**6th Place Solution**

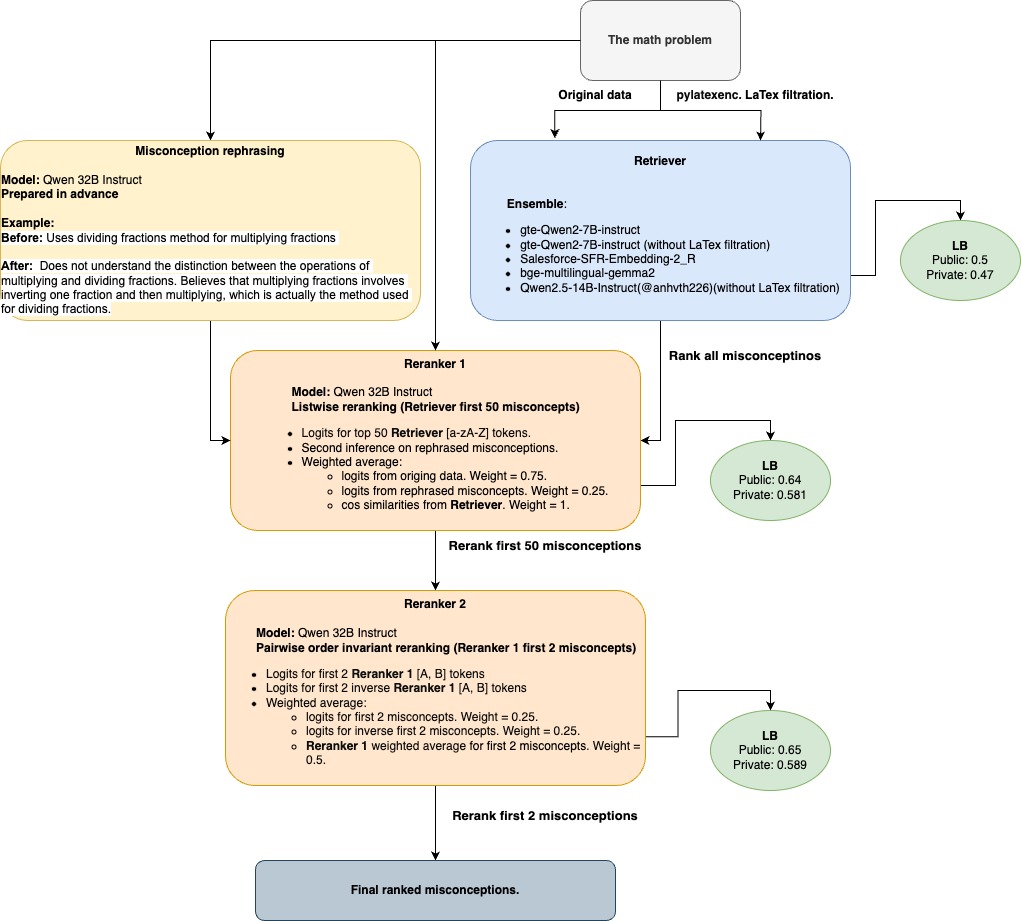
I am very happy to have finally achieved the goal that I set for myself a year ago when I began relatively seriously participating in Kaggle competitions: solo gold.  
Although I had previously reached solo gold on the public a couple of times, I did not succeed to survive on the private. This time, dedication over the last months, focusing on a generalizable solution and the relatively similar distributions of public and private data have enabled me to reach this goal.

**Gratitude**

I want to thank Kaggle and the hosts for organizing such an interesting competition, where we had to explore the reasoning abilities of different approaches to effectively rank math misconceptions by their relevance to the distractors.  
Special gratitude for designing public/private splits with a balanced number of seen and unseen misconceptions from the training data (I am sure that gold places that survived agree with me). That allowed to reflect the robustness of the developed approaches on the leaderboard, resulting in only a very moderate shuffle on the private. After the last competitions, I really feel that I should thank for that :)

**Solution**

**Diagram showing the general overview of the solution:**



**Validation strategy**

From the beginning, as most participants, I could not find a good correlation between CV by the group split on question id and the public LB. So it became obvious that hidden data contains multiple unseen misconceptions that were not present in the train and I should make the validation split in the way that it would incorporate a part of the unseen misconceptions that train data would not have.  
Despite the group split on question id already provided a certain proportion of the unseen misconceptions, they usually took up maximum up to 20% of all misconceptions in the fold, that is why a lot of team faced the overfit to seen misconceptions that provided good general CV score even if unseen ones were very suppressed in terms of retrieving.  
For example, if fold consists of **~80% seen** data and **20% unseen**, then with severe overfit to seen misconcepts like 65% MRR and the lack of good retrieving for unseen like 30% MRR, the weighted average of the MRR would be  
0.65\*0.8 + 0.3\*0.2 = 0.58  
and that would be far away from what the LB would display.  
Finally, I based my validation stratagy on an assumaption that the hidden distribution is quite balanced in terms of seen/unseen misconceptions and the key for the good CV is to set equal weight to seen and unseen misconceptions.  
To maintain a high dynamic of experiments and avoid spending too much on VMs, I stuck with using only one fold as a validation set among experiments. This strategy proved sufficient, as the results across different folds were very similar.

**Synthetic data generation**

Attempts to train retriever models solely on the provided train data showed bad results, so it became inevitable to generate synthetic data for unseen misconceptions.  
While experimenting with LORA pretraining and few-shot generation using Qwen 32/72 instruct, I discovered that LORA pretraining seems to decrease the models' reasoning abilities (catastrophic forgetting ?). The best approach turned out to be few-shot generation. Later, I came across the great [EEDI paper](https://arxiv.org/pdf/2404.02124), which reached a similar conclusion that few-shot generation is more effective for this task.

The main drawback of few-shot generation was that the generated samples were very similar in formulation to the few-shot examples. Any attempt at prompt engineering couldn’t change this without losing the correct connection between distractors and misconceptions.

**Generation details:**

* Qwen 72B instruct (32B struggled with tricky misconcepts)
* 3 random few-shot examples for most similar train misconcepts
* Full math test generation aligned with train data structure, but with only 1 distractor relevant to misconcept
* 3 math tests per misconcept repeated for 3 different seeds. So in total 9 tests for each misconcept
* 2xA40 took 3 hours to generate all samples

**Retrievers**

The LaTex format of the math problems contained many redundant symbols that slowed the convergence. So instead of manually filtering the text, I researched open-source libs to convert LaTex to plain text and simplify equations. Although most libs ingored most redundant symbols, I found pylatexenc lib, which handled the conversion quite well, mostly without losing important elements from the text or equations (there were a few cases in the training data with redundant symbols that did not follow LaTex formatting at all). However, I included two models in the final ensemble that used data without LaTex filtration to ensure no important information was lost.  
An interesting moment is that, even though the model achieved a much higher MRR for unseen misconceptions compared to seen ones (due to synthetic oversampling) after training including the generated data, the recall remained the same between seen and unseen misconcepts. I believe the root cause is that the synthetic data allowed the model to converge well for simpler misconceptions but struggled to generate math tests for complex misconcepts. As a result, the model could retrieve most unseen misconcepts well (resulting in a high MRR) but was unable to find relevance for other complex misconcepts, leading to the same recall as for underrepresented seen misconcepts which math tests were similar between train and validation splits.

**Retriever details:**

* Models: gte-Qwen2-7B-instruct, Salesforce-SFR-Embedding-2\_R, bge-multilingual-gemma2, [Qwen2.5-14B-Instruct](https://www.kaggle.com/code/anhvth226/eedi-11-21-14b)
* Lora parameters:
  + rank = 64
  + alpha = 128
* Batch size = 32
* Negatives per sample = 5 (from 0 to - 30 most similar negatives)
* Two neg mining iterations:
  + initial neg mining with the subsequent 1600 training steps
  + second neg mining iteration by the pretrained model with the subsequent 400 training steps
* BNB 4 bit quantization
* 1xRTX 3090 took 9 hours to train one model

**Rerankers**

Using the DataCollatorForCompletionOnlyLM was crucial to exclude from training prompt tokens.  
Another critical aspect was excluding all test misconceptions from training. I observed that training solely on the training misconceptions allowed the model to generalize effectively to unseen misconceptions. Excluding test msicocnepts gave me probably the biggest LB improvement after reaching 0.6 on the public = ~0.61 - 0.63.

**Distillation**

Close to the deadline, I discovered an interesting trick: first, train the model on one seed. Then, create a dataset using a different seed (which usually alters the set of negative samples and the position of the correct misconception). Use the already trained model to infer logits on this new dataset. Finally, retrain the model with KL divergence distillation combined with simple cross-entropy loss. This approach yielded a ~0.01 improvement on the public.  
My intuition behind this trick is that, for some samples, more than one misconception may technically be correct. Overfitting the model to a single misconception reduces its ability to generalize. By incorporating soft labels through distillation, we mitigate overfitting and improve the model's generalization to unseen misconceptions.

**First stage reranker**

I randomly sampled 49 negative misconceptions from the 100 most similar misconceptions and placed the correct misconception at a random position. Then, I trained the model to classify which misconception was most relevant. The random order generated through sampling helped avoid overfitting to the initial order, where the retriever often placed the correct misconception among the first few options.

**Misconception rephrasing**

I noticed that misconceptions vary significantly, for example, some describe what knowledge is lacking, while others describe what the student believes. To address this, I rephrased the misconceptions into a general format that includes both the missing knowledge and the student’s belief that aligns with the misconception. Even when the rephrased misconceptions semantically repeated the original ones, this rephrasing served as effective regularization during model inference.  
I used the inference logits from the rephrased misconceptions and combined them with the logits from the original misconceptions, which resulted in the ~0.01 public improvement.  
To rephrase misconceptions Qwen 32B-Instruct was used.

**Second stage reranker**

Analyzing the logits obtained from the first-stage reranker, I noticed that Qwen tends to preserve the initial order of misconceptions as sorted by the retriever. I found further validation for this observation in [this excellent paper](https://arxiv.org/pdf/2306.17563), which inspired me to retrain Retrievers with more focus on MRR and develop the second-stage reranker to refine the results of the first stage reranker.  
In the second stage, I selected a small window of the top ranked misconceptions from the first stage reranker and applied all possible permutations to their order. I then averaged the logits across these permutations and reranked misconcepts from that window again. For my surprise, the optimal window size turned out to be 2. This means I took the top 2 misconceptions from the first stage reranker, performed inference on them, inverted their order, ran inference again, and finally used the averaged logits for the final reranking.  
**Reranker details:**

* Qwen 32B instruct was used
* Lora parameters:
  + rank = 64
  + alpha = 128
* Batch size = 2 \* 6 grad accum
* AWQ 4 bit quantization
* 1xA40 took 6 hours to train one model

**What did not work**

* **A lot of things**
* Training fp16 Rerankers with all available lora adapters with subsequent qunatization
* Training Qwen 72B as a Reranker. I got the same results as for 32B
* Using synthtetic data in the Reranker training

**Hardware**

Since Ukraine's infrastructure is under constant attacks and, as a result, we have blackouts every day, renting VMs became inevitable.  
I mostly used A40, and for some experiments, I rented A100.  
In total, this cost me about $300.