1.Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

My real name is 刘欣迪 (Xindi Liu), but I usually use my online English name Dylan Liu. I'm a freelance programmer (AI related) with 7 years of experience. One of my main incomes now is prizes from data science competition platforms (Topcoder, Kaggle and DrivenData, although I have no income yet on Kaggle, only a solo gold medal).

2.What motivated you to compete in this challenge?

I would love to participate in various competitions involving deep learning, especially tasks involving natural language processing or LLM . At the same time, competition prize money is also one of my main incomes.

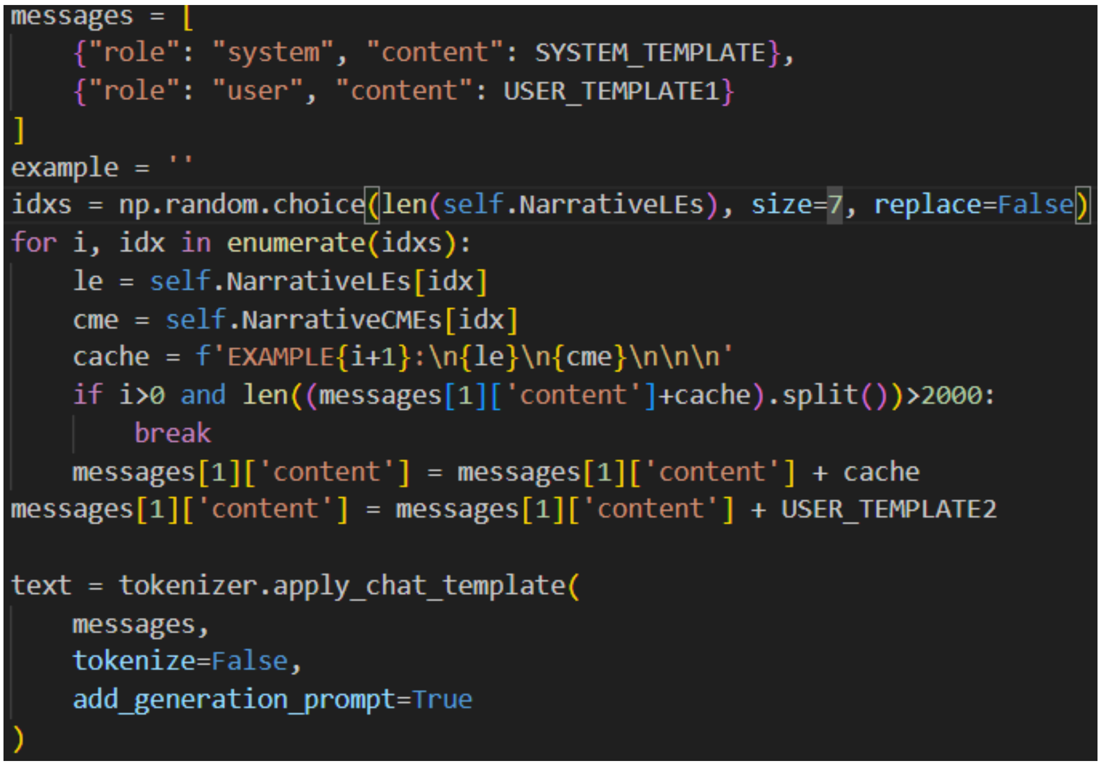
3.High level summary of your approach: what did you do and why?

At first, I saw that there were only 4000 samples, too less data made me try to generate data using some LLMs in a few-shot way. About 4000 samples were generated in the first step.
I used the commonly used text classification model: Deberta-v3. The classification head was changed to a combined classification head, of which the input size is 21+6+12, where 21 is first 21 classes of binary classification, and 6 and 12 are last 2 classes of multi-class classification. I tried different sizes of Deberta-v3, and Deberta-v3-large performed the best.
I tried data label correction, which is to relabel some data with higher loss. It improved cv by ~0.04. After doing this, the cv (cross validation) score reached ~0.845 and lb (leaderboard) score reached 0.854.
Since the data is too less and the generated data is not well aligned with given labels, I was focused on generating some unlabeled data for semi-supervised learning, then the Deberta-v3-large model was used to predict soft labels for the unlabeled data.
I then trained a gemma2-9b model with lora on all the data I have. The initial gemma2-9b model got a cv score of ~0.852 and a lb score of ~0.86.
With different data generating LLMs and random seeds, the semi-supervised learning was repeated using the gemma2-9b model as the soft labeling model, and the gemma2-9b model was also used for soft labeling all the other data. Each repetition generated ~4000 pieces of data.
20000 pieces of data (including official data) were used for training the final model. The final model got a cv score of ~0.86(~0.849 without threshold searching) and a lb score of ~0.863.
Finally, 2 more gemma2-9b models and 1 gemma2-2b model with different lora parameters were trained and combined with the final model in the weighted way. The final submission achieved a lb score of ~0.869.

4.Do you have any useful charts, graphs, or visualizations from the process?

In the early stage, I made some charts to show text length and label distribution. Apparently labels are unbalanced, especially InjuryLocationType and WeaponType1.

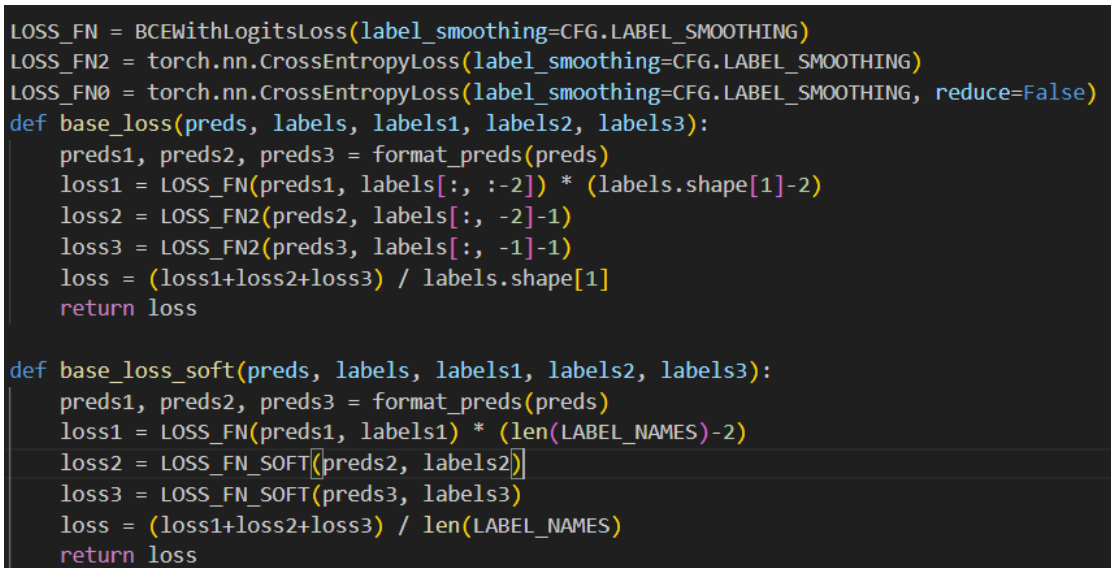
5.Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.



```python
messages = [
    {"role": "system", "content": SYSTEM_TEMPLATE},
    {"role": "user", "content": USER_TEMPLATE1}
]
example = ''
idxs = np.random.choice(len(self.NarrativeLEs), size=7, replace=False)
for i, idx in enumerate(idxs):
    le = self.NarrativeLEs[idx]
    cme = self.NarrativeCMEs[idx]
    cache = f'EXAMPLE{i+1}:\n{le}\n{cme}\n\n\n'
    if i>0 and len((messages[1]['content']+cache).split())>2000:
        break
    messages[1]['content'] = messages[1]['content'] + cache
messages[1]['content'] = messages[1]['content'] + USER_TEMPLATE2

text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True
)
```

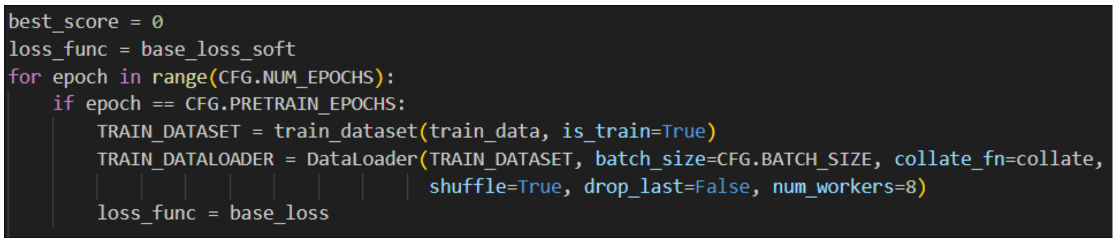
The above code is the prompt code used to generate data for semi-supervised learning in a few-shot way. Each time some official data is randomly chosen as few-shot examples to generate realistic and diversified extra data.



```python
LOSS_FN = BCEWithLogitsLoss(label_smoothing=CFG.LABEL_SMOOTHING)
LOSS_FN2 = torch.nn.CrossEntropyLoss(label_smoothing=CFG.LABEL_SMOOTHING)
LOSS_FN0 = torch.nn.CrossEntropyLoss(label_smoothing=CFG.LABEL_SMOOTHING, reduce=False)
def base_loss(preds, labels, labels1, labels2, labels3):
    preds1, preds2, preds3 = format_preds(preds)
    loss1 = LOSS_FN(preds1, labels[:, :-2]) * (labels.shape[1]-2)
    loss2 = LOSS_FN2(preds2, labels[:, -2]-1)
    loss3 = LOSS_FN2(preds3, labels[:, -1]-1)
    loss = (loss1+loss2+loss3) / labels.shape[1]
    return loss

def base_loss_soft(preds, labels, labels1, labels2, labels3):
    preds1, preds2, preds3 = format_preds(preds)
    loss1 = LOSS_FN(preds1, labels1) * (len(LABEL_NAMES)-2)
    loss2 = LOSS_FN_SOFT(preds2, labels2)
    loss3 = LOSS_FN_SOFT(preds3, labels3)
    loss = (loss1+loss2+loss3) / len(LABEL_NAMES)
    return loss
```

These are loss functions. The soft cross entropy loss with soft labels improved cv score a lot.



```
best_score = 0
loss_func = base_loss_soft
for epoch in range(CFG.NUM_EPOCHS):
    if epoch == CFG.PRETRAIN_EPOCHS:
        TRAIN_DATASET = train_dataset(train_data, is_train=True)
        TRAIN_DATALOADER = DataLoader(TRAIN_DATASET, batch_size=CFG.BATCH_SIZE, collate_fn=collate,
                                      shuffle=True, drop_last=False, num_workers=8)
        loss_func = base_loss
```

This part makes the first epoch of training as pretraining. After pretraining, only the official data with hard labels is used for later training. This makes sure that the extra data is used but not overfitted.

6.Please provide the machine specs and time you used to run your model.

●CPU (model): intel i7

●GPU (model or N/A): NVIDIA 4090

●Memory (GB): 48GB

●OS: Linux

●Train duration: ~40h (with 4090GPU*3)

●Inference duration: ~3h (on the competition’s virtual machine)

7.Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

1000 test examples are too few for a stable and reliable test dataset, not to mention that this is a 23-class classification task combining binary and multi-class classification. In theory, my best submission shouldn't be the best, because its CV score is not outstanding. The best cv score is ~0.87, but its lb is mediocre (0.8639 public, 0.8590 private). Different hyperparameters have a great impact on the results of the model. There are some submissions similar to the best submission, but the score gap between them and it is not small. So it is recommended to do some hyperparameter tuning when training your model.

8.Did you use any tools for data preparation or exploratory data analysis that aren’ t listed in your code submission?

I used LLMs of gemma2, qwen2.5 and mistral with different parameters (like random seeds, model sizes, quantization settings, etc) for data generation. The idea is to enlarge the diversity of the generated data.

9.How did you evaluate performance of the model other than the provided metric, if at all?

I simply used classic classification loss: cross entropy loss. I found that it was aligned with the provided metric very well.

10.What are some other things you tried that didn’ t necessarily make it into the final workflow (quick overview)?

I tried to implement the provided metric as the loss function (You can find the loss function average_f1_loss() in the training code). Theoretically, the metric as loss is the best loss, but after some steps of training with this loss, the model stopped convergence. I checked the model’s output and found out the output is all 1 for binary classes. I also tried debugging and modifying the loss implementation but failed to solve it.
I also tried to combine the above loss with the cross entropy loss, but the result was worse than that of using only the cross entropy loss.
I tried mistral-7b and some other smaller models, but due to poor results and limited resource, I stopped training them more.
I tried training with all the extra data, and it improved the cv score but the lb score was dropped a lot. I believe that it’s because the very few text data caused very unstable lb performance.

11.If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

I will be focused on extra data collection and data generation. According to my experience, for this kind of classification task with many unbalanced classes, a stable and reliable model requires at least 100k diverse and high-quality data, so I will try to repeatedly perform a cycle of generating new data, screening new data, and training new model.

12.What simplifications could be made to run your solution faster without sacrificing significant accuracy?

My model is already very fast that 1 model/1 fold takes only 10-15min to finish submission test. If you need a faster solution, you just need to train fewer models and do more hyperparameter tuning to get an efficiency solution.