

# YOLO vs ViT+DETR



let's compare yolo , vit+detr+temporal for posvoc data setss

Let's compare **YOLO** vs **ViT+DETR+Temporal Module** specifically for the **Pascal VOC-format lane cutting dataset** provided in the Gahan AI Hackathon. Here's a structured comparison across different dimensions that matter in this project context:

## 1. Task Context Overview

You're solving a **multi-stage** problem:

- **Stage 1:** Detect vehicles (object detection)
- **Stage 2:** Classify if an object is *cutting into the ego lane* (behavior classification)

## 2. Architecture Comparison

Feature	YOLO (v8/v5)	ViT + DETR + Temporal
Type	CNN-based one-stage detector	Transformer-based (ViT encoder + DETR decoder + LSTM/GRU/3D-CNN)
Input Type	Single frames	Multi-frame sequences (5-10 frames from same folder)
Temporal Awareness	✗ None	✓ Explicit via temporal module
Object Detection Accuracy	✓ High for real-time tasks	✓ Potentially better on rare/difficult cases via context
Lane Cutting Behavior Detection	✗ Needs a second model manually	✓ Easier to model spatial-temporal cues
Training Speed	⚡ Very fast	🐢 Slow due to transformer and multi-frame input
Inference Speed	⚡ Real-time ready	🐢 Slower; not suited for real-time without optimization
Implementation Complexity	● Easy to medium	● Complex (ViT + DETR + temporal fusion, needs track IDs)
Best Suited For	Baseline and object-level tasks	Behavior understanding across time (cut-ins, overtakes)

## 3. Dataset Compatibility

Factor	YOLO	ViT + DETR + Temporal
Pascal VOC (XML)	✓ Directly supported	✓ Requires preprocessing to COCO/JSON or conversion to tensors

Factor	YOLO	ViT + DETR + Temporal
Use of Sequences (Folder-wise)	✗ Only uses single frames	✓ Can use 5–10 frame temporal windows within folders
Object ID Tracking	✗ Needs extra (like Deep SORT)	✓ Naturally aligned with temporal tracking via embeddings

#### 4. Evaluation Goals in Hackathon Context

Criteria	YOLO	ViT + DETR + Temporal
Precision/Recall/F1 (50%)	High object-level, but may struggle with accurate cut-in classification	Likely better on nuanced behavioral cases via temporal patterns
Innovation (20%)	Moderate	High — multi-frame ViT+DETR+Temporal shows novelty
Code Quality (30%)	Easier to modularize and debug	Needs strong design but offers depth in final presentation

#### 5. Practical Recommendations

- **Baseline:** Train YOLOv8 on Pascal VOC images for vehicle detection. Add a separate shallow model (MLP or SVM) to classify lane-cutting based on bounding box movement and relative position change using Deep SORT IDs.
- **Advanced/Final Submission:** Train ViT encoder on 5-frame stacks, use DETR decoder for object query, and fuse outputs with RNN/3D CNN to classify lane-cutting. Use Deep SORT IDs to ensure track consistency across frames.

#### Summary

Category	YOLO	ViT + DETR + Temporal
Best for	Fast object detection	Behavioral pattern classification
Challenge Fit	Baseline model	Final polished innovative model
Use Temporal Data?	✗ No	✓ Yes
Inference Time	⚡ Fast	🐢 Slower
Difficulty Level	● Medium	● High

Let me know if you want:

- Full pipeline implementation (code) for either.
- Inference integration for the test set using `Submission.csv` format.
- Deep SORT + cut-in labeling pipeline.
- Training script and Docker setup for deployment.



what goes best for my kind of hackathon ?

Given your **Gahan AI hackathon** setup (48 hours, Pascal VOC dataset, lane-cutting behavior detection), here's a focused recommendation tailored to **win under constraints** of time, evaluation, and innovation:

## ✓ Best Strategy: Hybrid Approach

Use **YOLOv8 + Deep SORT + Lightweight Lane-Cutting Classifier**

### 🔥 Why it's best for this hackathon:

Reason	Justification
🕒 Time Constraint	YOLO is easy to train + Deep SORT is plug-and-play. You can focus on feature engineering for behavior.
🧠 Accuracy	YOLOv8 offers high precision for object detection (main metric = 50%). Deep SORT gives tracking data to infer lane cutting.
💡 Innovation (20%)	Adding a temporal lane-cutting behavior classifier (e.g., using bounding box deltas or ego-relative distance) brings uniqueness.
📦 Output Format Fit	YOLO + Deep SORT output can be directly converted to <b>Submission.csv</b> (recording, frame, class, bbox, cutting).
👨‍💻 Code Quality & Demo	Easy to modularize (object detection → tracking → behavior classification). Ideal for a clear demo.

## 💡 Architecture Flow

```
mathematica

Input Image Sequence (from folder)
  |
  YOLOv8
  |
  Detected Vehicles (bbox, class)
  |
  Deep SORT (Tracking IDs)
  |
  Frame-to-frame tracking of vehicle positions
  |
  Lightweight Classifier (XGBoost/MLP/LSTM)
  |
  Predict 'cutting = True/False'
  |
  Write to Submission.csv
```

## 🛠️ Tools and Models

Component	Tool / Model	Notes
Object Detection	YOLOv8 (pretrained + finetune)	Pascal VOC-friendly

Component	Tool / Model	Notes
Object Tracking	Deep SORT	Associates IDs across frames
Lane Cutting Classifier	Logistic Regression / MLP / LSTM	Use bounding box deltas + ego position
Data Format Conversion	Python (XML → usable CSV)	You'll need pre-processing
Final Inference	YOLO + Deep SORT + Classifier	Generates <b>Submission.csv</b>

## Feature Ideas for Lane Cutting Classifier

Extract per vehicle track:

- Distance from ego lane center
- Slope change over time
- Lateral velocity ( $\Delta x / \Delta t$ )
- Time-to-collision estimate
- Distance moved across lane lines (if detectable)

You can calculate these heuristically from bounding boxes across 5–10 frames.

## ViT + DETR + Temporal?

 **Avoid for hackathon** unless:

- You **already have working training code + preprocessed data**
- You have **TPU/GPU ready from the start**
- Your team has a **dedicated member for pure model dev**

Otherwise, transformer-based pipelines are *risky and time-expensive* under 48-hour constraints.

## Final Verdict:

Use **YOLOv8 + Deep SORT + temporal behavior classifier**. It gives you the best trade-off between **performance, innovation, speed of execution, and alignment with hackathon scoring**.

If you'd like, I'll now:

- Generate full file structure
- Implement modular code templates
- Show **Submission.csv** generation

Would you like to proceed with that?



It's 5 team member , 48 hours hackathon with 50k inr prize pool

Perfect. With **5 team members, 48 hours**, and a decent **₹50k prize pool**, your **goal should be to dominate both the performance and innovation criteria** — not just "complete it".

Here's a **strategic breakdown** tailored to your setup:

## 🎯 Strategy to Win: Tiered Implementation Plan

Team Member Role	Focus Area (Core)	Goal by 36–40 Hr Mark
🧠 <b>Model Lead</b>	ViT + DETR + Temporal (Track 2)	Complete multi-frame model + eval
⚙️ <b>Engineer #2</b>	YOLOv8 + Deep SORT (Track 1 - Baseline)	Clean inference + CSV generation
🔍 <b>Classifier Expert</b>	Lane cutting behavior classifier (MLP/LSTM using bbox tracks)	Feature design + model output
🔧 <b>DevOps &amp; Infra</b>	Dataset conversion, training infra, colab/GPU/cloud setup	Setup, conversion & integration
🎤 <b>Presenter / QA</b>	Real-time debugging, visuals, video demo, README, final pitch	Submission polish & demo ready

## 🏆 Optimal Architecture: Hybrid Dual-Track

### ♦ Track 1 (Safe Baseline): YOLOv8 + Deep SORT + MLP

- Trains fast
- Great for evaluation
- Reliable Submission.csv output





### ♦ Track 2 (Innovative Shot): ViT (spatial) + DETR (decoder) + LSTM/3D CNN (temporal)

- Loads 5–10 frames per object track
- Classifies lane cutting based on **motion across time**
- Gives you **edge in innovation (20%)** + maybe F1 if tuned

If Track 2 underperforms, fall back to Track 1 for submission.

## 🧠 How to Impress Judges

Criteria	How to Nail It
🎯 Accuracy (50%)	Use YOLOv8 trained on Pascal VOC + Deep SORT + classifier (YOLO handles static accuracy)

Criteria	How to Nail It
 Innovation (20%)	ViT + DETR + LSTM module: novelty in transformer + temporal behavior classification
 Code Quality (30%)	Modular repo, clean config files, README, dockerized/notebook-based training and eval
 Final Presentation	Use a 60–90 sec video showing: live frame sequence → prediction → highlight of cut-in
 Submission.csv	Must strictly follow the frame order and format – automate with script!

## Technical Stack Suggestion

Component	Tool / Library
Object Detection	<a href="#">YOLOv8</a> (Ultralytics)
Tracking	<a href="#">Deep SORT</a>
Temporal Model	PyTorch, huggingface <b>ViT</b> , custom LSTM head
Dataset Tools	Python (Pascal VOC XML parser), Albumentations
Inference Engine	Custom Python scripts with modular prediction pipeline
Visualization	OpenCV (for video preview), Matplotlib
Submission Prep	Script: XML + model output → CSV

## Timeline Suggestion (High Impact Plan)

### Day 1 (0–12 hrs)

- Setup dataset preprocessing
- Start YOLOv8 + Deep SORT
- Convert VOC to COCO if needed for DETR
- ViT + DETR skeleton setup

### Day 2 (12–36 hrs)

- Train YOLO, generate first outputs
- Deep SORT ID integration
- Extract tracks → behavior features
- Train MLP/LSTM classifier
- Start ViT temporal model training

### Final Phase (36–48 hrs)

- Compare YOLO vs ViT model output
- Evaluate lane-cutting classifier performance
- Finalize Submission.csv
- Create a demo video + README + presentation

- Submit

## Conclusion

With 5 people and 48 hours, **you can afford to build both performance and novelty**. Treat YOLO+SORT as your “anchor” and the ViT+Temporal stack as your “swing for the fences.”

I can now:

- Generate directory structure
- Scaffold all modules (`detector/`, `tracker/`, `classifier/`, `inference/`)
- Provide starter code for all

Would you like me to generate this full working structure for you now?



how to submit a quota increase request on google cloud

To **submit a quota increase request on Google Cloud**, follow these steps:

## Step-by-Step: Submit Quota Increase Request on GCP

### ✓ 1. Go to the Quotas Page

- Open: <https://console.cloud.google.com/iam-admin/quotas>
- Select the **project** where you want to increase the quota (top nav bar)

### 2. Filter the Quota You Need

Use filters like:

- **Service:** e.g., “Vertex AI API”, “Compute Engine API”, “Cloud Storage”
- **Metric:** e.g., “GPUs (all regions)”, “NVIDIA T4”, “Vertex AI quota units”
- **Region:** e.g., `us-central1`, `asia-south1`, etc.

### 3. Select Quotas

- Tick checkboxes next to the **specific quota metrics** you want to request an increase for
  - For example: `NVIDIA T4 GPUs`, `Requests per minute per user`, `Vertex AI quota units`, etc.

### 4. Click "Edit Quotas"

- Top of the page → Click **Edit Quotas**
- A **sidebar form** will appear

## 5. Fill in Request Form

- Enter:
  - **Contact info** (your email)
  - **New quota limit** (be reasonable — mention what your usage case is)
  - **Business justification:** e.g.,

"We are participating in a 48-hour AI Hackathon using Vertex AI for vision-based sequence modeling. We need 8 T4 GPUs for model training and benchmarking."

## 6. Submit the Request

- Click **Submit** and wait for email confirmation
- Google typically replies within a few hours to 1 day

## Tips for Getting Approved Fast

Tip	Why
Be specific with your <b>use case</b>	Hackathon, MVP, educational project
Ask for <b>realistic quotas</b> first	Don't request 100 GPUs without prior usage
Use <b>institutional email</b> if possible	Improves credibility
Mention <b>time sensitivity</b>	"Hackathon deadline on 15 June, 3PM IST"

If you'd like, I can help you **draft the quota request message** for your hackathon use case. Just tell me:

- Region (e.g., **asia-south1**)
- GPU type (e.g., **T4, A100**)
- Which GCP service you're using (**Vertex AI, Compute Engine**, etc.)

Would you like me to prepare that template for you?