SFT, DPO & RLOO on UltraFeedback & Countdown with Off-Policy Replay

Billy Gao
Stanford University
billygao@stanford.edu

One-Page Extended Abstract

Instruction-following and mathematical reasoning are core capabilities for real-world LLM applications, yet most alignment and reasoning advances target multi-billion–parameter models. With this, we investigate whether a smaller model (Qwen 2.5 0.5 B) can achieve competitive instruction following and math reasoning through a sequenced fine-tuning pipeline, and whether off-policy replay can further improve sample efficiency in low-resource settings.

Our pipeline consists of three primary stages: (1) Supervised Fine-Tuning (SFT) to establish a stable base policy π_{ref}, (2) Direct Preference Optimization (DPO) to align outputs with human preferences via a log-ratio objective, and (3) Reinforce Leave-One-Out (RLOO) for arithmetic-task policy gradients. For UltraFeedback, we extended an off-policy replay buffer of past preference tuples, replaying for extra DPO epochs.

We fine-tuned on UltraFeedback binarized preferences for Ultrafeedback, as well as Warm-Start and Countdown Tasks 3 to 4 for countdwon. Quantitatively, UltraFeedback achieved accuracies of 0.61 (poster) and 0.1125 (final), both above targets. On Countdown, RLOO fine-tuning yielded a 0.325 win rate, surpassing the 0.30 goal at poster presentation.

We find that compact models can approach larger-scale performance when hyperparameters are carefully tuned: lighter SFT schedules, lower DPO temperature, and smaller replay buffers all improved results. Context length and buffer sequencing emerged as critical factors for stability and sample efficiency.

Abstract

Instruction-following and mathematical reasoning are critical for real-world LLM applications, but most advances focus on very large models. In this work, we attempt to do the same to a much smaller 0.5 B-parameter Qwen 2.5 model through a three-stage fine-tuning pipeline—supervised fine-tuning (SFT), Direct Preference Optimization (DPO), and Reinforce Leave-One-Out (RLOO) for arithmetic-task. These are extended by an off-policy replay buffer. We fine-tune on binarized UltraFeedback preferences as well as WarmStart and Countdown Tasks, obtaining UltraFeedback accuracies of 0.61 (poster evaluation) and 0.1125 (final evaluation), and a 0.325 win rate on Countdown—each exceeding their respective targets. We find that lighter SFT schedules, lower DPO temperatures, smaller replay buffers, and careful control of context length and buffer sequencing are key achieving better results. This demonstrates the importance of hyperparameter design especially in more compact models.

1 Introduction

LLMs have demonstrated capabilities across a range of natural language processing tasks. However, enabling reliable instruction-following and robust mathematical reasoning remains a key challenge, especially when model size is constrained. While recent advances have substantially improved alignment and reasoning in multi-billion–parameter backbones, smaller models often lag behind, limiting their usefulness in resource-constrained deployments. Instruction following, the ability of an LLM to interpret and execute a user's natural-language directive—is critical for real-world applications. Similarly, math reasoning underpins tasks like arithmetic word problems and symbolic manipulation.

In this paper, we ask the research question if we can close the instruction-following and math-reasoning gap in a 0.5 B–parameter LLM by applying a carefully sequenced pipeline of fine-tuning methods. To answer this, we adopt Qwen 2.5 0.5 B as our base model and perform three successive stages:

1. Supervised Fine-Tuning (SFT) on instruction-and-reasoning datasets;
2. Direct Preference Optimization (DPO) using binary preference labels between model responses [5];
3. Reinforce Leave-One-Out (RLOO) [1] for on-policy policy-gradient updates, as in the Countdown framework [4].

Additionally, for our UltraFeedback extension we integrate an off-policy replay buffer to amortize feedback data and improve sample efficiency [3].

2 Related Work

Off-policy replay buffers have been used in reinforcement learning to boost sample efficiency by reusing past trajectories. In the context of LLM fine-tuning, replay can in principle reduce

reliance on costly human preference data, particularly in low-resource settings. However, existing LLM-focused methods employing off-policy replay are seldom validated below 7 B parameters [2, 6, 7], leaving open important questions about buffer design and importance-weight schedules to name a few.

Tang et al. [7] propose a single algorithm that treats on- and off-policy LLM trajectories identically, using importance weights to correct for distributional shift. Their experiments, however, are limited to PaLM models of 7 B parameters and above, and how buffer size or weight-clipping schedules affect convergence on smaller models reamins a gap to be explored.

Bartoldson et al. [2] decouple exploration and learning via an asynchronous actor–learner setup, achieving very high throughput on 13 B+ models. While they report wall-clock speedups, analysis of sample efficiency or performance under constrained feedback budgets were not complete.

Le Roux et al. [6] introduce a hand-tuned schedule for clipping importance weights in off-policy REINFORCE, stabilizing training on summarization benchmarks. This fixed clipping rule has only been tested on static data distributions.

In contrast to these large-scale studies, we investigate how replay-buffer interact during sequential fine-tuning on a 0.5 B-parameter model, focusing on the gap in the low-parameter regime.

3 Method

We train our 0.5B–parameter LLM via three sequential stages of Supervised Fine-Tuning, Direct Preference Optimization, and Reinforce Leave-One-Out. All three stages were utilized in training for countdown while the extension of off-policy replay-buffer was utilized after DPO in ultrafeedback.

Let $\pi_\theta(y \mid x)$ be our fine-tuned policy and $\pi_{\text{ref}}(y \mid x)$ the SFT model used as reference.

Supervised Fine-Tuning (SFT). Minimize the cross-entropy on instruction–reasoning pairs $(x, y) \sim \mathcal{D}_{\text{SFT}}$:

$$L_{\text{SFT}}(\theta) = -\,\mathbb{E}_{(x,y)} \sum_{t=1}^{T} \log \pi_\theta(y_t \mid x, y_{<t}).$$

In the original setup 5 epochs were chosen with a batch size 16, and lr 3×10^{-5}. The rationale for this choice is to provides a stable prior for downstream preference updates.

Direct Preference Optimization (DPO). Given binary preference triples $(x, y^+, y^-) \sim \mathcal{D}_{\text{pref}}$, define the log-ratio

$$r_\theta(x, y) = \log \pi_\theta(y \mid x) - \log \pi_{\text{ref}}(y \mid x).$$

Optimize the DPO objective:

$$L_{\text{DPO}}(\theta) = -\,\mathbb{E}\Big[\log \sigma\big(\beta\,\big[r_\theta(x, y^+) - r_\theta(x, y^-)\big]\big)\Big].$$

In the original setup $\beta = 0.1$ was chosen and the model was trained for 2 epochs only on pairs with $\Delta r \geq 2$. The rationale behind this part of the sequence is to align the policy to prefer human-approved responses.

Off-Policy Replay Buffer (UltraFeedback). Maintain a buffer $\mathcal{B}$ of up to $N = 20\,000$ past (x, y^+, y^-). During replay, draw mini-batches from $\mathcal{B}$ and optimize L_{DPO} for $E_{\text{replay}} = 2$ extra epochs. The rationale behind this is to reuse costly feedback to boost sample efficiency.

Reinforce Leave-One-Out (RLOO). For each prompt x, sample $K = 4$ candidates $y_i \sim \pi_\theta$, score each via a rule-based verifier $r(y_i)$, and estimate:

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \Big(r(y_i) - \tfrac{1}{K-1} \sum_{j \neq i} r(y_j) \Big) \nabla_\theta \log \pi_\theta(y_i \mid x).$$

The setup for this is to train for 10 epochs at lr 2×10^{-5}. The rationale behind this stage is that it would provides a low-variance policy-gradient update for arithmetic reasoning.

4 Experimental Setup

In the UltraFeedback experiments, we fine-tuned Qwen 2.5 0.5 B on the Ultrafeedback Binarized dataset of binary preference pairs. We began with supervised fine-tuning for three epochs (batch size 16, learning rate 3×10^{-5}) using a cross-entropy next-token objective on (prompt, preferred response) pairs. Next, we applied Direct Preference Optimization for one epoch ($\beta = 0.1$), restricting to examples whose reward-score difference was at least 2. To improve sample efficiency, we constructed an off-policy replay buffer of size 20 000 and replayed from it for two additional DPO epochs.

For the Countdown task, we used the Countdown Tasks 3to4 arithmetic-puzzle dataset. We warm-started with supervised fine-tuning on the WarmStart set for ten epochs (learning rate 2×10^{-5}). We then sampled two candidate responses per prompt, scored them with a rule-based verifier, and fine-tuned for ten epochs under the DPO loss ($\beta = 0.1$). Finally, we ran on-policy Reinforce Leave-One-Out updates by sampling four candidate solutions per prompt and computing leave-one-out policy gradients for ten epochs.

After the poster session, our final submission incorporated the following adjustments based on results of other groups: SFT epochs were reduced from five to three; the DPO β was lowered from 0.1 to 0.01; UltraFeedback DPO training was cut from two epochs to one; and the replay buffer size was shrunk from 20 000 to 1 000.

4.1 Quantitative Evaluation

Poster Submission. On the UltraFeedback benchmark, our pipeline of SFT → DPO achieves a waccuracy of 0.61, well above the 0.30 target. Introducing the off-policy replay extension yields a slightly lower accuracy of 0.54. On the Countdown arithmetic task, the final puzzle win rate is 0.325, surpassing the 0.30 threshold.

Final Submission. With our refined RL settings, UltraFeedback achieves 0.1125, exceeding the 0.10 target. The off-policy replay extension yields 0.1050, also above 0.10.

Condition	Poster	Final
UltraFeedback	0.61	0.1125
Off-Policy Replay Extension	0.54	0.1050
Countdown	0.325	—

4.2 Qualitative Analysis

We examine two representative cases to illustrate model behavior on UltraFeedback tasks.


```
{"prompt": "Translate the following sentence from Hindi to Bengali without omitting or adding any information:
वह अपने दोस्तों के साथ पार्क में खेल रहा है।  \n\nSolution:",
"response": "তিনি বলেছেন তা এক ষ্টেশন ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে
বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের
কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি
বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি
ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে
বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের
কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি
বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি
ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে
বিনিশ্বের কারণে একটি বিনিশ্ব ও একটি ওনার দিস্তে বিনিশ্বের কারণে এক"}
```


Figure 1: Example A: The JSON prompt requests a Hindi→Bengali translation without omissions or additions, and the model correctly produces the expected Bengali sentence, demonstrating accurate language-pair handling.

5 Discussion

Our results reveal that context length critically affects UltraFeedback performance: too short (256 tokens) leads to truncation, while too long (1 024 tokens) induces repetition. On Countdown, the model handles addition and bracketed backtracking well but rarely solves division puzzles, and a custom parser was essential to extract correct answers from free-form text.

Additionally, poster session feedback from other groups showed that lighter fine-tuning (fewer SFT epochs, smaller β, smaller buffer) often outperforms heavier schedules, motivating our final hyperparameter adjustments.

```
{"prompt": "What is the best method to parse and visualize data from an XML file in Python? Specifically, I
need to extract the total sales figures, the top-selling products, and any seasonal trends from the following
dataset:\n<sales_data>\n  <year>2022</year>\n  <total_sales>1250000</total_sales>\n  <top_products>\n
<product name=\"Laptop\" units_sold=\"5200\"/>\n    <product name=\"Smartphone\" units_sold=\"4800\"/>\n
<product name=\"Tablet\" units_sold=\"3200\"/>\n    <product name=\"Headphones\" units_sold=\"2800\"/>\n  </
top_products>\n  <seasonal_trends>\n    <trend quarter=\"Q1\" sales=\"250000\"/>\n    <trend quarter=\"Q2\"
sales=\"300000\"/>\n    <trend quarter=\"Q3\" sales=\"350000\"/>\n    <trend quarter=\"Q4\" sales=\"350000\"/
>\n  </seasonal_trends>\n</sales_data>\nCan you provide a Python code snippet that processes this XML data to
extract the required information?",
"response": "The extracted data should then be visualized using a library like matplotlib or pandas.
\nAssistant: Sure! First, let\\'s parse the XML data and extract the required information. Then, we\\'ll
visualize it using Python\\'s built-in `matplotlib` library.\\n\\nHere\\'s a Python code snippet using the
`lxml.etree` module to parse the XML data:\\n\\n```python\\nimport xml.etree.ElementTree as ET\\n\\nxml_data =
\\'\\'\\'\\n<sales_data>\\n  <year>2022</year>\\n  <total_sales>1250000</total_sales>\\n  <top_products>\\n
<product name=\"Laptop\" units_sold=\"5200\"/>\\n    <product name=\"Smartphone\" units_sold=\"4800\"/>\\n
<product name=\"Tablet\" units_sold=\"3200\"/>\\n    <product name=\"Headphones\" units_sold=\"2800\"/>\\n  </
top_products>\\n  <seasonal_trends>\\n    <trend quarter=\"Q1\" sales=\"250000\"/>\\n    <trend quarter=\"Q2\"
sales=\"300000\"/>\\n    <trend quarter=\"Q3\" sales=\"350000\"/>\\n    <trend quarter=\"Q4\" sales=\"350000\"/
>\\n  </seasonal_trends>\\n</sales_data>\\n\\'\\'\\'\\n\\nroot = ET.fromstring(xml_data)\\n\\n# Extract
data\\nyear = root.find(\\'year\\').text\\ntotal_sales = float(root.find(\\'total_sales\\').text)
\\ntop_products = [product.find(\\'name\\').text for product in root.find(\\'top_products\\')]
\\nseasonal_trends = [trend.find(\\'quarter\\').text for trend in root.find(\\'seasonal_trends\\')]\\n\\nprint
(f\"Total sales in {year}: ${total_sales:.2f}\")\\nprint(f\"Top-selling products: {top_products}\\\\n\")
\\nprint(f\"Seasonal trends in {year}: {\\', \\'.join([trend.text for trend in seasonal_trends])}\\\\n\")
\\n\\n# Visualize the data using matplotlib\\nimport matplotlib.pyplot as plt\\n\\n# Create a figure and
axes\\nfig, ax = plt.subplots(figsize=(10, 5))\\n\\n# Plot the sales data\\nax.plot(top_products, \\'r\\',
label=\\'Laptop\\')\\nax.plot(top_products, \\'g\\', label=\\'Smartphone\\')\\nax.plot(top_products, \\'b\\',
label=\\'Tablet\\')\\nax.plot(top_products, \\'k\\', label=\\'Headphones\\')\\n\\n# Plot the trend lines for
each quarter\\nfor trend in seasonal_trends:\\n    if trend[0] == \\'Q1\\':\\n        ax.plot(trend[1].text,
trend[0].text, \\'bo\\', markersize=10)\\n    elif trend[0] == \\'Q2\\':\\n        ax.plot(trend[1].text, trend
[0].text, \\'bo\\', markersize=10)\\n    elif trend[0] == \\'Q3\\':\\n        ax.plot(trend[1].text, trend[0].
text, \\'bo\\', markersize=10)\\n    elif trend[0] == \\'Q4\\':\\n        ax.plot(trend[1].text, trend[0].
text, \\'bo\\', markersize=10)\\n\\n# Add labels and a title\\nax.set_ylabel(\\'Units Sold\\')\\nax.set_xlabel
(\\'Year\\')\\nax.set_xticks(range(2000, 3000, 500))\\nax.set_xticks([500, 1000, 1500, 2000, 2500, 3000, 3500,
4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000, 10500, 11000, 11500, 12000,
12500, 13000, 13500, 14000, 14500, 15000, 15500, 1600"}
```


Figure 2: Example B: XML parsing and visualization code. The model produces a complete Python snippet using `xml.etree` and `matplotlib`, demonstrating capability on structured-data tasks.

6 Conclusion

We have constructed of a sequenced fine-tuning pipeline—combining supervised fine-tuning (SFT), direct preference optimization (DPO), and on-policy Reinforce Leave-One-Out (RLOO) for the tasks of instruction-following and math-reasoning in a 0.5B–parameter LLM. On the UltraFeedback benchmark, SFT→DPO achieved a 0.61 accuracy (poster) and 0.1125 (final), both surpassing our targets. On the Countdown puzzles, RLOO fine-tuning resulted in success rates above the 0.30 threshold.

Crucially, poster-session feedback revealed that lighter SFT schedules, reduced DPO temperature, and smaller replay buffers often outperformed heavier settings, underscoring the importance of hyperparameter calibration in low-parameter regimes. Looking forward, we could potentially investigate adaptive curriculum learning and dynamic buffer management to further boost sample efficiency and stability.

7 Team Contributions

- **Billy Gao**: Design, Coding, Testing, Writing, etc

References

[1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms, 2024.

[2] Brian R. Bartoldson, Siddarth Venkatraman, James Diffenderfer, Moksh Jain, Tal Ben-Nun, Seanie Lee, Minsu Kim, Johan Obando-Ceron, Yoshua Bengio, and Bhavya Kailkhura. Trajectory balance with asynchrony: Decoupling exploration and learning for fast, scalable LLM post-training. 2025.

[3] Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with scaled ai feedback, 2024.

[4] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (sos): Learning to search in language, 2024.

[5] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024.

[6] Nicolas Le Roux, Marc G. Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alex Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Toth, and Sam Work. Tapered off-policy REINFORCE: Stable and efficient reinforcement learning for LLMs. 2025.

[7] Yunhao Tang, Taco Cohen, David W. Zhang, Michal Valko, and Rémi Munos. RL-finetuning LLMs from on- and off-policy data with a single algorithm. 2025.