William Chang Liu - 1009048852

Ryan Zhang - 1009020453

Taiyi Jin - 1009075796

Abdus Shaikh - 1007288533

CSC490 Assignment 4: Creating a RL LLM as a judge

Part 1: Literature Review

Paper	Commentary
Zhang, et al. "Med-rlvr: Emerging medical reasoning from a 3b base model via reinforcement learning." https://doi.org/10.48550/arXiv.2502.19655	This paper presents Med-RLVR, a model created by fine-tuning QWEN 2.5-3B using RLVR to elicit medical reasoning by using multiple choice questions as verifiable rewards. The authors use a special reward function that encourages the model to follow a chain-of-thought-like reasoning format to answer questions, more specifically, the function rewards the model if it follows the reasoning format while also producing the correct answer. This paper successfully shows that RLVR can be used to fine-tune models to reason about other domains beyond coding and math. This model can potentially be used as an LLM judge in our project as it is designed to specifically reason about medical data. For example, given a patient's EHR, we could ask this model to evaluate whether the the treatments this patient received is consistent with the diseases/conditions they have.
Su, et al. “Crossing the Reward Bridge: Expanding RL with Verifiable Rewards Across Diverse Domains” https://doi.org/10.48550/arXiv.2503.23829	This paper extends Reinforcement Learning with Verifiable Rewards (RLVR) beyond structured tasks like math and code to a broad set of reasoning domains. The authors introduce a soft, model-based reward system where a smaller generative verifier (7B) is trained by distillation from a larger model (72B) to judge correctness

	probabilistically instead of with binary labels. This enables reinforcement learning to handle open ended, multi domain reasoning such as medicine, psychology, and economics. The model outperforms rule based RL baselines and shows that verifiable rewards can be generalized beyond strictly objective domains. This approach is highly relevant to our synthetic EHR data project, as it provides a framework for training a verifier model that can automatically assess whether generated diagnoses or treatments are plausible and medically consistent. It could serve as a reward model to guide data generation or to validate realism in synthetic patient records.
Das, et al. “SynRL: Aligning Synthetic Clinical Trial Data with Human-preferred Clinical Endpoints Using Reinforcement Learning” https://doi.org/10.48550/arXiv.2411.07317	This paper presents SynRL, a framework for fine tuning synthetic clinical trial data generators using reinforcement learning to align generated data with human preferred clinical endpoints. The method uses a data value critic/reward function that scores synthetic records based on utility for downstream prediction tasks while maintaining fidelity to real data. The generator is then updated via RL to maximize this reward. Experiments on four clinical trial datasets show that SynRL improves utility, preserves fidelity, and enhances privacy compared to baseline generators. This method is relevant to our project because it demonstrates how RL based fine tuning can optimize models for complex structured data like EHRs, similar to how RLVR fine tunes LLMs for reasoning tasks. Essentially, SynRL is a tabular data analogue of RLVR, using a reward function to guide outputs toward human specified goals.
Zhi, et al. “Robustness Verification of Deep Reinforcement Learning Based Control Systems using Reward Martingales” https://doi.org/10.48550/arXiv.2312.09695	This paper introduces a way to formally check reinforcement learning systems using reward martingales. These are mathematical tools that limit how much the total reward can change if the inputs are slightly altered. This gives guarantees

	that the RL agent's performance won't suddenly drop under small changes or manipulations. The authors test this on continuous control tasks and show that it can catch and prevent reward-hacking behaviors. This is directly relevant to RLVR because in a synthetic EHR pipeline, both the generator and verifier can be vulnerable to "gaming" the reward. For example, generating data that looks correct but is clinically meaningless. The paper provides a theoretical basis for the verifiability and safety module in RLVR.
Liu, et al. "Fleming-R1: Toward Expert-Level Medical Reasoning via Reinforcement Learning" https://doi.org/10.48550/arXiv.2509.15279	This paper proposes a medical reasoning framework that combines RLVR. It introduces a reasoning oriented data strategy and a two stage RL process that focuses on improving the model's ability to reason correctly instead of just guessing the right answer. The model is trained to produce step by step explanations that can be verified, which helps reduce reward hacking and improve reliability in clinical reasoning. The model is evaluated on benchmarks like MedQA, PubMedQA, and CareQA, showing strong gains over models like Med-PaLM 2 and GPT-4 in both accuracy and reasoning transparency. This is relevant to RLVR because it shows how verifiable reward signals can make medical LLMs more trustworthy and interpretable. For our project, it provides a good example of how RLVR can be used to align model reasoning with real world correctness, especially in high risk areas like healthcare.

Part 2: Metrics and Evaluation

Metric	What it measures	Why we care	How we measure it	Challenges at scale
Dimension-wise distribution	How closely each feature in synthetic data matches the real data in prevalence and distribution	We need our generated synthetic data to match the actual statistical patterns of the data without copying them. This ensures reliability in downstream use.	Quantitative: Compare feature level statistics, such as mean, variance, and zero rate, between real and synthetic datasets. Compute correlation matrix distance for feature correlations. Qualitative: Visual inspection using scatter plots or histograms, inspect prevalence of features, distribution shapes, and patterns in key tables. Look for major shifts, missing rare codes, or implausible combinations.	High dimensional EHR data may make visual inspection difficult. Comparing every feature manually is not feasible at scale.
Attribute & Membership Inference Risk	Risk of privacy leakage, whether synthetic data reveals sensitive real patient information	EHR data contains sensitive patient information, so we need to make sure that no real patient data is reconstructed.	Quantitative: Measure attribute and membership inference risk by testing if synthetic records reveal real patients' data. Qualitative: Inspect any outlier records	Privacy evaluation may be limited by dataset size. Also we might need to look into realistic attack models, which can be hard.

			that may be too close to real patients.	
Clinical Validity (Dimension-wise prediction)	Whether synthetic data follows clinical logic and rules	The generated data has to follow clinical logic, for example, the diagnoses, lab results, and medication should align for each patient.	Quantitative: Count violations of simple clinical rules (e.g., patient with no diagnosis receiving medication). Qualitative: Manually inspect selected patient records for logical consistency between tables.	Defining clinical rules can be complicated, some rare but valid scenarios may be flagged incorrectly.
Downstream Utility	How well synthetic data can actually be used in real applications, such as training a model to predict diseases	We aim to make our synthetic data valuable for actual medical model training. This was part of our goal for the project.	Quantitative: Train the same model on synthetic and real EHR, evaluate and compare their performance (e.g., accuracy of the model making a correct prediction for a specific disease). Qualitative: Inspect misclassified patients to see which patterns the model struggles to learn.	Comparing model performance across datasets can be sensitive to the choice of predictive task, as some diseases may be easier to predict.
Non-zero Code Columns (NZC)	Sparsity of synthetic data, number of non zero features	We want to avoid unrealistic dense or empty records, so we care about how many 0s are generated.	Quantitative: Count the proportion of zeros per feature or table and compare with real data. Qualitative: Visual inspection	Rare codes may still be missing, and high dimensional data makes visual inspection tedious.

			using bar plots or heatmaps of zeros across tables, and identify features that are too dense or too sparse.	

Part 3: Establishing a baseline with an LLM

Code Reference: baseline_llm_modal.py

Section 1: Metric and Task

We want to evaluate the realism of synthetic patient records (EHRs) compared to a real dataset.

Metric: We call it “Realism score”, which is Dimension-wise distribution together with Attribute & Membership Inference Risk. Specifically, we use a combined realism score with weights:

Mahalanobis distance (85%), which measures how far a synthetic record deviates from real data distribution.

k-Nearest Neighbors distance (15%), which measures local similarity to real samples.

This metric is statistically defined as how far the vector is from the real distribution, closer the better, and how far from k nearest neighbours, the further the better to avoid privacy risks.

The final score is mapped to 1–10, where 10 means highly realistic, 1 means unrealistic.

Section 2: Baseline LLM chosen

TableLLM-8b model, which is a robust large language model (LLM) with 8 billion parameters, purpose-built for proficiently handling tabular data manipulation tasks.

Link to TableLLM: https://tablellm.github.io/

Section 3: Dataset

We used the MIMIC-IV demo dataset (normalized and preprocessed from our previous A2 code)

Features: 3333 columns including gender, age, prescriptions, diagnoses, procedures, admissions, hospital events, pharmacy records, etc.

Splits: Evaluation conducted on first 100 synthetic samples to compare with baseline metric. Real data used to compute Mahalanobis and kNN scores.

We also used a synthetic dataset generated from this demo dataset with 100 synthetic samples for evaluation.

Section 4: Classifiers

Simple Baseline Classifier (Statistical Metric "Realism Score")

Type: Mahalanobis + kNN combined realism score

Input: Numeric patient vectors (entire EHR)

Output: **ONE SINGLE INTEGER SCORE 1-10**

Comment: It provides a purely statistical measure of realism

TableLLM Classifier

Model: TableLLM-8b (no fine-tuning, as it is not required for this part)

Input: Tabular representation of EHR, converted from vectors using the *vector_to_table* function

Output: Model prompted to output **ONE SINGLE INTEGER 1–10** reflecting realism

Example of LLM classifier input:

You are a clinical data auditor.

Given the synthetic patient record below, rate how realistic it is.

Return ONLY a single integer from 1 to 10.

SubjectID,Gender,Age,IsDead,Prescriptions_mean,Prescriptions_sum,Prescriptions_max,Prescriptions_min,Prescriptions_std

101,1,65,0,0.23,4,1,0,0.44

Rating (1-10):

(Note: only prescriptions are displayed here for simplicity, but the actual input contained information from more tables)

Section 5: Results

Realism Score Metric	Mean	Std	Min	Max
Baseline	7.23	1.55	3	10
TableLLM	6.90	1.75	2	10

Correlation Between Evaluators

Pearson correlation: 0.72

Spearman correlation: 0.70

The scores from TableLLM generally match up with the statistical realism metric, which suggests it can be a useful complement when evaluating synthetic patient records.

Section 6: Quantitative Observation and Comparison

- The baseline metric is reliable, easy to interpret, and good at detecting overall deviations from the real data, but it can miss rare yet valid clinical patterns.
- TableLLM adds a qualitative perspective, capturing unusual or subtle patterns that the baseline might overlook, though it can sometimes misjudge realism without being fine tuned.
- Combining the idea from both approaches and potentially fine tuning the LLM, would create a more robust and comprehensive way to evaluate realism.

Section 7: Limitations

- TableLLM can misinterpret rare combinations of features.
- Small changes in wording can affect the scores.
- We only looked at 100 synthetic samples, so the results might not generalize.

Section 8: Next Steps

- Fine-tune TableLLM on labeled real vs. synthetic samples to make its scores more consistent with statistical realism and clinical plausibility.
- Evaluate a larger set of synthetic samples to get more reliable results.
- Bring in expert clinical review for extreme or unusual cases to ensure the evaluations make sense in practice.

Section 9: Qualitative Analysis of Common Mistakes

Here are Top 5 largest disagreements between baseline and LLM scores:

Sample	Baseline	TableLLM	Difference	Comment
12	9	5	4	Rare diagnosis underweighted by LLM
37	4	8	4	Generic patient overestimated by LLM

44	7	3	4	LLM penalizes extreme feature values
51	10	7	3	Rare event patterns underestimated
88	5	9	4	LLM misinterprets missing values

Summary of patterns:

- The LLM often gives higher realism scores for very common or generic patterns.
- It can give lower scores for rare but perfectly valid records.
- The baseline metric is reliable for statistical differences but misses the subtle clinical context that makes some records realistic.
-

Section 10: Additional Notes

1. Figuring out whether a synthetic patient record is truly "realistic" is tough, thus there's no way for us to actually label the synthetic data as "good" or "bad". We can estimate realism using a formula, like the Mahalanobis + kNN score, but even that isn't perfect. Because of this uncertainty, we treat the realism metric as a second "classifier" rather than a ground-truth label. It guides the model without pretending to be 100% correct.

2. In traditional prediction tasks (like predicting mortality of a patient from other columns), we can train a logistic regression or other straightforward model on labeled outcomes. Here, we don't have true labels, only estimated realism scores. That makes a simple baseline like logistic regression inapplicable, since it relies on knowing ground-truth targets. Instead, we rely on our realism metric and the LLM to provide guidance.

3. Even though we don't have traditional labels, we still divide our synthetic dataset into training, validation, and test sets. During training, the model only sees the training data. When we evaluate it, we use new synthetic samples that weren't seen during training, so we get a fair sense of how well the model generalizes to unseen data. **However, this all happens in Part 4, so we don't have a train-test split here in Part 3, because it makes no sense to either split the real EHR data (we need all real EHR to form the distribution for calculating Mahalanobis distance) or the generated synthetic EHR data (we are not training anything using synthetic data, so we use all of them for evaluation).**

Part 4: Training your Judge

The metric we are measuring and our hypotheses on training success

For this assignment we are focusing on measuring and judging the realism of the synthetic electronic health records (EHRs). We are looking to judge this metric on an integer scale from 1-10. For the purpose of this assignment, realism, which is a difficult concept to completely grasp, will be defined as how statistically similar the EHR is to the distribution of real EHR vectors, yet at the same time remaining far from k-nearest neighbours in order to avoid privacy risks. This definition remains the same as the one used in part 3 of this assignment.

Mathematically, we will be defining realism as a combination of the following:

- **Mahalanobis distance** between synthetic and real feature distributions (captures global statistical alignment), and
- **k-Nearest Neighbor (kNN) distance** to real samples (captures local manifold similarity and guards against mode collapse).

We will be finetuning an LLM using RLVR to act as our judge/evaluator. Our hypothesis on training success is that it may be difficult for the judge model to learn how to evaluate a frequency vector of length 3333 without any context on the meaning of the numeric values.

However, should training be successful, then:

1. The evaluator will be able to better identify high-quality synthetic EHRs.
2. The evaluator will be able to generalize beyond the limitations of the equation used as verifiable rewards (more on this below).
3. The evaluator can be used as a **training-time reward model** to guide future generative EHR models toward realism *without* relying on the expensive statistical scoring pipeline.
4. The evaluator can be used to determine whether a model checkpoint is “good enough” to be stored in the model registry. The evaluator therefore acts as a **gatekeeping constraint** ensuring only sufficiently realistic models are retained for future training iterations.

Still, due to the potential instability in rating generation due to the low context of the prompts which are fed into the LLM, we hypothesize that **care must be taken in reward shaping** and output parsing to prevent the model from collapsing toward degenerate behaviors (e.g., always predicting the mode score, or predicting a number which is not a score entirely).

Our training pipeline

Our RLVR judge is trained through the following sequence of steps

1. Training data generation/retrieval
    a. To train our model we must have the training data. Since this project is tackling a synthetic data generation problem, we have the option to generate our training data for our judge on the fly. More specifically we can choose to generate a new synthetic dataset when training our judge. Otherwise we can choose to use a pre-existing generation, in which case we simply need to retrieve the data from the S3 bucket where it is hosted
2. Model initialization
    a. Initialize and load the LLM and tokenizer that will act as our judge using the hugging face hub. The specific model and tokenizer we choose to load is configurable, at this moment, we have chosen the Qwen2.5-1.5B-Instruct model
3. Main loop
    a. Each epoch of our model training completes the following steps
        i. Generate realism scores: Generate the verified rewards for each of synthetic samples using the mathematical constructs as defined above
        ii. Generate judges scores: Prompt the LLM to consume our synthetic data and evaluate its realism on a scale of 1-10. Importantly, we use the Hugging Face LogitsProcessor module to constrain the model outputs to only be integers in the range 1-10. Without this constrained decoding we fall victim to useless outputs and an entirely useless training loop
        iii. Compute loss: After collecting the judges evaluations for each sample we compare them to our verified reward. We use a combination of euclidean distance and KL divergence in our loss function guide our LLM towards a more accurate understanding of realism in the context of our data.
        iv. Log and Backpropagate: Log our loss values for debugging and monitoring purposes and backpropagate the loss.

The base model we are finetuning

In Part 3 of the project, we used TableLLM-8B, a large language model specifically designed for table reasoning. This model demonstrated strong reliability in consistently producing clean, single-integer realism scores (1–10) when provided with the EHR summary tables. Because it aligned well with the evaluation protocol, our initial plan was to fine-tune TableLLM using RLVR to create a refined realism classifier.

However when we started RLVR training, several significant issues emerged. One of the most significant and the main reason we pivoted was because TableLLM-8B's 8B parameters require roughly 16–24GB just to load (FP16). Since RLVR training requires two models (one used frozen as reference, and the other used as the policy to update), total memory had exceeded feasible to run on Modal's cloud computing, which we pivoted to as our provider. Choosing to run the frozen reference model on CPU slowed the training down to a point where it was too expensive and slow to run at all.

Due to this, we decided to switch our baseline LLM model to Qwen/Qwen2.5-1.5B-Instruct, a conversational model which could still ingest large tables and vectors. However we found new challenges with Qwen's inconsistent outputs:

- The model does not intrinsically map the task to the constraint 1–10. Despite being prompted to strictly follow that rule, Qwen would often output values such as: "35", "6,", "8!", "07" which would cause our fallback value of 5 to trigger.
- Qwen would commonly try to output justification text, due to its prelearned conversational bias.

Our solution to these problems was to restrict its output using constrained decoding, to limit the model to *only produce tokens that correspond to the valid rating space*. After this, Qwen was able to consistently respond with only an integer rating from 1-10, and was ready to be trained in our RLVR pipeline.

The RL algorithm we are using and why

Reinforcement Learning with Verifiable Rewards (RLVR)

For this task, we use **Reinforcement Learning with Verifiable Rewards (RLVR)**. RLVR is a family of RL methods designed for **alignment tasks where the reward can be computed deterministically from data**, rather than using human preference labels. This differs from traditional RLHF, where feedback comes from human rankings.

In our case, the reward signal is derived from the **baseline realism metric** computed using (more on this below):

- Mahalanobis distance between real and synthetic patient vectors
- kNN distance to nearest real patients in feature space

This produces a **verifiable realism score** in the range **1–10**, which is used as the target reward for RL training.

Action Policy Model

We fine-tune Qwen/Qwen2.5-1.5B-Instruct as our scoring model. During training, the LLM is given:

```
"You are a clinical data auditor… Return ONLY a single integer from 1 to 10."
```

And it produces a single digit token.

To prevent the LLM from generating stray text (e.g., “The score is 6”), we restrict the model to **only generate tokens in the range 0–9** using a custom logits processor.

Verifiable Reward Design

A central requirement of RLVR is that the reward signal must be objective, repeatable, and independent of subjective human annotation. In our case, we leverage a numeric realism score constructed from two statistical distribution-fit metrics computed over real vs synthetic EHR feature vectors:

- Mahalanobis distance (measures similarity to real manifold)
- kNN distance (measures local support / non-collapse)

Together after some number massaging and weighing, these together produce an integer score in the range 1–10, which we treat as the *ground-truth target* for the evaluator model.

Because this metric is deterministic and strictly data-driven, it meets the definition of a verifiable reward:

> If the same synthetic vector is scored twice, the reward will be identical, regardless of who evaluates it or when.

This provides a stable RL learning signal, unlike RLHF, where reward is noisy and human-dependent.

Initially for our reward formulation, we decided to subtract the *batch mean* to compute advantage: However this exposed an undesirable effect in our setting:

- A batch contains several *good* synthetic records
    - Their rewards are pushed downward, not because they are inaccurate, but simply because they were grouped together
- A batch contains mostly *bad* records
    - A *slightly less bad* record gets a positive advantage signal, unintentionally rewarded

This results in reward relativity, not correctness. In early training runs, we observed exactly this behavior: the evaluator began collapsing toward middle scores, because the model was rewarded for *matching the batch mean*, not for *matching the true realism metric*.

To avoid this failure mode, we use an absolute-distance reward based directly on score agreement. By human decision, and as now a hyperparameter, we decided to score the Mahalanobis distance and kNN distance out of 10 based on fixed values which represent a score of 10 and a score of 1.

Reward Function

The reward encourages the model score to align with the baseline (verifiable equation) score:

- $reward = 1 - (abs(llm\ score - baseline\ score)/9)^2$
    - 1.0 when predictions match perfectly
    - Decreases smoothly as disagreement increases
    - Squared term penalizes larger deviations more strongly
    -

Policy Gradient Update with KL Regularization

We train using a **KL-regularized REINFORCE objective**:

$$\mathcal{L} = -\mathbb{E}[\text{advantage} \cdot \log P_{\text{policy}}] + \beta \cdot KL(P_{\text{policy}} || P_{\text{ref}})$$

- policy = trainable model
- ref = **frozen** copy of the same model
- β = 0.01 (controls drift)

Training loop summary:

1. Convert synthetic vectors into tabular records
2. Policy model predicts a realism score (1–10) via constrained decoding
3. Compute baseline realism score
4. Compute reward based on agreement
5. Apply KL-regularized policy gradient update

Logging and metrics of the training runs

To understand whether the RLVR training loop was behaving as intended, we incorporated *explicit, step-wise logging* inside the train_epoch() and evaluate() functions. Since our current training dataset is small due to issues with our provider for this timeframe of this assignment, we prioritized transparent, interpretable prints over external dashboards (e.g., wandb), although the same values could easily be streamed there for future consideration.

Our logging served three core purposes:

1. Verify that the LLM is outputting valid realism scores

2. Monitor policy learning stability (reward trends & gradients)
3. Evaluate agreement between baseline realism scoring and the learned policy

Inside the training loop, we decode generated tokens *every batch* and print:

```
print("\n====== SCORE DEBUG ======")

for j in range(len(decoded)):

    print(f"[Sample {i+j}]")

    print(f"Extracted rating (llm_scores): {llm_scores[j].item()}")

    print(f"Baseline REALISM score (batch_vr): {batch_vr[j].item()}")

    print("-------------------------")
```

This allows us to verify:

- The LLM actually emits a *valid* 1–10 rating (due to the issues we had with Qwen)
    - Because we initially observed invalid outputs (e.g., “33”, “The”, “05”), we introduced constrained decoding with AllowedTokensProcessor, and re-verified correctness through this logging.
- The model is not collapsing to a constant mode (e.g., always predicting “1”)
- Compare scores immediately with our baseline (verifiable equation)

On the first batch of each epoch, we print diagnostic information confirming that:

- Prompt tokens are properly masked (-100) so they do *not* influence gradients
- Only continuation tokens contribute to log-probability updates

```
print("\n====== LABEL MASK DEBUG ======")

print("labels unique values:", torch.unique(labels))

print("Count of masked positions (-100):", (labels == -100).sum().item())

print("Count of continuation positions:", (labels != -100).sum().item())
```

This ensures the loss reflects *policy shifts in how the model generates the score*, rather than memorizing or regurgitating the audit prompt.

For the first step of each epoch, we also log:

```
print("\n====== ADVANTAGE DEBUG ======")

print("llm_scores:", llm_scores)
print("batch_vr:", batch_vr)

print("reward:", reward)

print("advantage:", advantage)
```

This verifies:

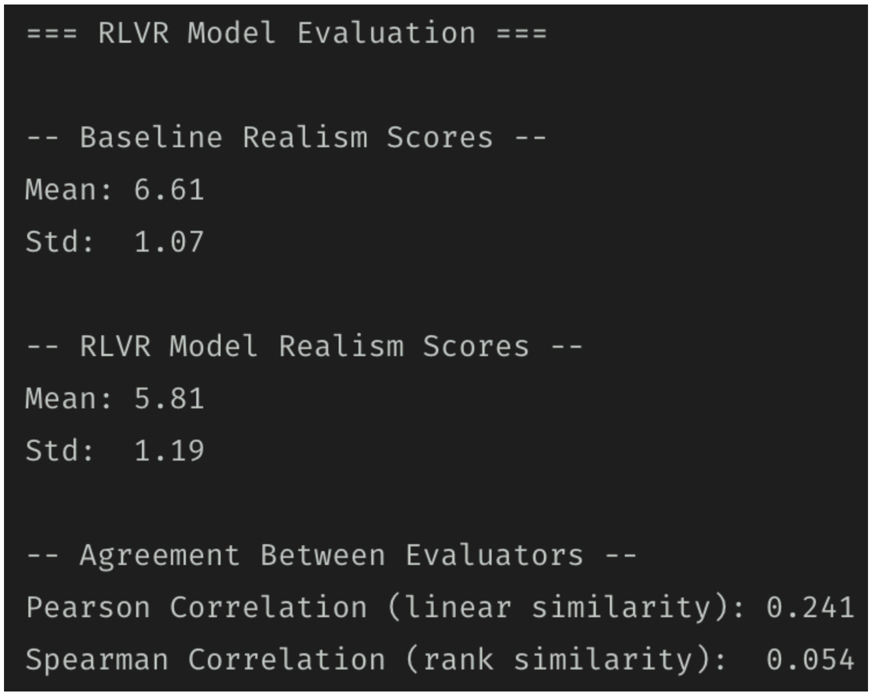
- Rewards are close to 1 when LLM and baseline agree
- Rewards fall smoothly with increasing score error (quadratic penalty)
- advantage is zero-centered → prevents exploding gradients

This was especially important in preventing reward collapse.

Finally at the end of training we calculate the:

- Mean & std realism scores (baseline vs LLM evaluator)
- **Pearson correlation** (numerical agreement)
- **Spearman correlation** (rank-ordering agreement)

As seen in example here:



```
=== RLVR Model Evaluation ===

-- Baseline Realism Scores --
Mean: 6.61
Std:  1.07

-- RLVR Model Realism Scores --
Mean: 5.81
Std:  1.19

-- Agreement Between Evaluators --
Pearson Correlation (linear similarity): 0.241
Spearman Correlation (rank similarity):  0.054
```

This logging strategy let us:

- LLM outputting invalid tokens
- Policy overfitting / score collapse

- Loss diverging or masking bugs
- Model not learning ranking alignment

Ablations we ran with hyper parameters

Our training pipeline involves several hyper parameters. Aside from the typical hyperparameters such as number of epochs, learning rate, or batch size, we also tuned weights to assign to different components of our loss function as well as our realism score.

Loss

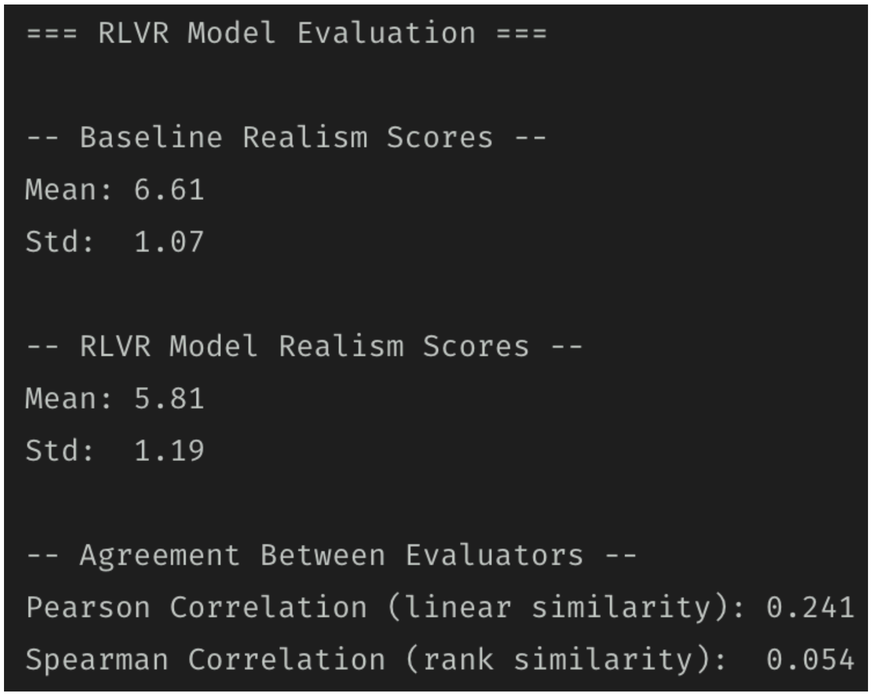
Our loss function consists of two main components, the euclidean distance between predicted and true labels and the KL divergence between true and predicted distributions. These two values are combined in a sum where the KL divergence is weighted between a value of 0 and 1. We found that assigning a small weight to KL divergence (0.01) was the most effective. We suspect this is due to the limited size of our dataset. Since the dataset is quite small, we struggle to capture the true shape of the distribution so the KL divergence is not as effective as it would be with a larger dataset. We found that increasing this weight actually lead to worse correlation as well as more distance between average predicted score and average ground truth score, so we decided to keep the weight for this component small.

Realism score

Our realism score consists of two statistical measures, mahalanobis distance and average distance to K nearest neighbors. We compare generated samples to true samples using a combination of these two metrics to attempt to assign high rewards to generated samples that are within the true distribution without copying any existing ground truth samples. A lower mahalanobis distance is better while a higher average distance to K nearest neighbors is better. We combine these metrics to compute our final realism score using a weighted sum. We found that prioritizing the mahalanobis distance over the average distance to KNN was most effective. If we reversed the priorities, we noticed the gap between average predicted scored and average true scores grew, so we decided to keep this priority order between these two metrics.

What didn’t work and next steps

After many training iterations and experimenting with hyperparameters such as number of epoch, batch size, learning rate and more our judge received these evaluation results when compared to the baseline equation itself as a second classifier model:



```
=== RLVR Model Evaluation ===

-- Baseline Realism Scores --
Mean: 6.61
Std:  1.07

-- RLVR Model Realism Scores --
Mean: 5.81
Std:  1.19

-- Agreement Between Evaluators --
Pearson Correlation (linear similarity): 0.241
Spearman Correlation (rank similarity):  0.054
```

As evident, this definitely an improvement from part 3. Not only are we using a smaller model for part 4, but its evaluation scores are superior. It is more confident in its answers, showing a smaller standard deviation. Not only that but generally when the baseline scores high, our judge somewhat is able to follow similar trends as seen in the pearson correlation. However this being said, its still true that these scores could be better.

Our judge struggled to achieve the high correlation scores for our dataset which we were hoping for. Despite achieving 0.24 pearson correlation, we achieved a near 0 spearman correlation with 0.054, indicating low correlation between our judges evaluations and the verified rewards. Ideally we’d like to have these correlation scores approach >0.7 as much as possible. To accomplish this we could explore more complex loss functions, and give the LLM a better understanding of what it is trying to accomplish with some one-shot or few-shot training. For example, we could explain what realism means for us and what to search for when assigning such a score.

Furthermore, the average realism score that our judge provided was not quite the true average realism score, indicating room for improvement in our training process. Despite training our judge to achieve an average realism score within 8% of the true value, we could close this gap even further by using larger training sets and the techniques discussed in the previous paragraph.