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 6 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 sequence prediction or natural language processing. 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?

I first used opencv to extract rotation angles between images, that is, convert the image sequences into rotation sequences. The rotation sequences are then used as the feature input of the decoder part of a T5 model to train the T5 decoder with a custom loss that is the same as the official performance metric. During the training process I used a data augmentation method that disrupts the input sequences.

Because I would tend to use deep learning to solve problems. Extracting image features directly through deep learning will lead to serious overfitting (because there are very few types of spacecraft), and the requirements for equipment are very high, so I converted the image sequence into the rotation sequences. The T5 model's decoder part in the generation mode only takes one piece of feature input at a time, and the model will not use subsequent data, this perfectly meets the requirements of this competition. Since the movement trajectory of the camera is relatively random, I shuffled the sequence order as a method of data augmentation.

In addition, I spent a lot of time training models of various sizes (3 folds of tiny, 5 folds of mini, 5 folds of small and 2 folds of base) for more epochs, because the inference speed is much higher than I expected so that more models can be supported, but the results are not ideal, where the results are not much different from using only the tiny model. Maybe I missed out on the efficiency prize because of this.

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

The only visualization related thing I did was downloading Blender and putting some coordinates/rotations into it to see the visual relation between the camera and spacecraft.

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

```
def decompose_label(self, label, r0):
    locs = label[:, :3]
    locs[:, 0] = r0 - locs[:, 0]
    uint_locs = locs / np.linalg.norm(locs, axis=-1, keepdims=True)
    rots = label[:, 3:]
    delta_rots = rots[:-1]
    #delta_rots[:, 1:] = -delta_rots[:, 1:]
```

```
        new_uint_locs = [uint_locs[0]]
        new_rots = [rots[0]]
        for i in range(len(delta_rots)):
            new_rots.append(apply_delta_quaternion_train(rots[i+1],
delta_rots[i]))
            new_uint_locs.append(rotate_point_by_quaternion(uint_locs[i+1],
delta_rots[i]))
        return np.array(new_uint_locs), np.array(new_rots)

    def recompose_label(self, uint_locs, rots, lens, r0):
        new_locs = [uint_locs[0]*lens[0]]
        new_rots = [rots[0]]
        new_locs[0][0] = r0 - new_locs[0][0]
        for ul, ro, le in zip(uint_locs[1:], rots[1:], lens[1:]):
            delta_rot = new_rots[-1] * -1
            delta_rot[0] = -delta_rot[0]
            new_loc = rotate_point_by_quaternion(ul, delta_rot) * le
            new_loc[0] = r0 - new_loc[0]
            new_locs.append(new_loc)
            new_rots.append(apply_delta_quaternion_train(ro, delta_rot))
        return np.concatenate([new_locs, new_rots], axis=-1)
```

These two functions are an important part of data augmentation.

The first function is responsible for decomposing labels into unit vectors of the camera to the aircraft and rotation changes from the previous frame, which will make the labels of the sequence independent of each other. The output here will then be shuffled.

The second function will recombine the unit vectors and the rotation changes that the label is decomposed into and simulated random length into augmentated labels.
The probability of doing this part of data augmentation is 40%

```
def label_transform(labels):
    out = []
    l0 = labels[0]
    rot0 = l0[3:]
    true_loc0 = copy.deepcopy(l0[:3])
    true_rot0 = copy.deepcopy(l0[3:])
    true_loc0 *= -1
    true_rot0[1:] *= -1
    l0[3:] /= np.sum(l0[3:]**2)**0.5
    for l1 in labels:
        l1[3:] /= np.sum(l1[3:]**2)**0.5
        rot1 = l1[3:]
        true_loc1 = copy.deepcopy(l1[:3])
        true_rot1 = copy.deepcopy(l1[3:])
        true_loc1 *= -1
        true_rot1[1:] *= -1

        rot = apply_delta_quaternion_train(true_rot1, rot0)
        rot /= sum(rot**2) ** 0.5
        rot[1:] *= -1

        loc = true_loc1 - true_loc0
        loc = rotate_point_by_quaternion(loc, true_rot0)
        loc *= -1
```

```
            #rot[0] = -rot[0]
            out.append(np.concatenate([loc, rot], axis=0))

        #for i in range(len(out)):
        #    out[i][:3] = out[i][:3] - out[0][:3]
        return np.array(out)
```

This is the important function in the second part of data augmentation. The second part of data augmentation is to randomly reverse (50%) the sequence data and randomly delete the beginning part (0%-50%) of the sequence data, and then reset the first frame as the reference (coordinates (0, 0, 0) and rotation (1, 0, 0, 0)).

The probability of doing this part of data augmentation is 60%

```
    def predict(self, rot, rs, r0, past_key_values=None):
        # rot: the single rotation input
        # rs: the range value of the input image
         # r0: the first range value of the input
        # past_key_values: the model's memory
        embed = torch.cat([self.trans(rot[None]), rs[..., None, None]],
dim=-1)
        decoder_out = self.decoder(inputs_embeds=embed[None],
use_cache=True,
                                   past_key_values=past_key_values)
        out = decoder_out.last_hidden_state[0]
        past_key_values = decoder_out.past_key_values
        out = self.head(out)
        loc_offset = self.head2(out)
        out2 = out[:, -4:] / torch.linalg.norm(out[:, -4:], dim=-1,
keepdim=True)
        #out = self.head2(out) * 400
        base_loc = torch.tensor([[-r0, 0, 0]]*out2.shape[0],
dtype=out2.dtype,
                                  device=out2.device)
        out = rotate_vector_by_quaternion(base_loc, out2) - base_loc
        out = torch.cat([out+loc_offset, out2], dim=1)
        return out, past_key_values
```

This is the prediction part of the model. The characteristics of transformer's decoder (that is, the model in the generation mode only takes one piece of feature input at a time, and the model will not use subsequent data) perfectly meets the requirements of this competition. In addition, the model's head is modified to determine the camera's location using the predicted rotation and the input range value, and then plus the predicted location offset to the camera's location to get the final location.

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

●CPU (model): ADM ryzen5
●GPU (model or N/A): NVIDIA 3060
●Memory (GB): 32GP
●OS: Linux
●Train duration: ~40h (on 3060GPU*2)
●Inference duration: ~2h15m (on the competition's A4v2 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.)?

After training a fold, the current fold and its model name and cv results will be printed. The ideal cv scores should be:
fold0-2 : ~1.9
Fold3 : ~1.905
fold4 : ~1.916

But sometimes the model training process will suddenly diverge, and although the model will continue to converge to a greater or lesser extent, sometimes the convergence does not meet cv score expectation at the end. In this case, you may need to redo parameter turning, changing the random seed and/or the learning rate (such as *2 or /2) may solve it, or you can just drop this model/fold (not use it) if you don't want to train it again. I guess this is because of the gradient instability of the current custom loss. You can find all model configurations at the beginning of 'helper.py' as formed into a configuration list 'MODEL_CONFIGS'.

A simple fix to this case I can think of now is that the model loads the previous checkpoint when a drastic divergence is detected. This is what I would try to do first if there is a second round for this contest.

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

I used the ORB feature and AKAZE feature in opencv to extract the key points in the images, and BFMatcher to obtain the angle features.

I downloaded Blender and put some coordinates into it to see the visual relation between camera and spacecraft.

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

I made some constraints into loss, which can be seen in the loss2_func() function. This loss has two parts: alignment_penalty is the difference between the camera's current orientation and its orientation toward spacecrafts, and range_penalty is the difference between the distance from the camera to the spacecrafts and range data.

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

At first I tried using resnet and some other neural networks to extract image features, but this resulted in overfitting.

I tried other angular feature extraction methods (SIFT and BRISK), but the results were not significantly improved.

I tried other transformer models besides T5, but they all gave bad results.
I manually classified the training data according to the types of spacecraft and designed a 5-fold cv mechanism based on this (the types of the spacecraft are in 'assets/data_type.csv'), but it turns out that my solution faces an under-fitting problem, training loss is always higher than validation loss, so such cv mechanism may not be very useful.

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?

Firstly, I will try to solve the drastic divergence problem mentioned above.

If there is enough time, I think we should collect or generate data on a large number of spacecraft types (or even other objects similar to spacecraft, or simply anything), and then use deep learning methods to extract image features. The current biggest problem with deep learning should be very few types of spacecraft, but non-deep-learning feature extraction methods will lead to underfitting.

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

Very simple, since I'm not implementing any parallel data processing. I believe that the feature extraction process of opencv can be implemented in parallel.

Beyond that, based on my submission record, results close to the present can be achieved with just the tiny model. Obviously, spending a lot of time training bigger and more models is a waste of training and inference time.