Details of CNN-RNN Structure and Experiments on YOLOX-6D-Pose Models

LU Weicheng

2025/04/17

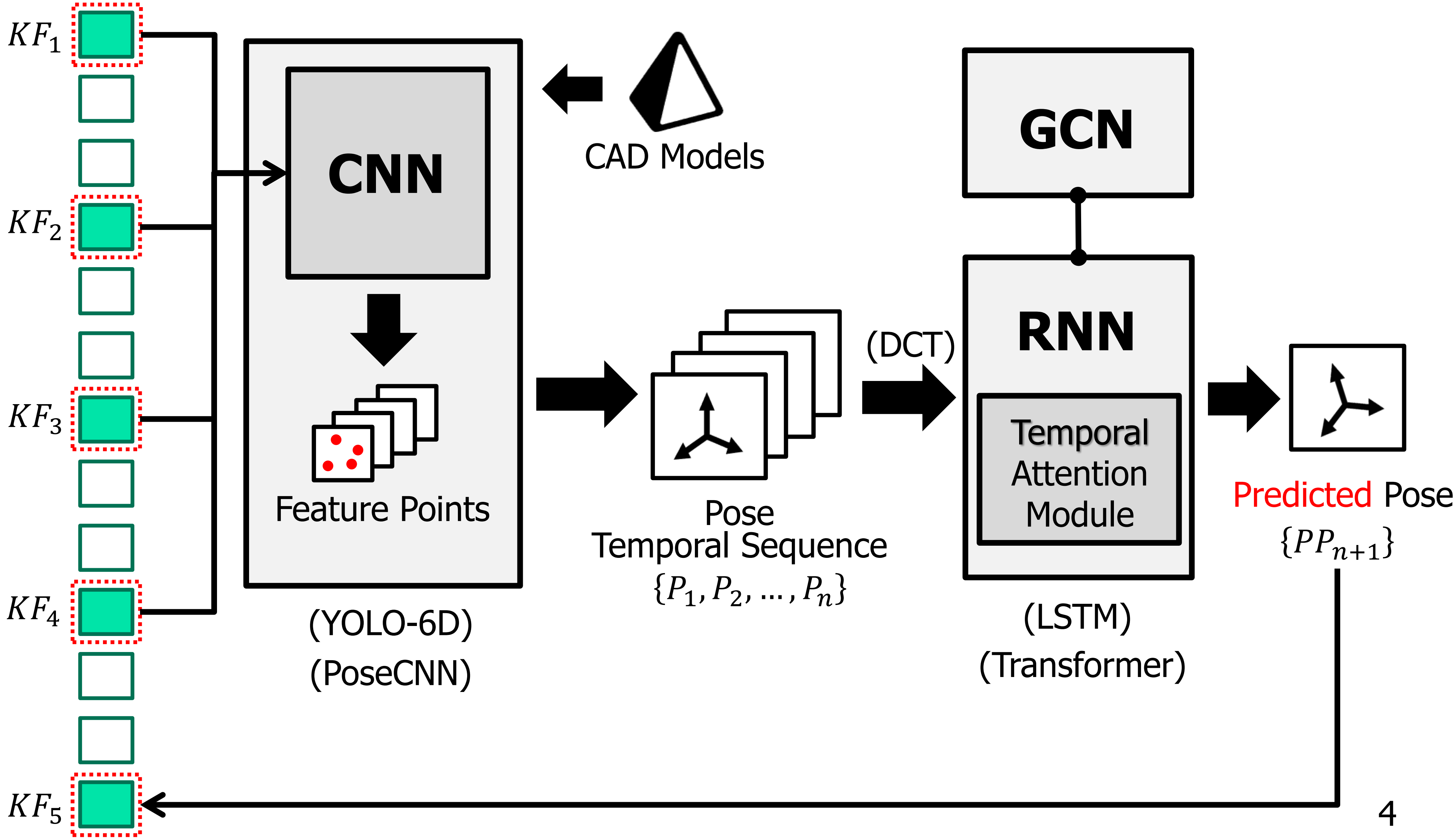
Outline

- Details of CNN-RNN Structure
- Experiments on YOLOX-6D-Pose Models
- Some Ideas
- Future Plan

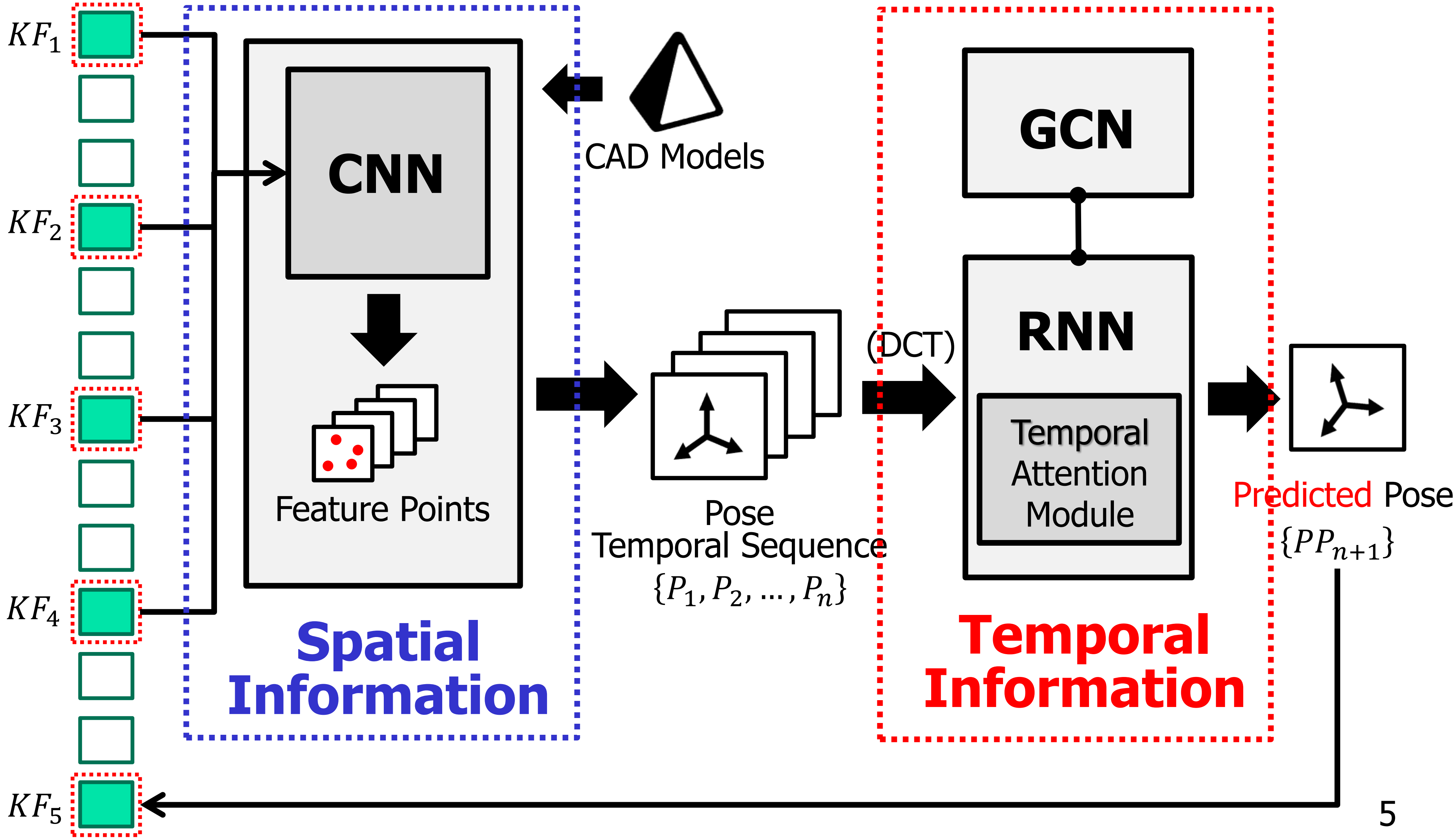
Outline

- Details of CNN-RNN Structure
  - CNN-RNN Structure
  - Spatio-temporal Explanation
  - PART1: From Image to Pose[R|T]
  - PART2: Transition
  - PART3: RNN Prediction
- Experiments on YOLOX-6D-Pose Models
- Some Ideas
- Future Plan

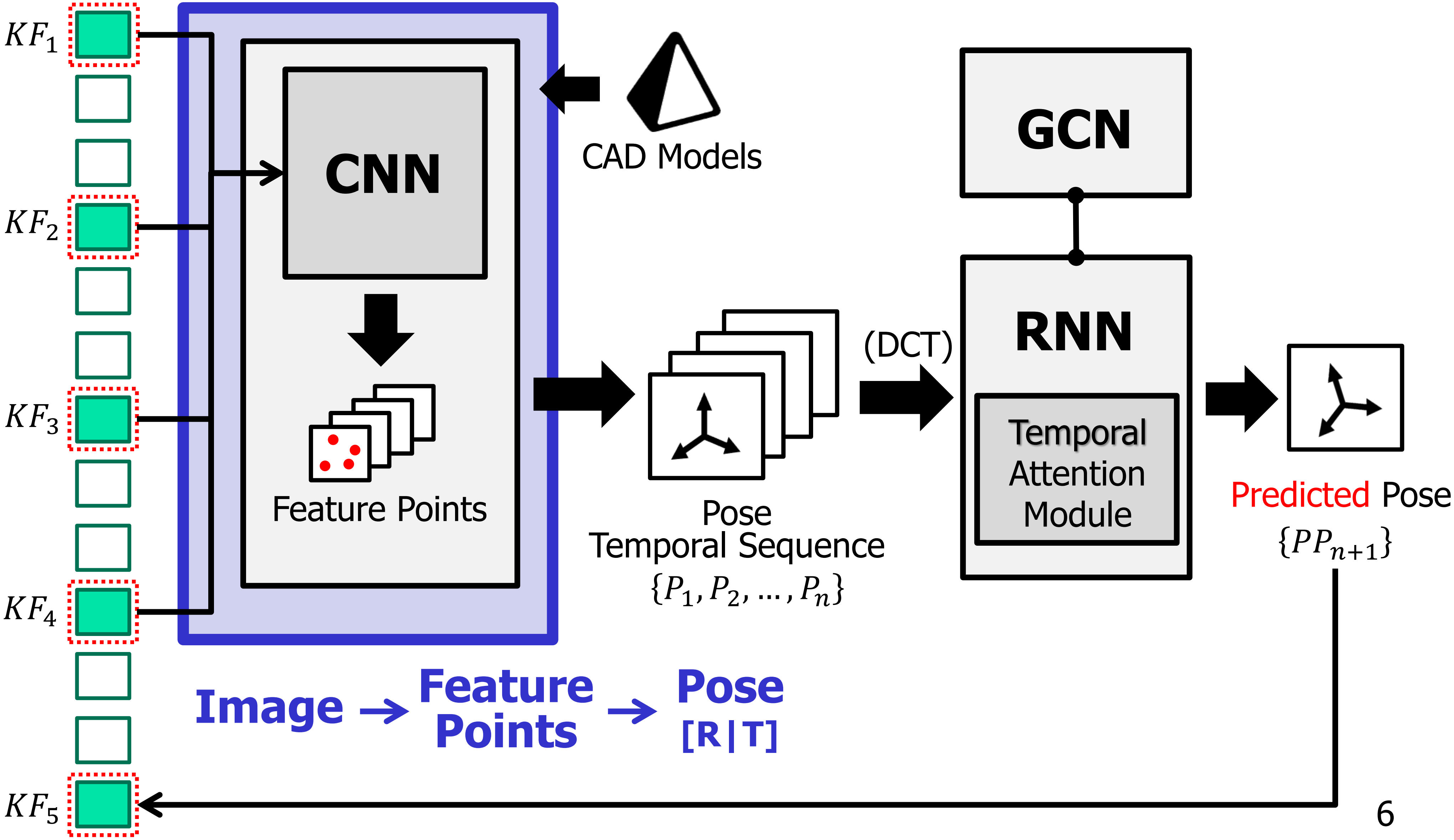
CNN-RNN Structure



Spatio-temporal Explanation



PART1: From Image to Pose[R|T]



KF_1

KF_2

KF_3

KF_4

KF_5

CNN

Feature Points

CAD Models

Pose
Temporal Sequence
$\{P_1, P_2, \ldots, P_n\}$

(DCT)

GCN

RNN

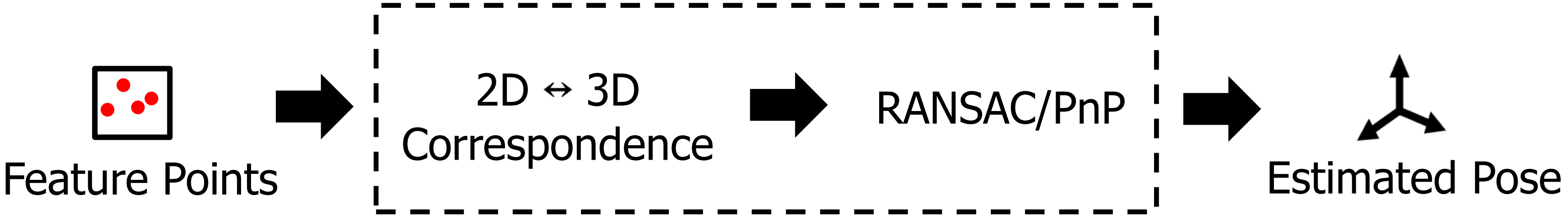
Temporal Attention Module

Predicted Pose
$\{PP_{n+1}\}$

Image → Feature Points → Pose [R|T]

PART1: End-to-end Network can be used

- Two main method of Monocular 6D pose estimation
  - Using Algorithm (Indirect)：



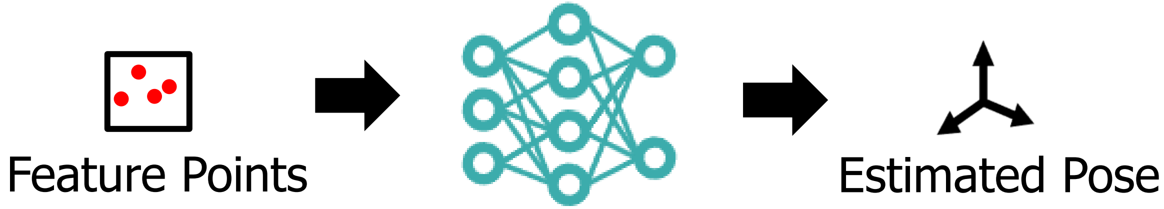
Feature Points

2D ↔ 3D Correspondence

RANSAC/PnP

Estimated Pose

  - Directly regress：

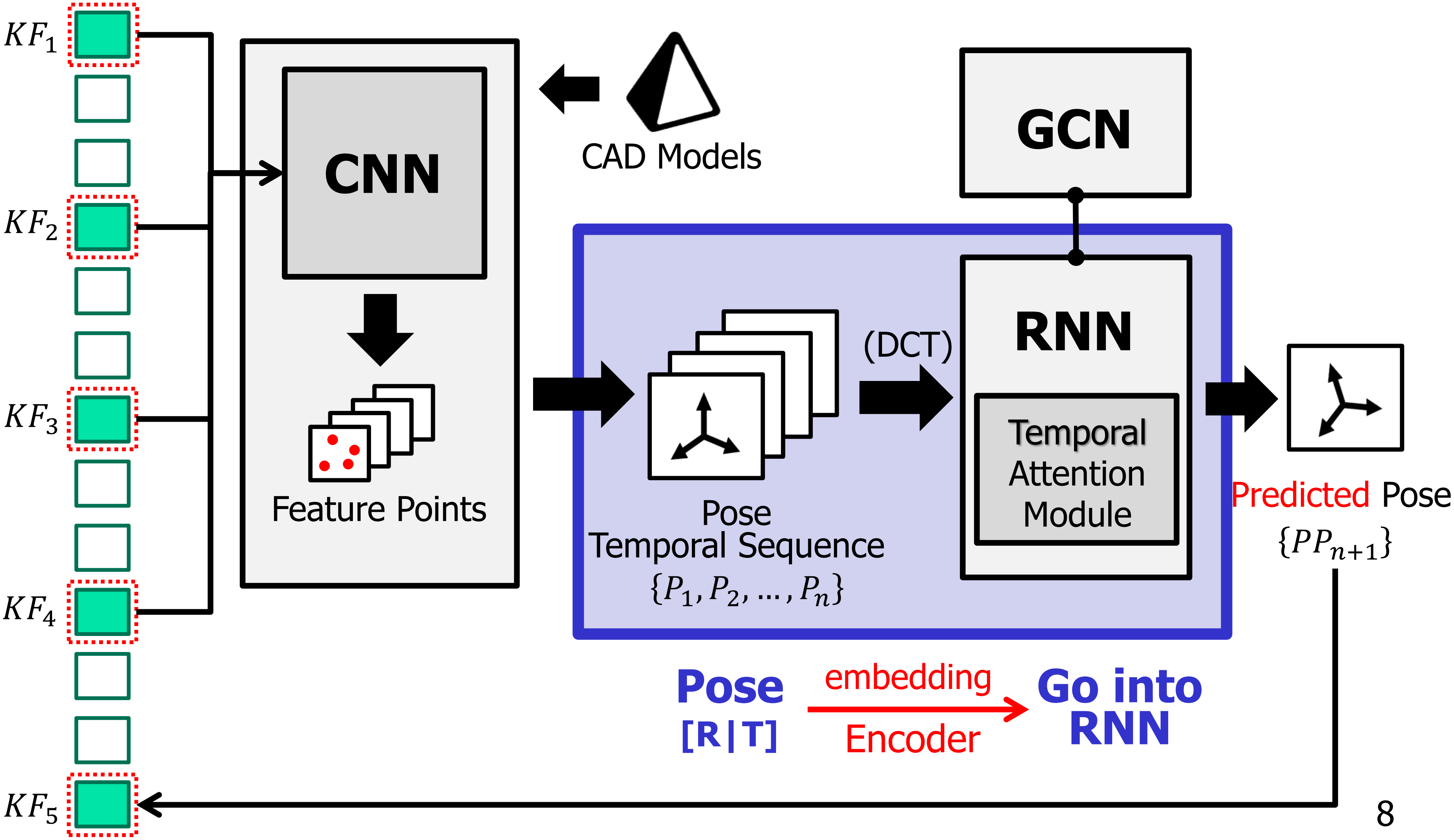


Feature Points

Estimated Pose

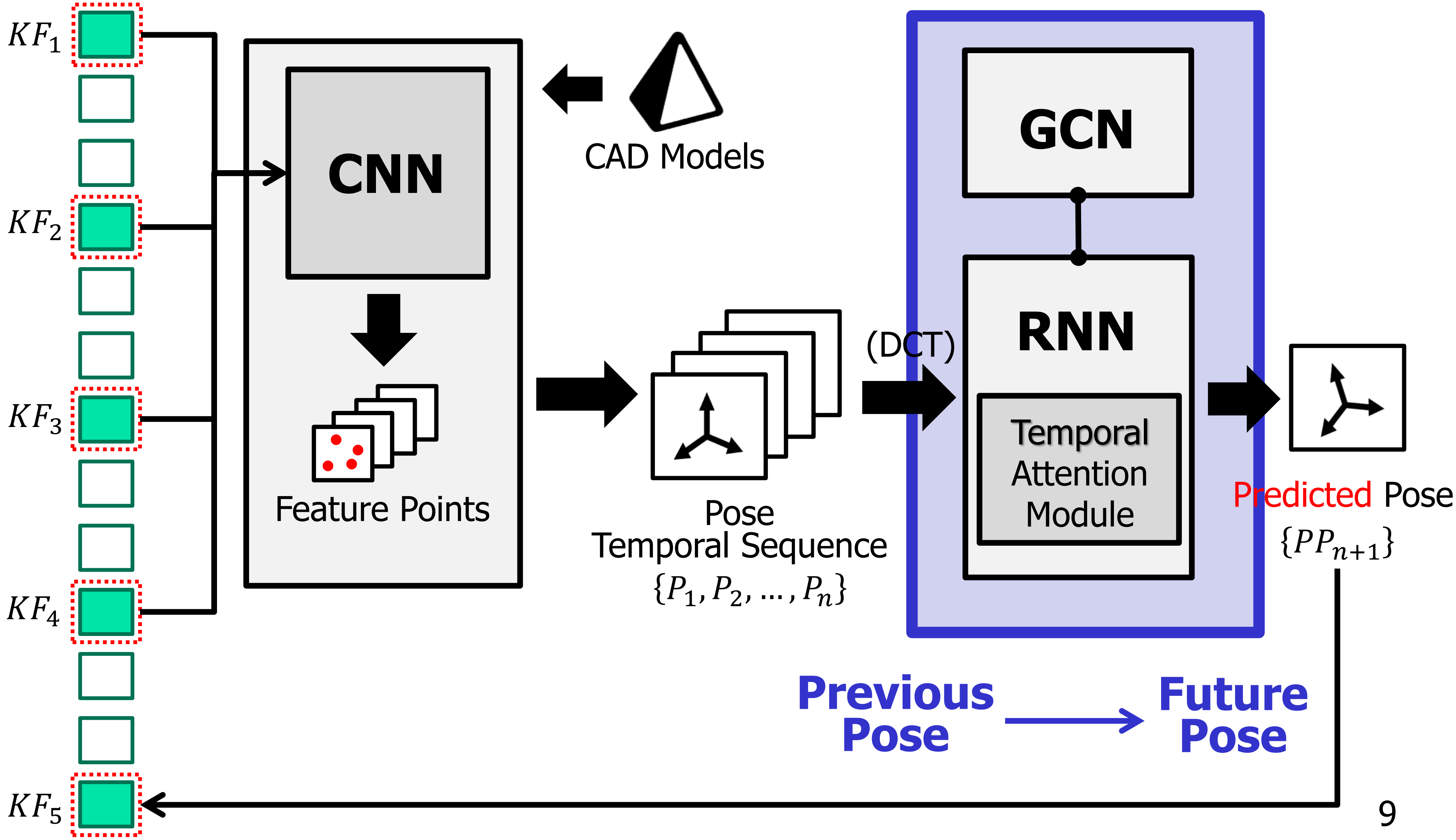
(Note that: They all need to do the Feature Extraction first, using CNN.)

PART2: Transition

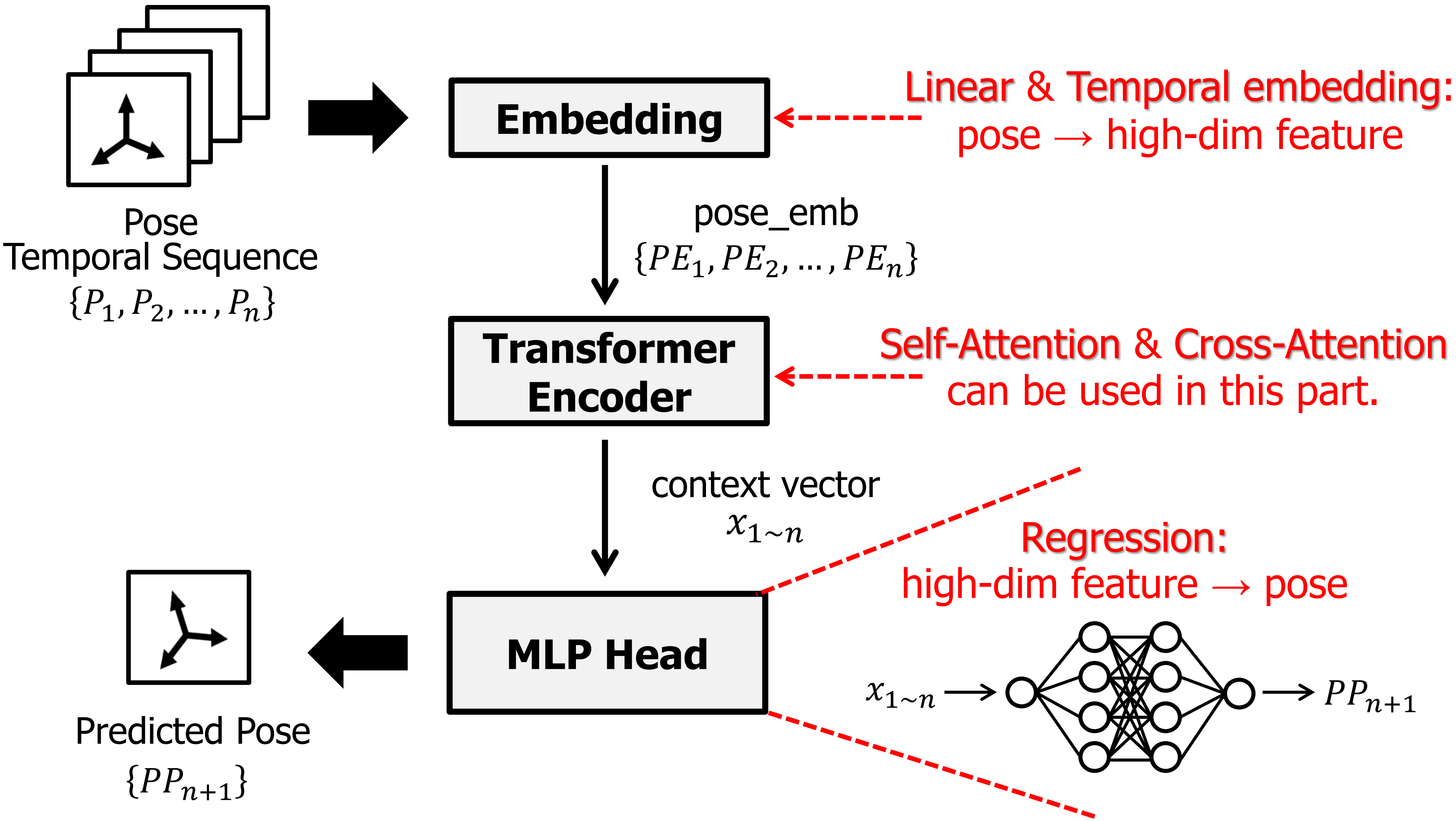


Pose [R|T] $\xrightarrow[\text{Encoder}]{\text{embedding}}$ **Go into RNN**

PART3: RNN Prediction



PART2&3: Details of RNN (Transformer)



Pose Temporal Sequence

$\{P_1, P_2, \dots, P_n\}$

Embedding

Linear & Temporal embedding:

pose → high-dim feature

pose_emb

$\{PE_1, PE_2, \dots, PE_n\}$

Transformer Encoder

Self-Attention & Cross-Attention can be used in this part.

context vector

$x_{1\sim n}$

MLP Head

Regression:

high-dim feature → pose

$x_{1\sim n} \longrightarrow$ $\longrightarrow PP_{n+1}$

Predicted Pose

$\{PP_{n+1}\}$

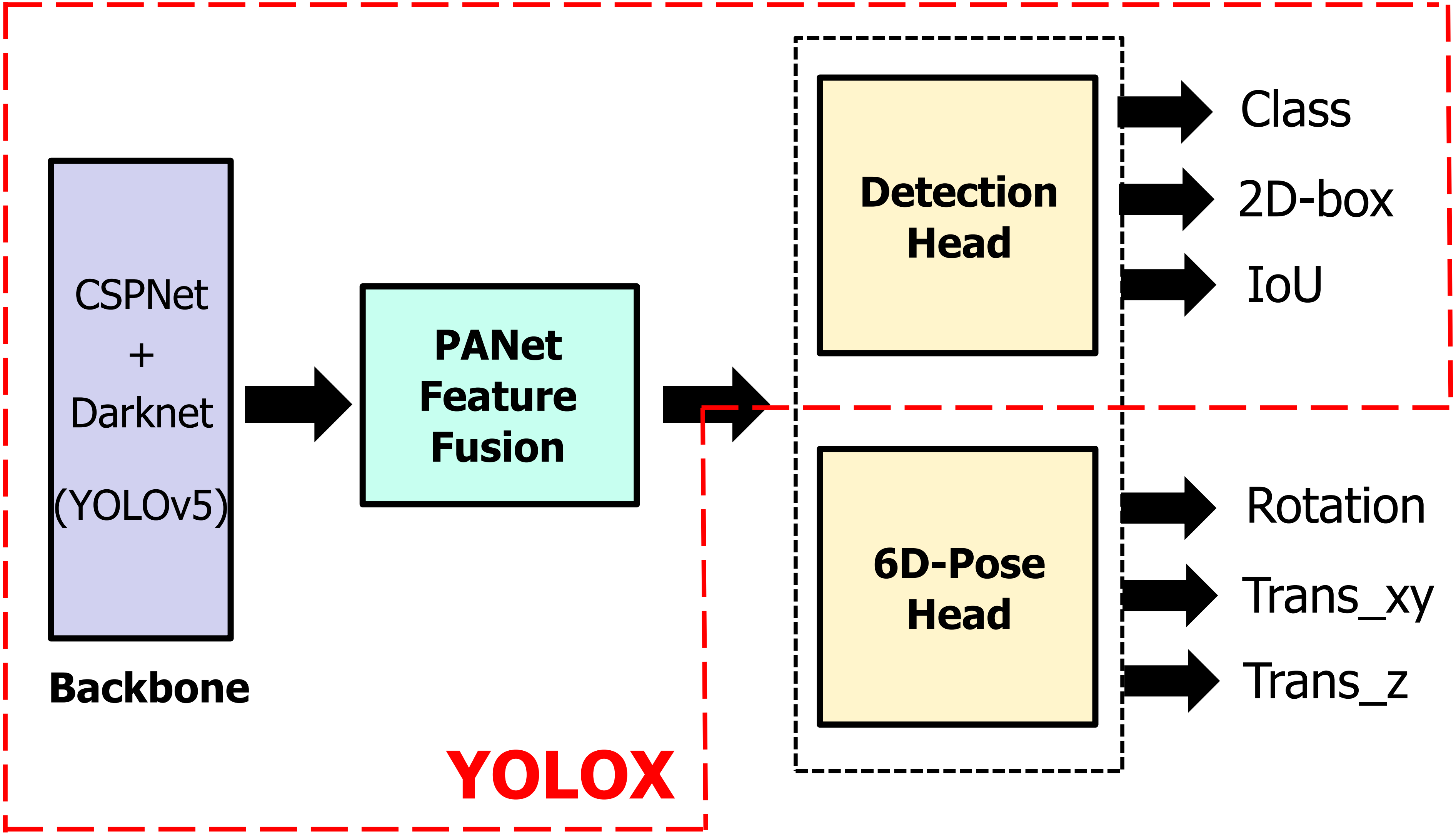
Outline

- Details of CNN-RNN Structure
- Experiments on YOLOX-6D-Pose Models
  - Pose Estimation Network on GPU
  - YOLO-6D-Pose: Basic Structure
  - YOLO-6D-Pose: 6D-Pose Head
  - YOLO-6D-Pose: test on YCBV
- Some Ideas
- Future Plan

Pose Estimation Network on GPU

Method	**PoseCNN**	**RNNPose**	**CRT-6D**	**YOLOX-6D-Pose**	**NOPE**	**YOLOv10-pose**
Year	2018	2022	2023	2024	2024	2025
Accuracy (ADD↑)	79.30%	97.37%	83.7%	x: 88.5% l: 83.7% m: 72.8% s: 66.3%	87.1%	>90%
Speed & Device	~70fps (RTX3090) ~90fps (RTX4090)	~60-20fps (RTX3090)	~38fps (GTX1080Ti)	x: ~34fps l: ~40fps m: ~47fps s: ~60fps (RTX2080Ti)	1fps (V100)	n: ~92fps s: ~87fps m: ~75fps b: ~68fps (RTX2080Ti)
My Comment	Classic e2e model which regress semantic and 6D pose.	Use RNN Structure to do pose refinement.	Use RNN Structure to do pose refinement.	YOLOX based, e2e network, 2*4=8 versions of different model size.	Cross-attention module, but slow.	YOLO based, but only keypoint head and no 6D-pose head.

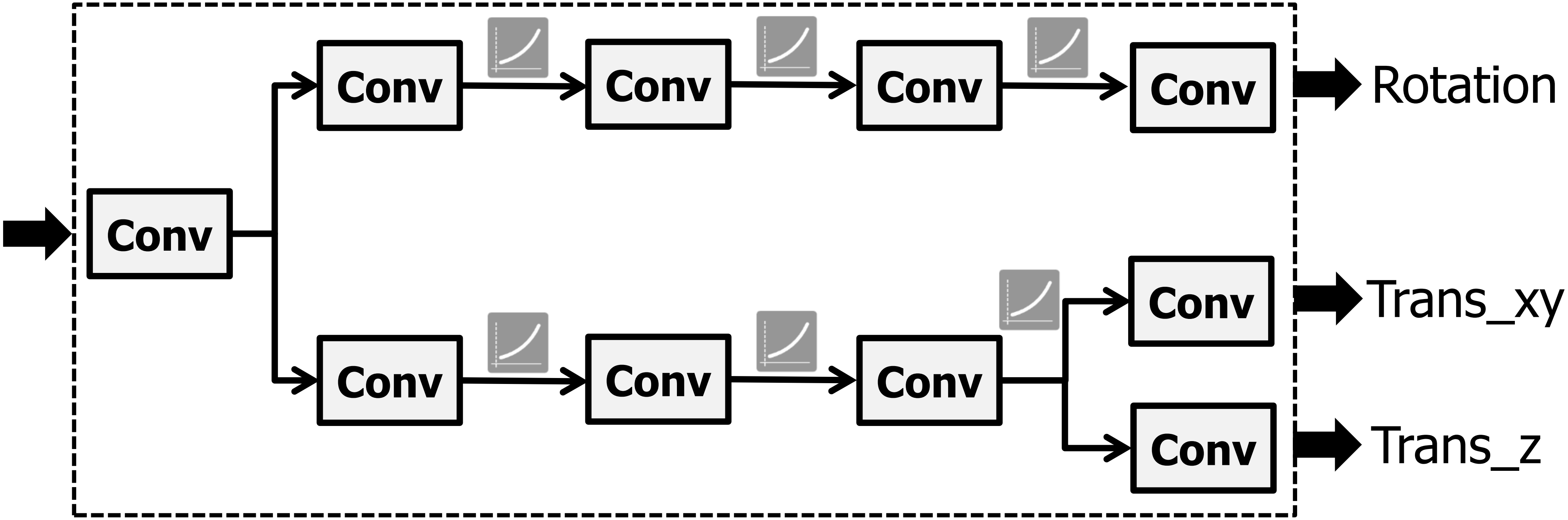
YOLOX-6D-Pose: Basic Structure



Maji D, Nagori S, Mathew M, et al. YOLO-6D-Pose: Enhancing YOLO for single-stage monocular multi-object 6D pose estimation[C]//2024 International Conference on 3D Vision (3DV). IEEE, 2024: 1616-1625.

YOLOX-6D-Pose: 6D-Pose Head

- In 6D-Pose Head, there are Convs everywhere.
- Instead of using PnP, it directly regress Pose [R|T] by just connecting convolution layers.

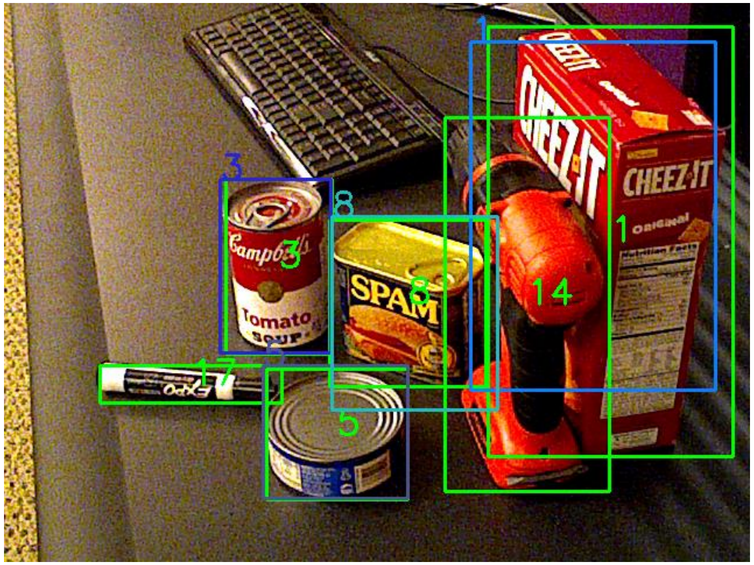


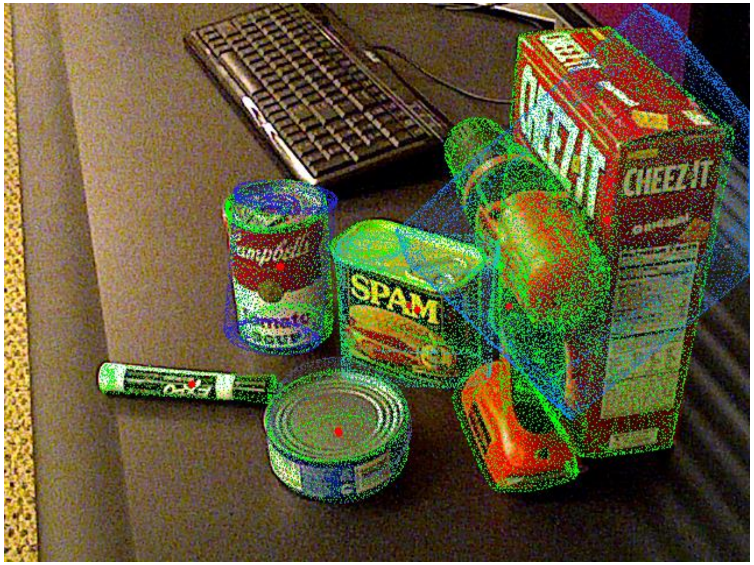
YOLOX-6D-Pose: test on YCBV

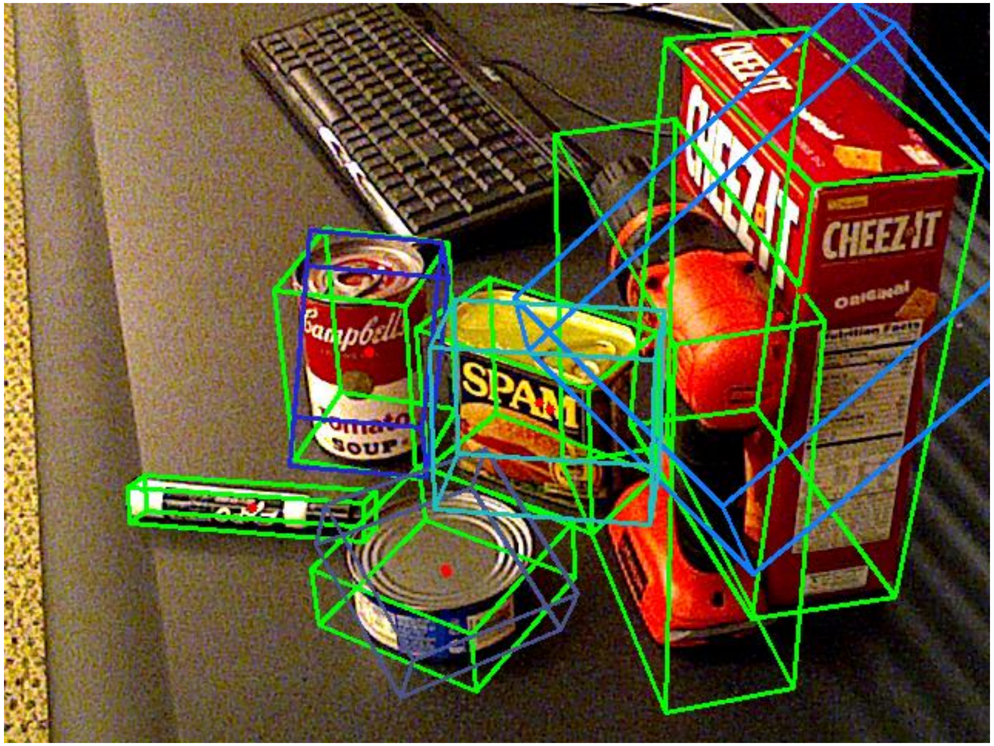
- Training Parameters:
  - max_epoch = 3 (for faster training)
  - batch_size = 16 or 32 (restrict by GPU memories)

Model	s	s-ti-lite	m	m-ti-lite	l	l-ti-lite
Inference Time	73.8ms	73.5ms	100.37ms	79.5ms	79.5ms	86.9ms
Speed	~13fps	~13fps	~10fps	~12.5fps	~12.5fps	~11.5fps
ADD-0.1	0.0567	0.0228	0.0679	0.0352	0.0673	0.0449
ADD-0.2	0.1626	0.0934	0.1954	0.1284	0.2391	0.1306
ADD-0.3	0.2518	0.1878	0.2836	0.2180	0.3652	0.1977
ADD-0.4	0.3313	0.2930	0.3826	0.2930	0.4594	0.2561
ADD-0.5	0.4265	0.3894	0.4760	0.3544	0.5492	0.3046

Result of yolox-l-object-pose







Outline

- Details of CNN-RNN Structure
- Experiments on YOLOX-6D-Pose Models
- Some Ideas
  - Encoding Pose or Image?
  - Model Pruning: Only 6D-Pose is Needed.
- Future Plan

Encoding Pose or Image?

- Inspiration from Xu Zhe's Head Pose Prediction Work: What about directly encode images?
  - My Proposed Method:

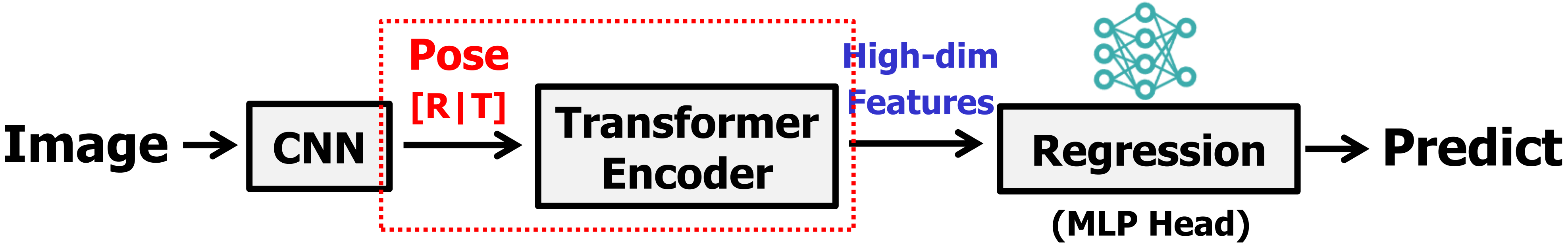


Image → CNN → Pose [R|T] → Transformer Encoder → High-dim Features → Regression (MLP Head) → Predict

  - Xu's TFSANet Method:

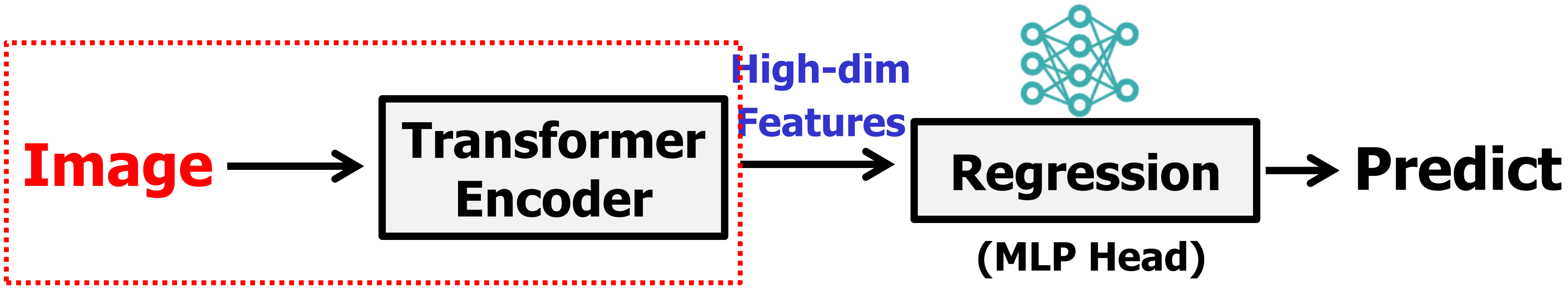
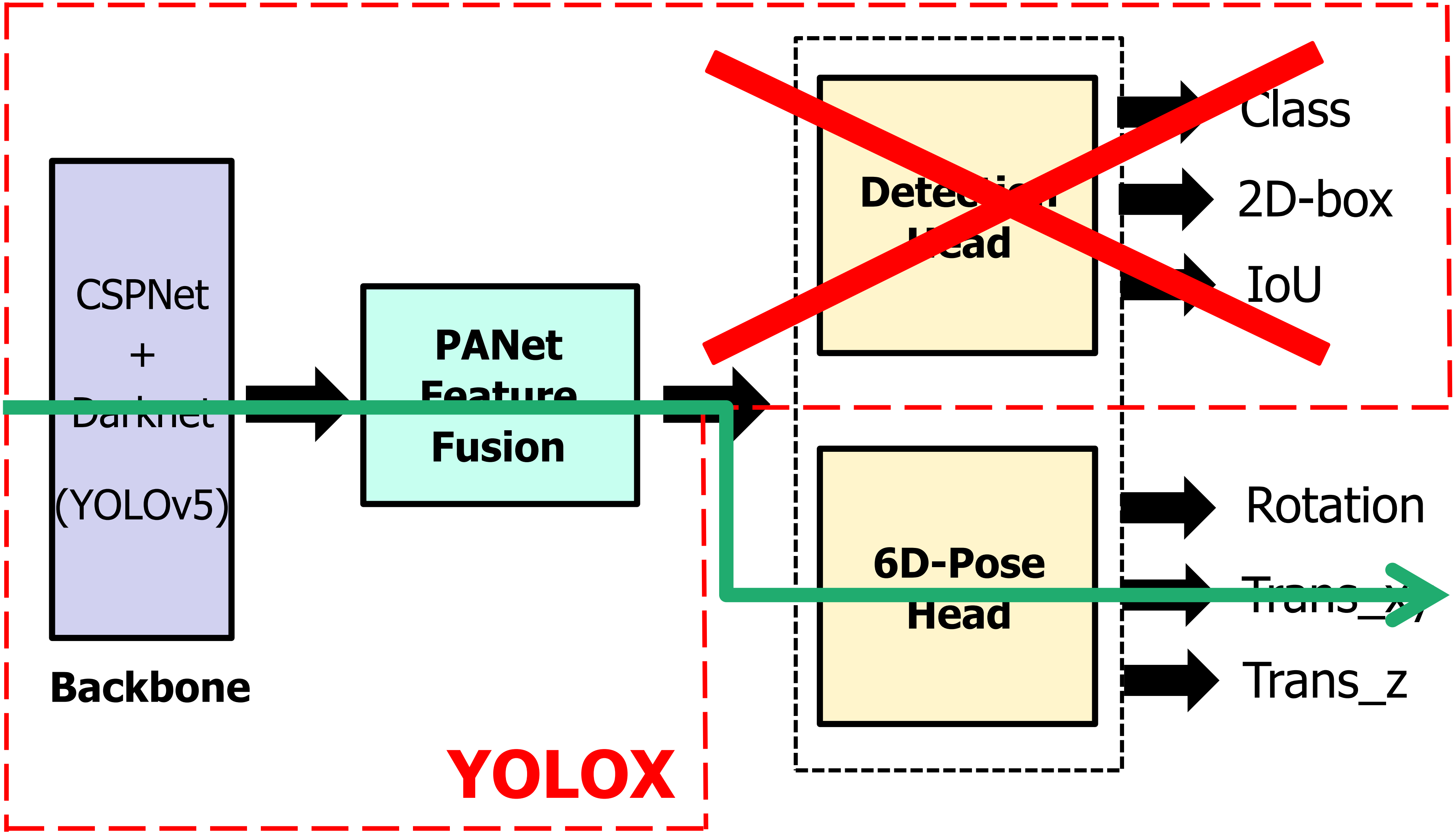


Image → Transformer Encoder → High-dim Features → Regression (MLP Head) → Predict

Model Pruning: Only 6D-Pose is Needed.

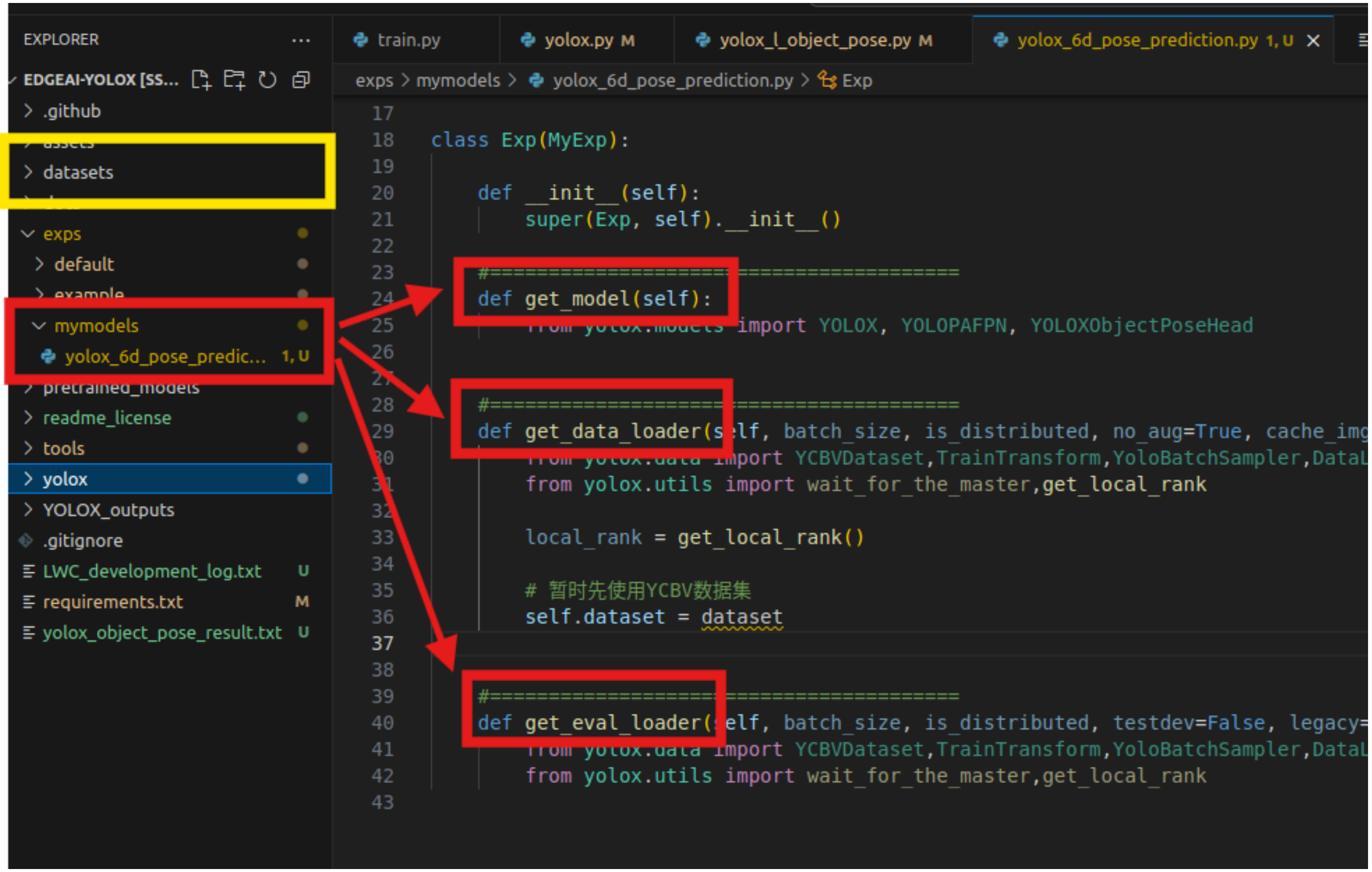


Outline

- Details of CNN-RNN Structure
- Experiments on YOLOX-6D-Pose Models
- Some Ideas
- Future Plan

Future Plan

- Try to train & test YOLOX-6D-Pose models with our own datasets, then evaluate their performance.
- Build up the prototype of the CNN-RNN Structure I proposed.



```

class Exp(MyExp):

    def __init__(self):
        super(Exp, self).__init__()

    #========================================
    def get_model(self):
        from yolox.models import YOLOX, YOLOPAFPN, YOLOXObjectPoseHead


    #========================================
    def get_data_loader(self, batch_size, is_distributed, no_aug=True, cache_img
        from yolox.data import YCBVDataset,TrainTransform,YoloBatchSampler,DataL
        from yolox.utils import wait_for_the_master,get_local_rank

        local_rank = get_local_rank()

        # 暂时先使用YCBV数据集
        self.dataset = dataset


    #========================================
    def get_eval_loader(self, batch_size, is_distributed, testdev=False, legacy=
        from yolox.data import YCBVDataset,TrainTransform,YoloBatchSampler,DataL
        from yolox.utils import wait_for_the_master,get_local_rank

```