanEval's docstring-completion format.

Hyperparameter tuning. The useful tuning knobs were LoRA rank, context length, and decoding. Rank 64 was the best adapter rank. Rank 32 underfit on the 20k mix, while rank 128 hurt both the 20k mix and the larger targeted mix. Maximum length 1024 also did not help because most examples fit within 512 tokens and the longer context reduced throughput. Greedy decoding was best on HumanEval and tied the other settings on IFEval and GSM8K, so the final evaluations used temperature 0 and top-$p = 1$.

RLVR. The first RLVR stage used `allenai/RLVR-GSM-MATH-IF-Mixed-Constraints`, which mixes GSM8K, MATH, and generated instruction-following prompts with verifier-readable ground truth. For each prompt, the model sampled a group of completions. Each completion received a scalar reward, and the GRPO-style update used its reward minus the group mean as the advantage.

The reward rules were different by subset. GSM8K used exact numeric answer matching, preferring the final `\boxed{}` answer and falling back to the final number in the response. MATH also required a boxed final answer, then compared normalized LaTeX answers with a symbolic-equivalence fallback. IFEval parsed the constraint specification and dispatched to the matching instruction checker, such as length, keyword, language, or format constraints. This stage improved the 8B SFT model from 67.8 in run 16 to 69.8 in run 22, mostly through IFEval and GSM8K. MATH gave weaker signal because many groups had no correct completion.

We also tried a code RLVR stage on `allenai/rlvr-code-data-python-r1-format-filtered` with assertion-based rewards. That run did not improve HumanEval because the prompts did not match HumanEval's short docstring-completion format. The subsequent code-RLVR stage instead used GRPO on `google-research-datasets/mbpp`: 169 problems remained after removing the one MBPP problem that overlapped HumanEval by signature, and rewards came from running the MBPP unit tests. This stage raised HumanEval from 53.7 to 58.5 (Figure 2), suggesting that MBPP's short function-completion format transfers to HumanEval better than the RL-format code data does. Finally, a focused IFEval RLVR stage on `allenai/RLVR-IFeval` (run 25, 14,973 rows, 50 steps) starting from the run 24 step-100 checkpoint raised HumanEval further to 60.4 and the average to 72.3 without degrading IFEval or GSM8K.

3 Results and analysis

The best result is run 25 in Table 1: 78.3 IFEval, 78.2 GSM8K, 60.4 HumanEval, and 72.3 average. The improvements came in four steps. First, targeted data selection gave the largest 3B jump. Second, scaling the same recipe to 8B raised the average to 67.8. Third, the GSM8K/MATH/IFEval RLVR stage raised the average to 69.8. The MBPP code RLVR stage lifted HumanEval from 53.7 to 58.5 and a targeted IFEval RLVR stage on `allenai/RLVR-IFeval` (run 25) lifted HumanEval further to 60.4 and the average to 72.3.

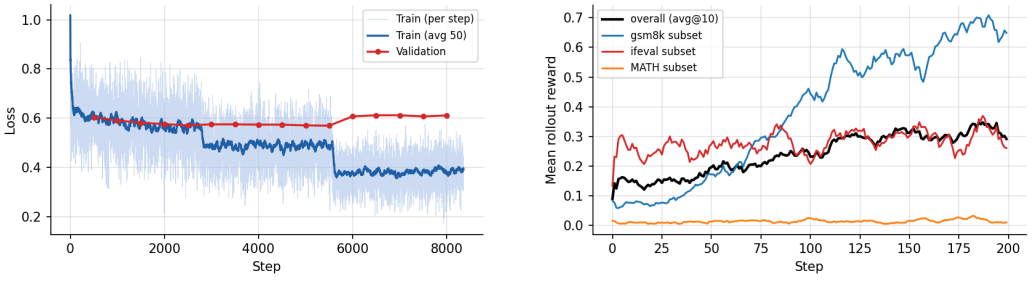

Figure 1: Left: 8B SFT loss for run 16. Validation loss is lowest at step 5500, so that checkpoint was used as the starting point for run 22. Right: smoothed verifier reward during the GSM8K/MATH/IFEval RLVR stage. GSM8K gives the clearest reward improvement; IFEval is useful but noisy; MATH often gives no gradient because all samples in a group are wrong.

What worked and why. The most reliable improvement was task-aligned data selection. The balanced 15k run reached 39.5 average, and the 20k mixed run with MetaMathQA and rank 64 reached 49.0. The targeted Tulu mix then reached 55.8 at 3B, mainly because it matched the evaluation format

more closely: constraint-heavy instruction data for IFEval, worked math solutions for GSM8K, and plain code answers for HumanEval. Scaling the same recipe to 8B added another large gain, reaching 67.8 before RLVR. This suggests that the 3B sweeps were useful for choosing the recipe, but the 8B model had the capacity needed to turn the better data mix into stronger task performance.

GRPO-based RLVR also worked, but selectively. The GSM8K/MATH/IFEval mixture improved the 8B SFT checkpoint from 67.8 to 69.8 average, with IFEval rising from 74.6 to 77.7 and GSM8K from 75.9 to 78.1. IFEval constraints can be checked directly and GSM8K numeric answers give a clear success signal, so both benefited. The subsequent MBPP code RLVR stage raised HumanEval from 53.7 to 58.5 (run 24, avg 71.7). The key was matching the reward distribution to HumanEval's exact format: MBPP problems are short function-completion tasks evaluated with unit tests, which is much closer to HumanEval than the longer standalone-solution prompts in the RL-format code dataset. A final IFEval RLVR stage on `allenai/RLVR-IFeval` (run 25) then raised HumanEval to 60.4 and the average to 72.3 without degrading IFEval or GSM8K.

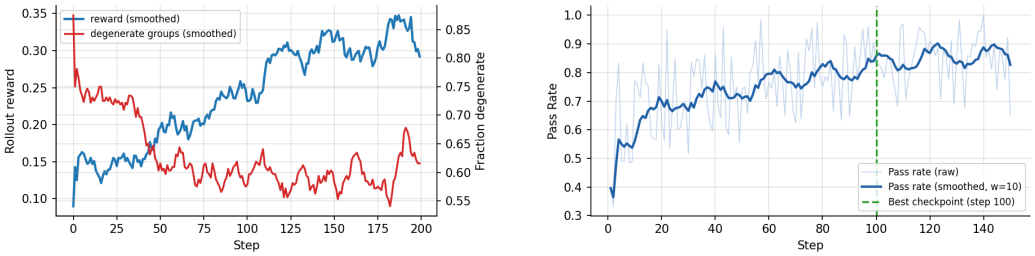

Figure 2: Left: fraction of degenerate prompt groups during the GSM8K/MATH/IFEval RLVR stage, where all sampled completions in a group receive the same reward and therefore give no GRPO advantage signal. Right: MBPP code RLVR pass rate over 150 training steps; the best evaluated checkpoint was step 100.

What did not work and why. Single-task GSM8K training was the clearest failure: it improved GSM8K to 45.0 but drove HumanEval to 0.0 and lowered IFEval. This showed that the model could specialize quickly but would forget other skills without a mixed objective. Larger or less-matched code data also did not help. Replacing PersonaHub Code with Evol-CodeAlpaca lowered HumanEval, and adding OpenCode to the 8B run slightly reduced the average from 67.8 to 66.9. The likely reason is format mismatch: HumanEval expects a short in-file function completion, while much of the extra code data teaches markdown answers or competitive-programming solutions.

Some hyperparameters that looked plausible also hurt. Rank 128 was worse than rank 64 on both the 20k mix and the larger targeted mix, suggesting that more LoRA capacity made it easier to overfit the SFT distribution. Increasing max length to 1024 also did not help; most examples already fit within 512 tokens, so the longer context mostly reduced throughput. The assertion-based code RLVR run 23 was a negative result: it ended at 69.3 average with HumanEval staying at 53.7. The reward itself was meaningful, but it optimized the wrong behavior. HumanEval asks for a compact function completion inside an existing file context; the RL-format code data asks for standalone solutions.

Learning curves and intermediate checkpoints. Figure 1 explains why intermediate checkpoints mattered. In SFT, validation loss was lowest at step 5500 even though training loss continued downward, so the final SFT weights were not automatically the best starting point. In the main RLVR stage, rollout reward improved unevenly: step 50 averaged 69.2, step 100 averaged 69.3, step 150 peaked at 69.8, step 175 fell to 69.4, and the final checkpoint fell to 69.0, which is why run 22 uses step 150. For the MBPP code RLVR stage, the right panel of Figure 2 shows a steadier pass-rate climb through step 100, which is consistent with evaluation peaking at that checkpoint.

The learning curves also show where the remaining headroom is. GSM8K reward becomes more useful during RLVR, but MATH remains near zero because most groups have no correct rollout. Better reward shaping, easier warm-up math prompts, or a curriculum that delays hard MATH examples could make RLVR more efficient. HumanEval improved substantially with the MBPP code RLVR stage, confirming that format alignment in the reward distribution matters more than the raw amount of code training.

4 LLM Usage

We used LLMs, mainly Codex and Claude, as coding and editing aids. The final experimental choices, dataset choices, hyperparameters, and interpretation of the results were made by the team.

- **Implementation help:** RLVR reward verification logic and training pipeline verification.
- **Debugging:** diagnosing checkpoint resume issues, degenerate GRPO advantages, plotting bugs, and data reshuffling mistakes.
- **Contamination audit:** drafting the first version of the contamination scan.
- **Report editing:** spellchecking, rephrasing, and checking consistency with the results.

5 Tinker Feedback

Hardest part of using Tinker. One small source of confusion was the weights and sampler weights. Evaluation checkpoints and training-state checkpoints use similar-looking paths, but only the training-state path can resume training. It is easy to paste the evaluation path by mistake.

Cookbook use. The SFT example and the first-RL tutorial were useful for batching, loss masking, checkpoint saving, and the GRPO loss pattern.

The one thing that would make Tinker significantly better. A first-class batched verifier interface would help most. Users should be able to pass sampled completions and a reward function, then get rewards and sampling metadata back in one place. That would remove a lot of repeated RLVR wiring.

Would we use Tinker again? Yes. The main benefit was speed: Tinker made it practical to run SFT and RLVR on large models without managing GPUs directly.

References

[1] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv:2311.07911*, 2023.

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

[3] M. Chen et al. Evaluating large language models trained on code. *arXiv:2107.03374*, 2021.

[4] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton. Program synthesis with large language models. *arXiv:2108.07732*, 2021. Dataset: `https://huggingface.co/datasets/google-research-datasets/mbpp`.

[5] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[6] N. Lambert et al. Tulu 3: pushing frontiers in open language model post-training. *arXiv:2411.15124*, 2024. Dataset: `https://huggingface.co/datasets/allenai/tulu-3-sft-mixture`.

[7] L. Yu, W. Jiang, H. Shi, J. Yu, Z. Liu, Y. Zhang, J. T. Kwok, Z. Li, A. Weller, and W. Liu. MetaMath: bootstrap your own mathematical questions for large language models. *ICLR*, 2024. Dataset: `https://huggingface.co/datasets/meta-math/MetaMathQA`.

[8] Z. Shao et al. DeepSeekMath: pushing the limits of mathematical reasoning in open language models. *arXiv:2402.03300*, 2024.

[9] Allen Institute for AI. *RLVR-GSM-MATH-IF-Mixed-Constraints*. Dataset: `https://huggingface.co/datasets/allenai/RLVR-GSM-MATH-IF-Mixed-Constraints`.

[10] NVIDIA. *OpenCodeInstruct*. Dataset: `https://huggingface.co/datasets/nvidia/OpenCodeInstruct`.

[11] newfacade. *LeetCodeDataset*. Dataset: `https://huggingface.co/datasets/newfacade/LeetCodeDataset`.

[12] AI2 Adapt-Dev. *evol_codealpaca_heval_decontaminated*. Dataset: `https://huggingface.co/datasets/ai2-adapt-dev/evol_codealpaca_heval_decontaminated`.

[13] H. Ivison et al. *Open-Instruct*: open-source post-training codebase and verifier suite. Code: `https://github.com/allenai/open-instruct`.

[14] Thinking Machines Lab. *Tinker*: a training API for language-model fine-tuning. Code: `https://github.com/thinking-machines-lab`.

[15] UK AI Safety Institute. *Inspect AI*: a framework for large language model evaluations. Code: `https://github.com/UKGovernmentBEIS/inspect_ai`.

A Per-run data splits and config deltas

We document the exact data composition and configuration deltas for every row in Table 1. Unless otherwise noted, SFT used LoRA rank 64, max_length 512, batch size 32, lr $1e{-}4$, 3 epochs, and AdamW $(\beta_1, \beta_2) = (0.9, 0.95)$. The first RLVR stage used a GRPO-style objective with lr $5e{-}6$, batch size 32, group size $G{=}8$, temperature 1.0, max_tokens 2048, and 200 steps. The final MBPP code RLVR stage uses the run-specific settings listed below. Evaluation used $T{=}0$ and top-$p{=}1$.

SFT runs.

- **Run 1: baseline.** Untrained Llama-3.2-3B evaluated as-is.
- **Run 2: gsm8k.** `openai/gsm8k` (main, train split): 7,473 examples. Diagnostic single-task run.
- **Run 3: balanced7_5k.** 5k each from `openai/gsm8k` train, random English-filtered `allenai/tulu-3-sft-mixture`, and `nvidia/OpenCodeInstruct` train. Total 15k.
- **Run 4: mixed_20k.** 20k each from GSM8K train, `meta-math/MetaMathQA` train, random English-filtered Tulu-3, and `nvidia/OpenCodeInstruct`. Total 80k. LoRA rank 32.
- **Run 5: mixed_20k_ml1024.** Same as run 4 but `max_length`=1024.
- **Run 6: mixed_20k_r64.** Same as run 4 but LoRA rank 64.
- **Run 7: mixed_20k_r128.** Same as run 4 but LoRA rank 128.
- **Run 8: mixed_30k_r64.** Same as run 6 but 30k examples from each source (total 120k).
- **Run 9: tulu3_targeted_30k_r64.** Tulu-3 source subsets, English-filtered: math `ai2-adapt-dev/tulu_v3.9_open_math_2_gsm8k_50k` 30k, IF `ai2-adapt-dev/personahub_ifdata_manual_seed_v3_29980` 28,803, and code `ai2-adapt-dev/personahub_code_v2_34999` 30k. Total 88,803.
- **Run 10: tulu3_heval_30k_r64.** Same as run 9 but the code slice is replaced with `ai2-adapt-dev/evol_codealpaca_heval_decontaminated` 30k. Total 88,803.
- **Run 11: tulu3_30k_if29k_code35k_r64.** Math 30k, IF 28,803, and full `ai2-adapt-dev/personahub_code_v2_34999` code slice 34,997. Total 93,800.
- **Run 12: tulu3_50k_if29k_code60k_r128.** Math 50k, IF 28,803, code 30k from `personahub_code_v2_34999` plus 30k from `evol_codealpaca_heval_decontaminated`. Total 138,803. LoRA rank 128.
- **Run 13: tulu3_50k_if29k_code60k_r64.** Same as run 12 but LoRA rank 64.
- **Run 14: tulu3_50k_if29k_opencode50k_r64.** Math 50k, IF 28,803, and `nvidia/OpenCodeInstruct` 50k. Total 128,803.
- **Run 15: tulu3_50k_if29k_opencode50k_full_r64_ml512.** Same as run 14, but the 50k OpenCode examples are sampled randomly from the full 5M-row split.
- **Run 16: tulu3_30k_if29k_code35k_8b_r64.** Same data mix as run 11 (93,800 rows), but base model is Llama-3.1-8B. 8,355 steps; best checkpoint at step 5500 by validation loss (0.570).
- **Run 17: tulu3_30k_if29k_code35k_opencode15k_8b_r64.** Same as run 16 plus 10k random examples from the full `nvidia/OpenCodeInstruct` split. Total 103,800.

Post-SFT runs.

- **Run 18: rlvr_from_tulu3_targeted_30k_r64_v2.** Base: run 9. RL data: `allenai/RLVR-GSM-MATH-IF-Mixed-Constraints` train split, all subsets. 200 steps.
- **Run 19: stage2_opencode_full_from_rlvr.** Base: run 18. Stage-2 is SFT, not RL, on 10k random examples from the full `nvidia/OpenCodeInstruct` split, with markdown-fence normalization so targets are plain code.
- **Run 20: leetcode_from_rlvr_r64_final.** Base: run 18. Stage-2 SFT on `newfacade/LeetCodeDataset` [11] (2,641 rows, no difficulty filter), prompt-completion format.
- **Run 21: stage2_anchor12k_from_rlvr_r64.** Base: run 18. Stage-2 SFT on a 12k anchor mix: 6k `nvidia/OpenCodeInstruct` plus 2k each from the Tulu-3 math, IF, and code subsets. Lr $2e{-}5$.

- **Run 22: rlvr_from_tulu3_30k_if29k_code35k_8b_r64_v1.** Base: run 16 step-5500 checkpoint. RL data: `allenai/RLVR-GSM-MATH-IF-Mixed-Constraints`, all subsets. 200 steps; step 150 selected by full three-task evaluation.

- **Run 23: rlvr_stage2_code_from_8b_rlvr_step150_v1.** Base: run 22 step-150 checkpoint. RL data: `allenai/rlvr-code-data-python-r1-format-filtered`, filtered for rows with assertions and decontaminated against HumanEval entry points. Max rows 5k, 75 steps, lr $1e{-6}$, batch size 16, group size 8. Reward is the fraction of unit-test assertions passed in a sandbox.

- **Run 24: grpo_code_from_rlvr_mix_v2_8b.** Base: run 22 final checkpoint (`rlvr_from_tulu3_30k_if29k_code35k_8b_r64_v1_final`). RL data: `google-research-datasets/mbpp` [4], full config, filtered to 170 unique training problems and then deduplicated against HumanEval by signature, leaving 169 problems. GRPO, 150 steps, lr $1e{-5}$, batch size 8, group size 8. Reward is binary pass/fail from running MBPP unit tests. Best checkpoint at step 100: IFEval 78.2, GSM8K 78.5, HumanEval 58.5, Avg 71.7.

- **Run 25: ifeval_rlvr_from_grpo_code_from_rlvr_mix_v2_8b_step100.** Base: run 24 step-100 checkpoint. RL data: `allenai/RLVR-IFeval` train split, 14,973 rows. GRPO, 50 steps, lr $2e{-6}$, batch size 32, group size 8, temperature 1.0, max tokens 1024. Final checkpoint selected: IFEval 78.3, GSM8K 78.2, HumanEval 60.4, Avg 72.3.

B Contamination-checking algorithm

We describe the algorithm used to verify that no training example contains a prompt from the GSM8K, HumanEval, or IFEval test sets.

Input extraction. For each training source, we extract the user-facing prompt text. Tulu-3 and RLVR sources store prompts as lists of chat messages, so we take the content of every message whose role is "user." OpenCodeInstruct, LeetCode, MBPP, and other non-chat sources store prompts in text fields, which we take directly.

Text normalization. All comparisons operate on a normalized form: text is lowercased, punctuation is replaced with spaces, and runs of whitespace are collapsed to a single space. This makes comparisons insensitive to capitalization and punctuation.

Signals. We use three high-confidence signals and intentionally do not use fuzzy n-gram similarity. Several training sources share sentence templates, formatting constraints, function names, or standard definitions with the benchmarks; these similarities are not evidence that a benchmark test sample was used.

- **Exact prompt match.** The normalized extracted training prompt equals a normalized benchmark test prompt.

- **Exact field match.** Any string field anywhere in the training row equals a normalized benchmark test prompt.

- **Substring match.** A normalized benchmark test prompt appears inside the normalized full row text. This catches cases where a benchmark prompt is embedded in a longer instruction, response, or metadata field.

Search. The benchmark prompts are indexed by normalized text. For each training row, the scanner checks the extracted prompt, all string fields, and the concatenated full row text against GSM8K test, HumanEval test, and the IFEval evaluation prompt set.

Contamination scan results. Table 2 summarizes the full strict contamination scan for the datasets used by the final model. The scan used exact/substring matching only: exact prompt match, exact field match, or benchmark prompt substring in the full row text. All match counts were zero.

Table 2: Strict contamination scan against GSM8K test, HumanEval test, and IFEval prompts. Each cell reports exact-prompt / exact-field / substring matches.

Training source	**Rows scanned**	**GSM8K**	**HumanEval**	**IFEval**
Tulu-3 math subset	50,000	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Tulu-3 IF subset	28,803	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Tulu-3 code subset	34,997	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
RLVR-GSM-MATH-IF mixture	29,946	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
MBPP full config	964	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
MBPP sanitized config	420	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
RLVR-IFeval	14,973	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Total	**159,103**	**0 / 0 / 0**	**0 / 0 / 0**	**0 / 0 / 0**