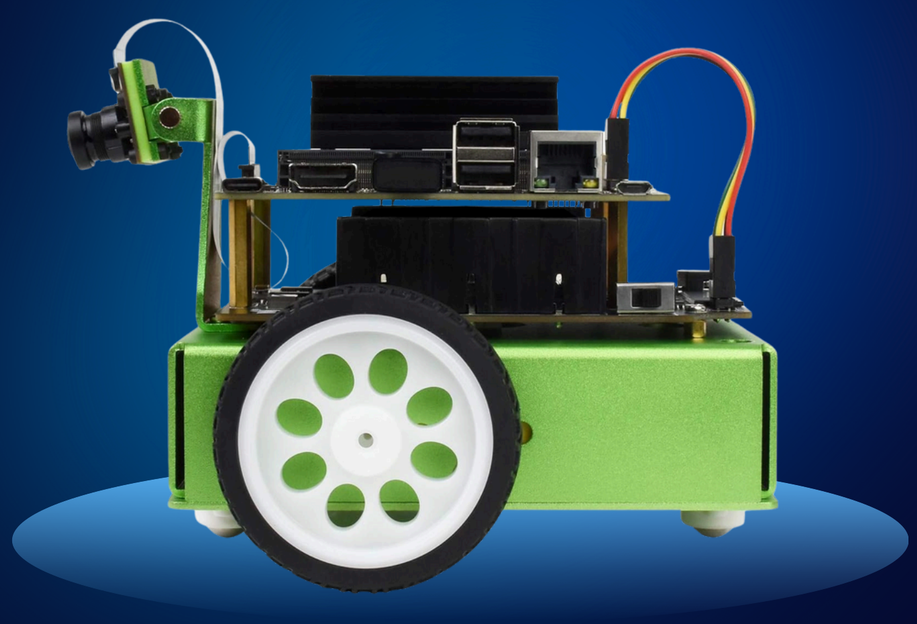
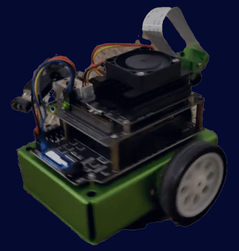
Imitation Learning for Path Following Using NVIDIA JetBot

Master in Intelligent Robotic Systems — University of Girona

Universitat de Girona

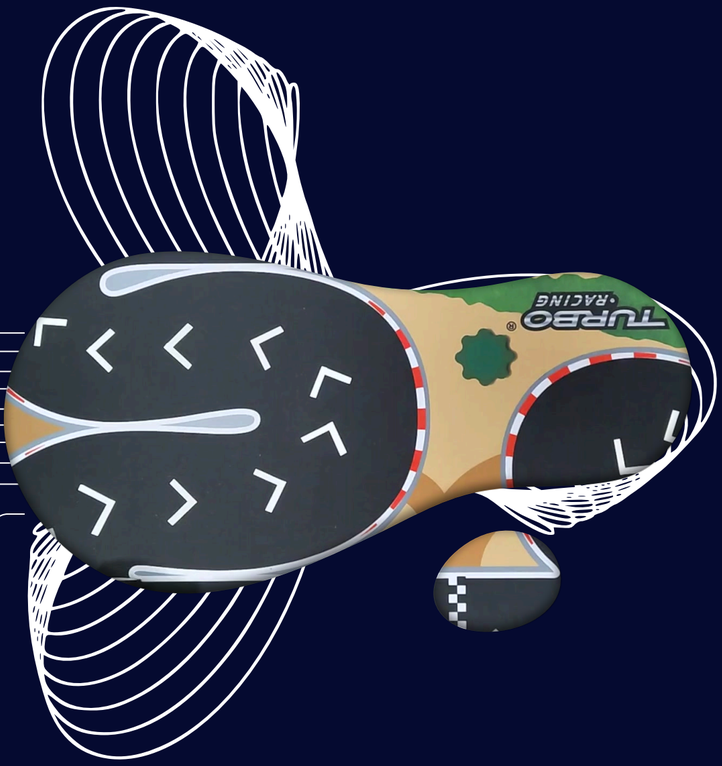


Presented By: Adel saidani - Enis Hedri - Mahra Alhosani

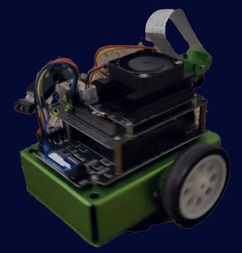


PRESENTATION OUTLINE

1. Motivation & Objective
2. JetBot Platform & Setup
3. Approach: Behavioral Cloning
4. Data Collection & Preprocessing
5. Model Architecture
6. Iterative Development & DAgger
7. Results
8. Live Demo
9. Conclusion & Future Work



MOTIVATION & OBJECTIVE



❌ Traditional approaches need hand-crafted rules

❌ Reinforcement learning requires expensive exploration

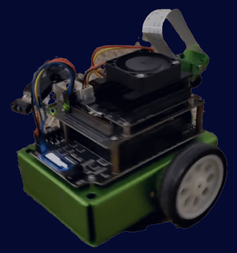
❌ Jetson Nano has limited compute for complex algorithms

✅ Learn directly from human demonstrations

✅ Simple supervised learning approach

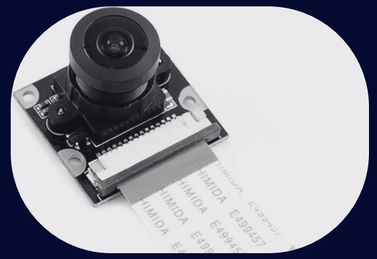
✅ Efficient real-time inference on embedded hardware

🎯 Goal: Enable JetBot to autonomously follow a track using only RGB camera input — predicting both steering and speed from a single image

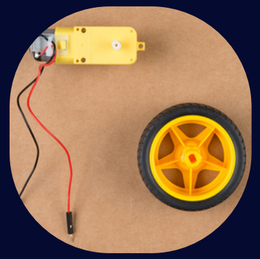


JETBOT PLATFORM & SETUP

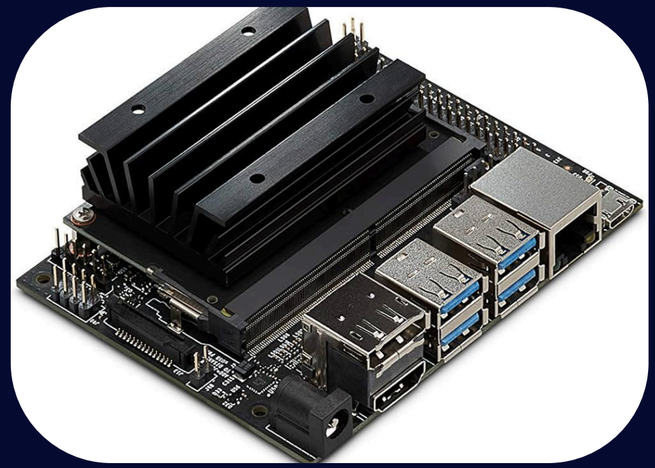
RC CAR CIRCUIT



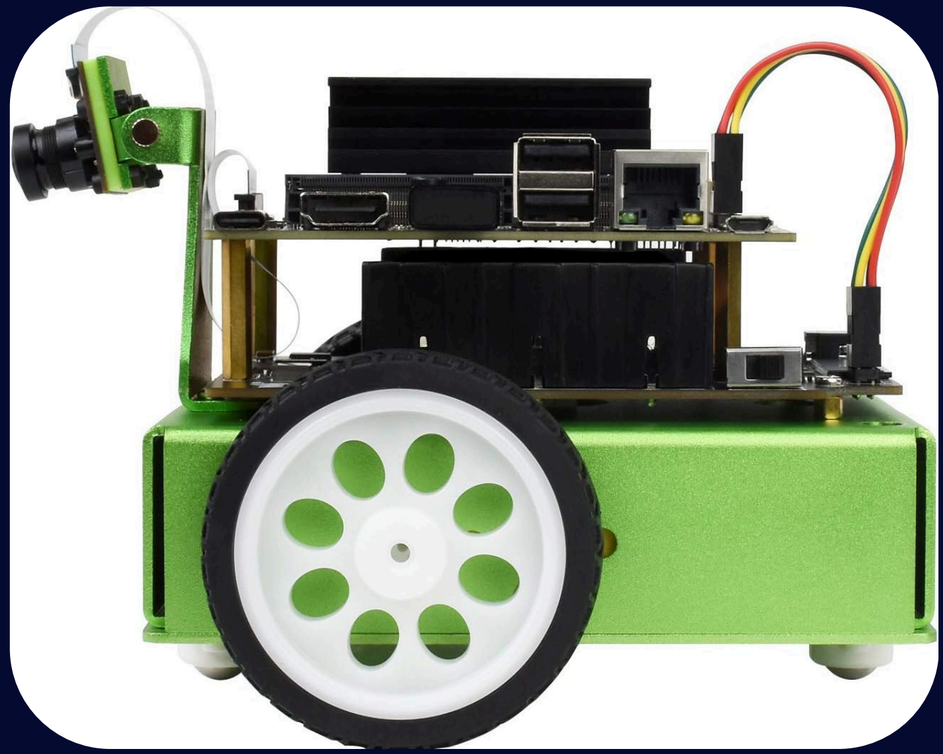
IMX219 CAMERA



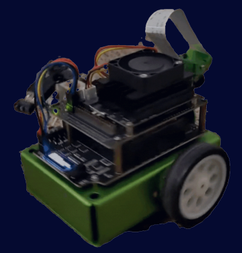
DC MOTORS



JETSON NANO



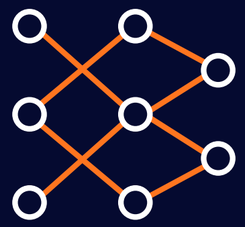
FULL SETUP



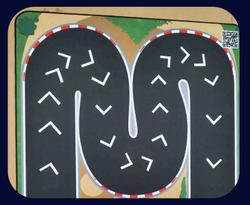
APPROACH – BEHAVIOR CLONING

Human expert
(Drives)

Record Data



Train CNN



Deploy

Observation (State)	Actions
RGB image (224×224×3)	Steering: [-1, +1]
Single frame	Speed factor: [0, 1]

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left[(steering_{pred} - steering_{true})^2 + (speed_{pred} - speed_{true})^2\right]$$

NO REWARD FUNCTION

NO SIMULATION

DATA COLLECTION

Human expert
(Drives)

Analog Steering [-1, +1]

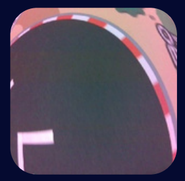
Start moving and record

Speed factor [0,1]

Final Dataset

- ~11,000 images
- 20 Hz recording rate
- 80/20 -- train/val split

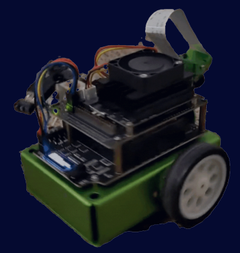
Filename encoding example:



1767136021235_-0.696_0.000.jpg

Timestamp_steering_speed.jpg

1767136016217_0.000_1.000.jpg



PREPROCESSING & AUGMENTATION

Raw Image
(640 x 480)

Cropped Image

Resized + Normalized
(224 x 224)

Crop Settings:

- Top: 20%
- Bottom: 0%
- Left: 8%
- Right: 12%

🎯 REMOVE IRRELEVANT REGIONS (DISTANT OBJECTS) AND FOCUS ON THE ROAD IMMEDIATELY AHEAD

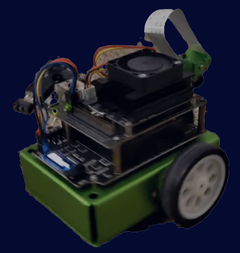
Normalization in every channel:

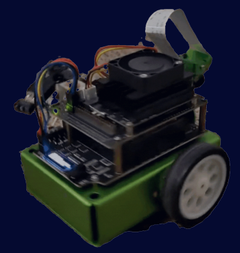
- IMAGENET_MEAN = [0.485, 0.456, 0.406]
- IMAGENET_STD = [0.229, 0.224, 0.225]

🎯 NORMALIZE USING IMAGENET STATISTICS — THESE SPECIFIC MEAN AND STANDARD DEVIATION VALUES PER RGB CHANNEL.

Color Jitter Augmentation:

Brightness	±30%
Contrast	±30%
Saturation	±30%
Hue	±30%





PREPROCESSING & AUGMENTATION

Raw Image
(640 x 480)

Cropped Image

Resized + Normalized
(224 x 224)

Crop Settings:

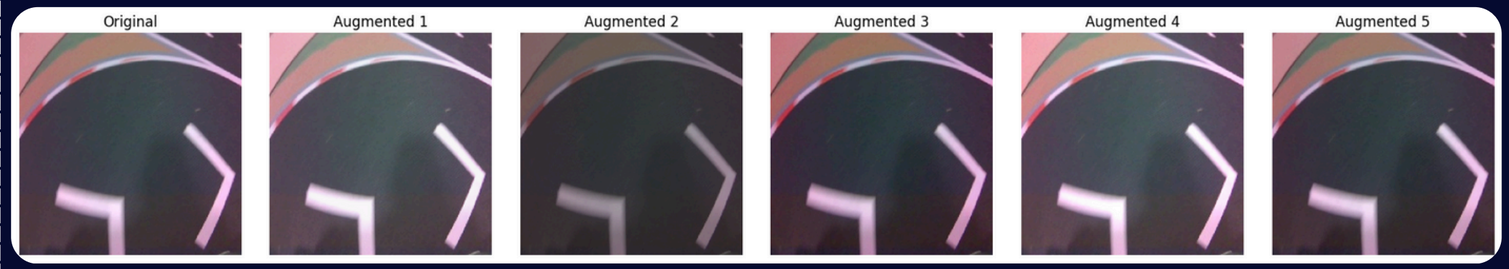
- Top: 20%
- Bottom: 0%
- Left: 8%
- Right: 12%

🎯 REMOVE IRRELEVANT REGIONS (DISTANT OBJECTS) AND FOCUS ON THE ROAD IMMEDIATELY AHEAD

Normalization in every channel:

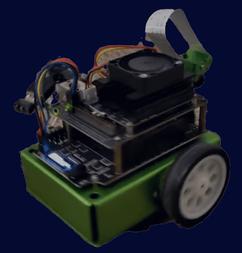
- IMAGENET_MEAN = [0.485, 0.456, 0.406]
- IMAGENET_STD = [0.229, 0.224, 0.225]

🎯 NORMALIZE USING IMAGENET STATISTICS — THESE SPECIFIC MEAN AND STANDARD DEVIATION VALUES PER RGB CHANNEL.



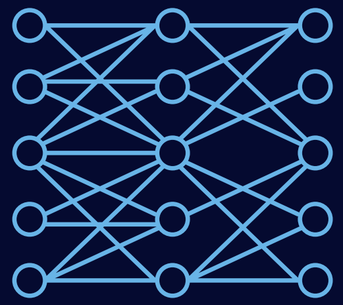
🎨 Color Jitter Augmentation:

Brightness	±30%
Contrast	±30%
Saturation	±30%
Hue	±30%

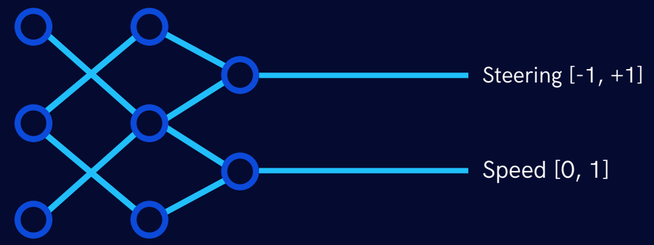


NEURAL NETWORK ARCHITECTURE

Input Image
(224×224×3)



ResNet18 Backbone
(pretrained)



FC Layer
512→2

🧠 ResNet18 (Pretrained on ImageNet)

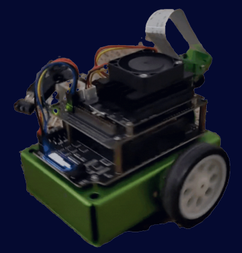
- 18 layers with residual connections
- Extracts visual features (edges, textures, shapes)
- Final classification layer removed

🔧 Custom Output Head

- Fully connected: 512 → 2 neurons
- Output 1: Steering angle
- Output 2: Speed factor

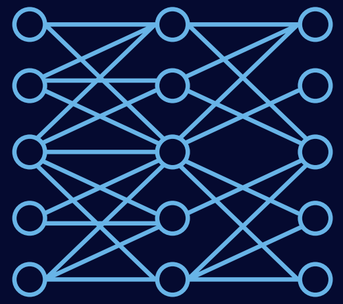
📊 Model Stats

- Total parameters 11.8M
- Trainable 11.8M
- Input size 224x224x3
- Output size 2

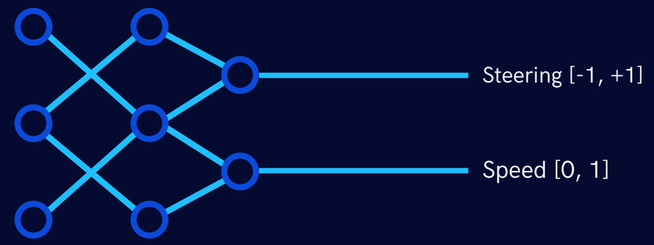


NEURAL NETWORK ARCHITECTURE

Input Image
(224×224×3)



ResNet18 Backbone
(pretrained)



FC Layer
512→2

🧠 ResNet18 (Pretrained on ImageNet)

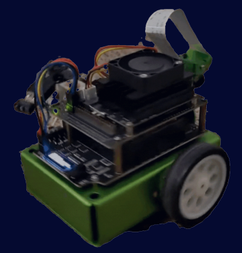
- 18 layers with residual connections
- Extracts visual features (edges, textures, shapes)
- Final classification layer removed

🔧 Custom Output Head

- Fully connected: 512 → 2 neurons
- Output 1: Steering angle
- Output 2: Speed factor

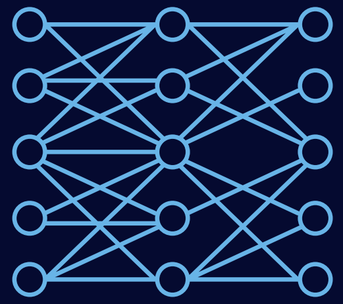
📊 Model Stats

- Total parameters 11.8M
- Trainable 11.8M
- Input size 224x224x3
- Output size 2

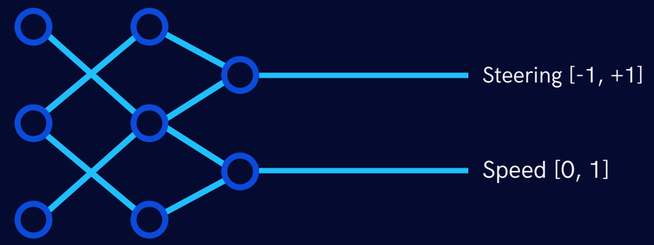


NEURAL NETWORK ARCHITECTURE

Input Image
(224×224×3)



ResNet18 Backbone
(pretrained)



FC Layer
512→2

🧠 ResNet18 (Pretrained on ImageNet)

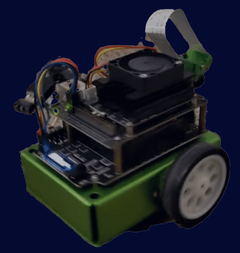
- 18 layers with residual connections
- Extracts visual features (edges, textures, shapes)
- Final classification layer removed

🔧 Custom Output Head

- Fully connected: 512 → 2 neurons
- Output 1: Steering angle
- Output 2: Speed factor

📊 Model Stats

- Total parameters 11.8M
- Trainable 11.8M
- Input size 224x224x3
- Output size 2



TRAINING SETUP

⚙️ Hyperparameters

Optimizer	Adam
Learning Rate	1e-4
Batch Size	8
Weight Decay	1e-5
Max Epochs	50
Early Stopping	Patience = 7

📉 Loss Function

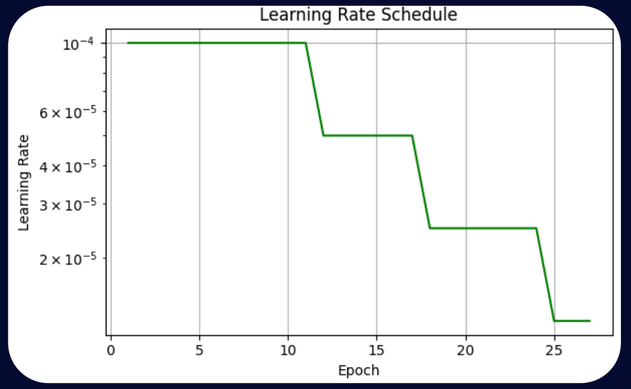
$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left[(steering_{pred} - steering_{true})^2 + (speed_{pred} - speed_{true})^2\right]$$

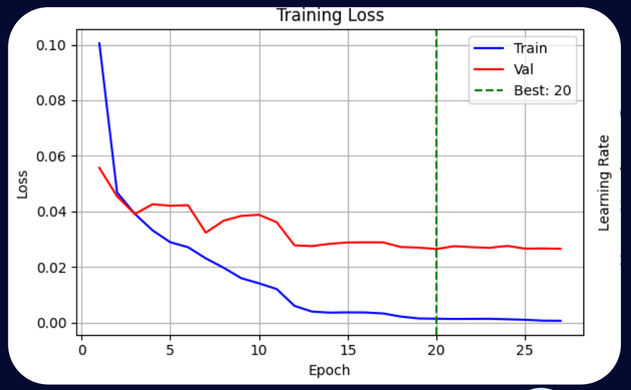
📈 LR Scheduler

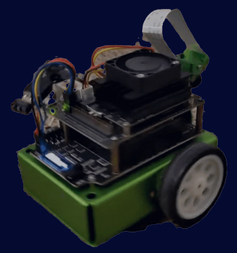
- Reduce by 0.5×
- After 3 plateau epochs

💻 Training Environment

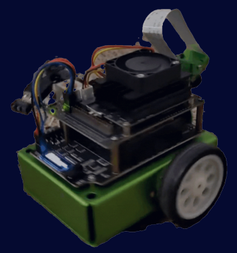
- PC with GPU (not on JetBot)
- Training time: ~35 minutes
- Model transferred to JetBot for deployment





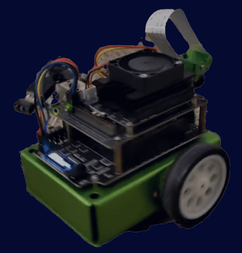


ITERATIVE DEVELOPMENT AND DAGGER



ITERATIVE DEVELOPMENT AND DAGGER

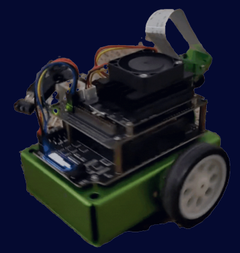
GRAYSCALE IMAGE + DIGITAL + SINGLE FRAME



ITERATIVE DEVELOPMENT AND DAGGER

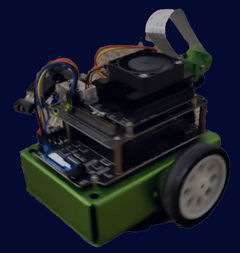
GRAYSCALE IMAGE + DIGITAL + SINGLE FRAME

GRAYSCALE + ANALOG + SINGLE FRAME

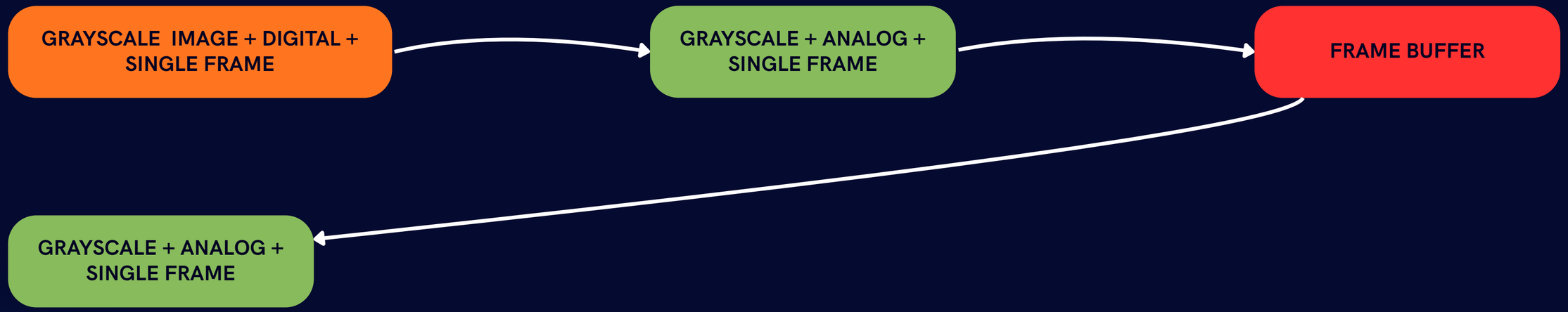


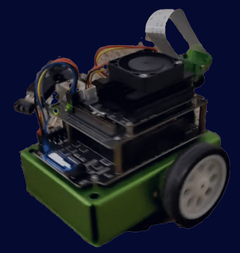
ITERATIVE DEVELOPMENT AND DAGGER

GRAYSCALE IMAGE + DIGITAL + SINGLE FRAME → GRAYSCALE + ANALOG + SINGLE FRAME → FRAME BUFFER

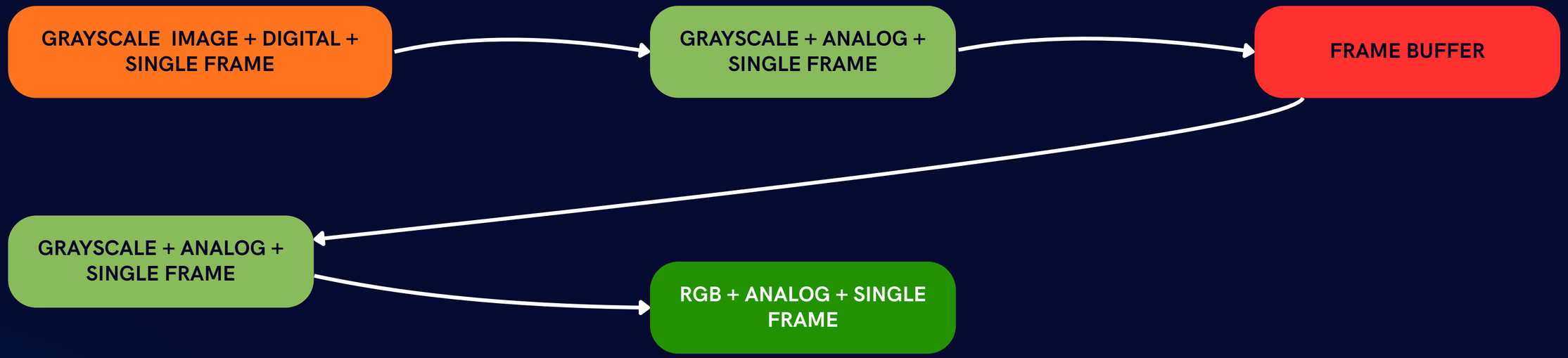


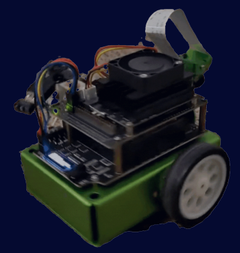
ITERATIVE DEVELOPMENT AND DAGGER



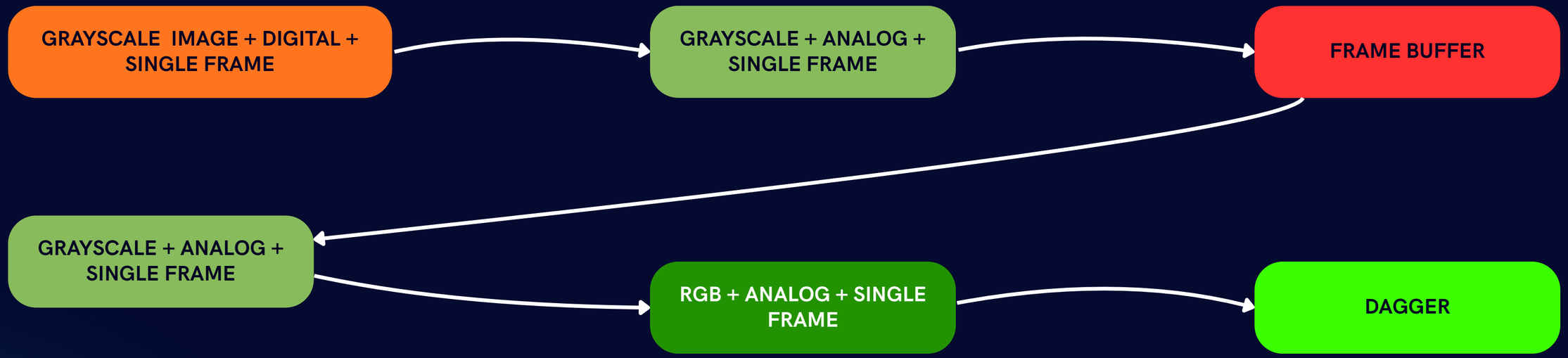


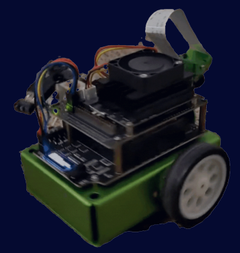
ITERATIVE DEVELOPMENT AND DAGGER





ITERATIVE DEVELOPMENT AND DAGGER





ITERATIVE DEVELOPMENT AND DAGGER

GRAYSCALE IMAGE + DIGITAL + SINGLE FRAME

GRAYSCALE + ANALOG + SINGLE FRAME

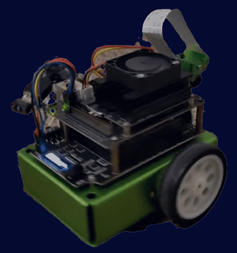
FRAME BUFFER

GRAYSCALE + ANALOG + SINGLE FRAME

RGB + ANALOG + SINGLE FRAME

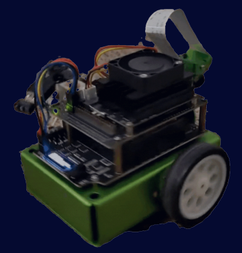
DAGGER

RGB + ANALOG + SINGLE FRAME + DAGGER + ADDING SPEED

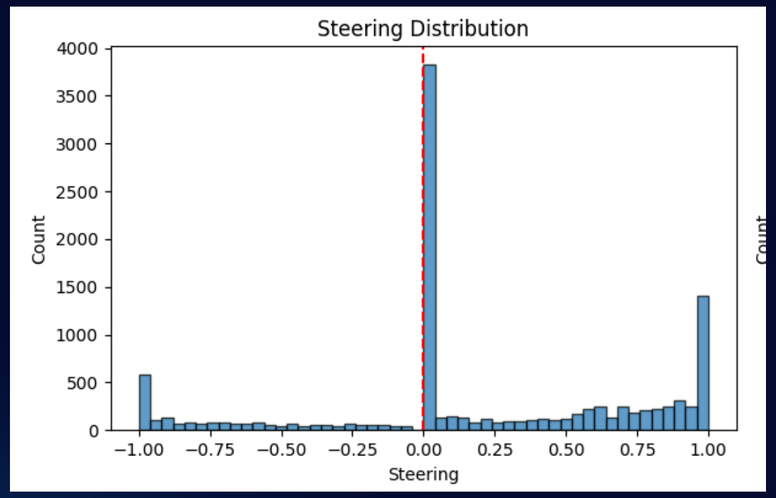


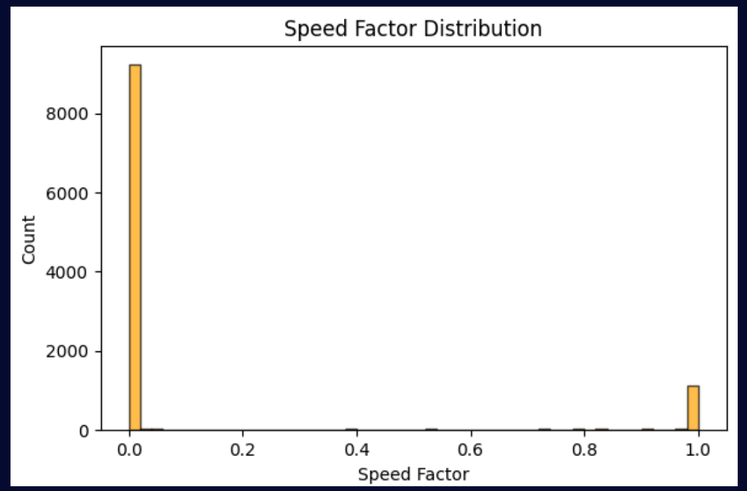
DAGGER – FIXING CORNER FAILURES

1. **Problem:** Initial model worked fine… except some corners
2. **Solution:** DAgger Run model on JetBot
   → when it fails, interrupt
   → take manual control
   → collect corrections
3. **Retrain:** Merge new data with original dataset
   → train again

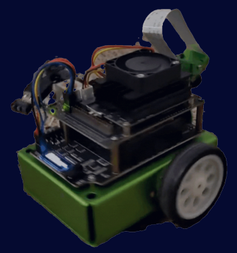


RESULTS — TRAINING PLOTS

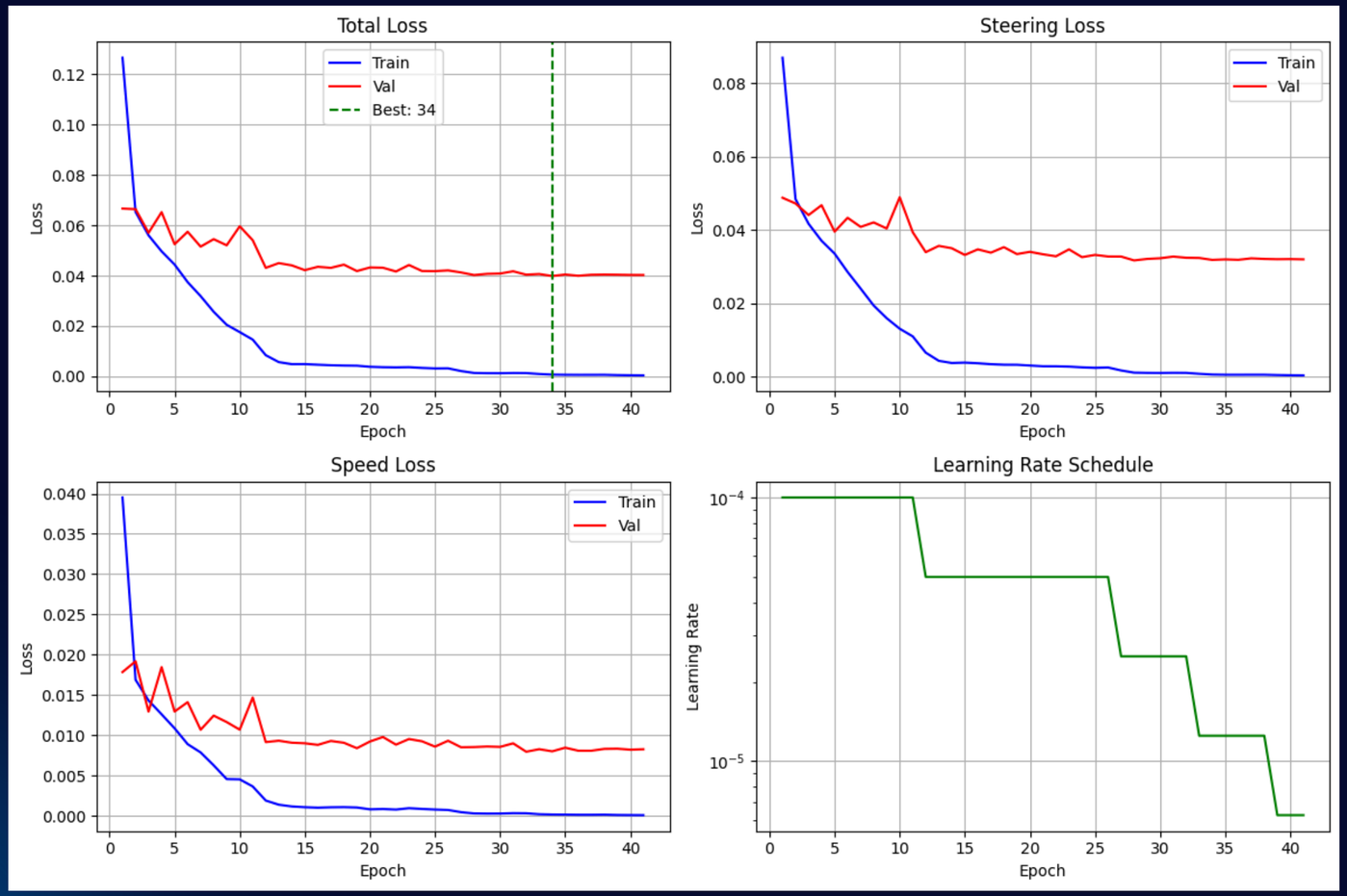




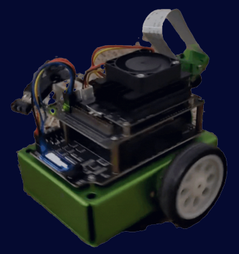
Images	11K
Epochs	34
	35 min
MSE speed	0.0318
MSE steeering	0.0080



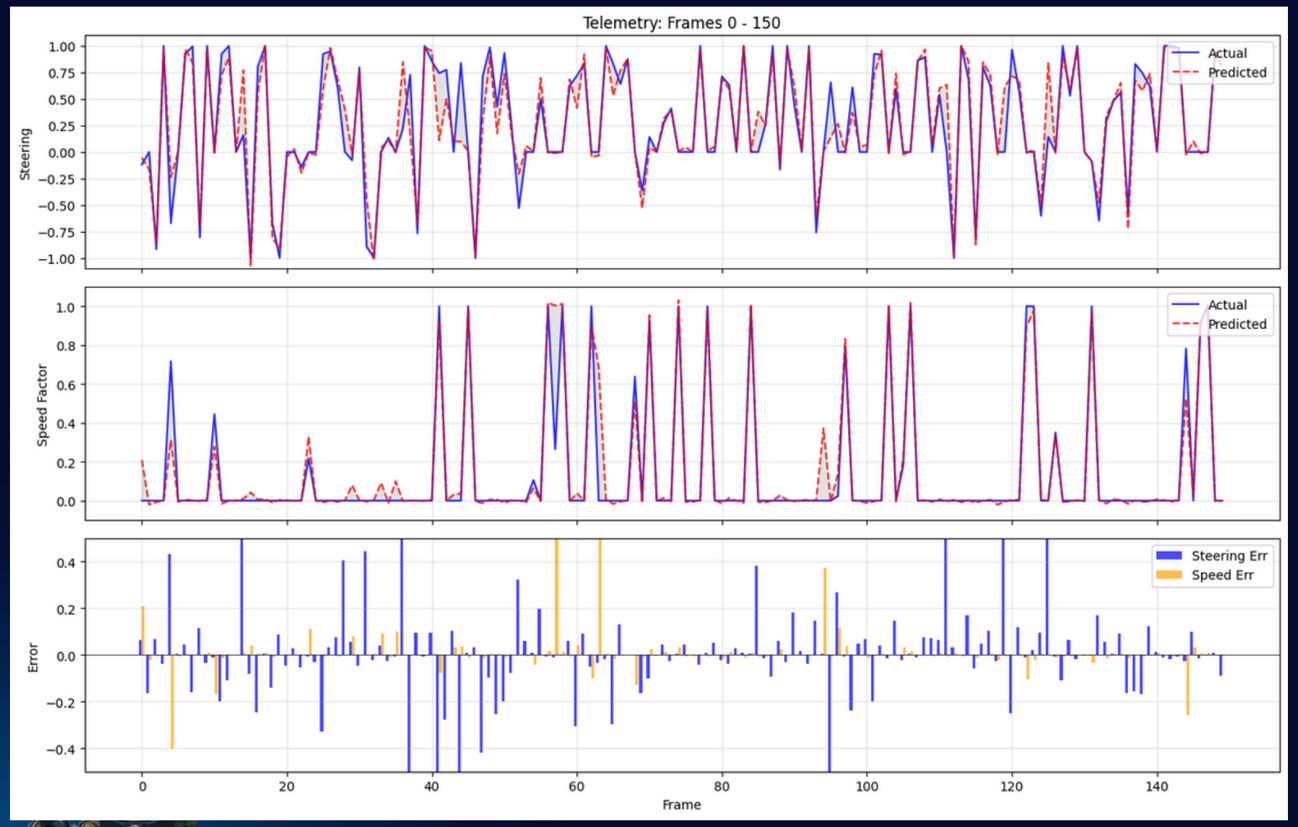
RESULTS — TRAINING PLOTS

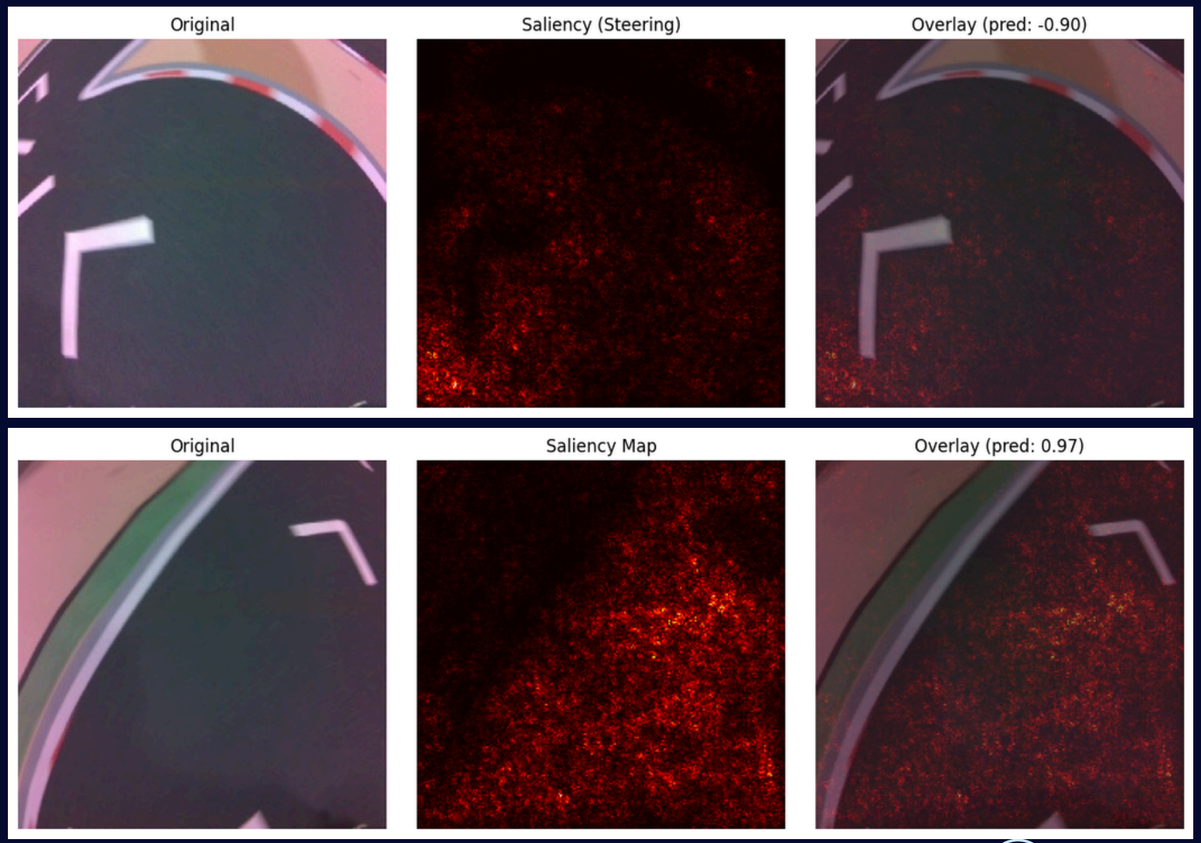


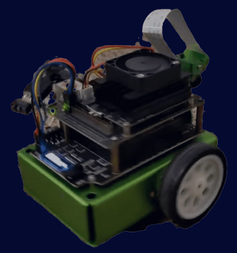
Images	11K
Epochs	34
	35 min
MSE speed	0.0318
MSE steeering	0.0080



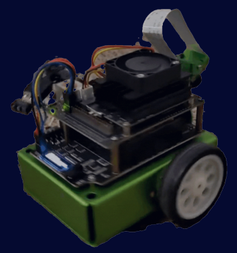
TRAINING SETUP







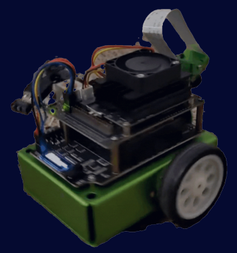
LIVE DEMO



CONCLUSION AND FUTURE WORK

- Behavioral cloning works ~11,000 images + 35 min training
- Preprocessing is crucial Cropping + augmentation = **generalization**
- **DAgger fixes edge:** Small data addition
  → big real-world improvement
- Single frame is enough
- Dual output enables adaptive speed Slow in corners, fast on straights

- **Generalize to new tracks, different lighting**
- **Obstacle detection and avoidance**
- **Join track again after driving error**



Thank You