Turn Signal Dataset Labeling (Classification) with Fine-Tuned ResNet-18, OpenAI API & Heuristics

Elizabeth Ann Dwenger
Institute of Computer Science
University of Tartu
Tartu, Estonia
elizabeth.ann.dwenger@ut.ee

***Abstract*—This report documents the development of a pipeline for detecting and classifying vehicle turn signals from dashcam footage provided by the Autonomous Driving Lab (ADL) at the University of Tartu. The work encompasses 8,790,531 total car crop images collected from YOLO crops that were produced by ADL, with a focus on both single-frame classification and temporal sequence analysis. The pipeline progresses through multiple stages: initial dataset cleaning and filtering, front/back vehicle classification, temporal tracking using ByteTrack and DeepSORT, manual annotation of light states, foundation model testing and algorithm creation. The results highlight the limitations of current foundation models for temporal perception tasks and demonstrate a more classical approach which outperforms the foundation models. Repository can be found at github.com/elizabeth-dwenger/Turn-Signal-Detector-Autonomous-Driving-Lab.**

***Index Terms*—Object tracking, Temporal classification, ResNet-18, Turn signal detection, Autonomous driving**

I. Introduction

Modern autonomous driving systems typically employ modular architectures, where distinct subsystems handle perception, prediction, planning, and control. In contrast, end-to-end self-driving approaches use a single neural network that directly maps raw sensor inputs (such as camera images) to vehicle control outputs (steering angle, acceleration, braking). While end-to-end methods have shown promise in research settings, they remain challenging due to several factors. Training these networks with imitation learning can lead to unpredictable generalization, as rare or unseen scenarios, like stopping for uncommon animals or reacting to new traffic rules, may not be correctly handled. Additionally, end-to-end networks often require vast amounts of labeled data and auxiliary supervision to shape their internal representations, meaning they are not truly "free from handcrafted knowledge." Consequently, modular perception systems, which decompose the driving task into interpretable components, are currently preferred for production autonomous vehicles.

Vehicle turn signal detection represents one such critical perception component that is specifically well-suited to computer vision approaches. By detecting turn signals (left, right, both, none), autonomous systems can predict other vehicles' intentions and make more informed decisions in traffic scenarios. Examples of this include a car pulling into traffic from a parked position or turning off of a roundabout. This work attempts to addresses the challenge of reliably annotating these light states from real-world dashcam footage collected during mapping runs in Tartu, Estonia in order to create a dataset of these light states.

The problem presents several technical challenges:

- Scale variation: Vehicles appear at varying distances and angles
- Lighting conditions: Varying weather, time of day, and ambient lighting
- Occlusion: Partial vehicle visibility due to traffic and environmental factors
- Temporal consistency: Turn signals are inherently temporal events, and it was determined that sequence analysis is required

My approach combines classical computer vision techniques (the classification of the turn signal) with deep learning methods (ResNet-18 and DeepSort), progressing from single-frame classification to temporal sequence modeling.

A. Data Collection

The dataset consists of dashcam footage collected from autonomous vehicle mapping runs in Tartu, Estonia, conducted throughout 2024. Videos were captured using multiple camera configurations including narrow front, wide front, and side-facing cameras at 10 FPS.

B. Dataset Scale

As of September 2025, the dataset contains:

- Total car crops (images): 8,790,531
- Unique video sequences: 274
- Time coverage: April 2024 – September 2024
- Geographic coverage: Streets of Tartu, Estonia

C. Detection Pipeline

Car crops were generated using YOLOv11 object detection applied to raw video frames. For each detection, the result includes:

- Crop image (isolated vehicle region)
- Full frame image with detection context
- Bounding box coordinates (YOLO format: normalized center x, y, width, height)
- Detection confidence score
- Frame ID and sequence identifier

While some verions of YOLO allow object tracking, this was not done on these video frames.

II. Directory Structure

The project follows a hierarchical directory structure. While the images are stored on the high-performance computing cluster of the University of Tartu, all scripts can be found on the GitHub.

A. Directory Components

The images are stored on the high-performance computing (HPC) cluster's shared filesystem. The path to the images is `gpfs/space/projects/ml2024/2024-{date}_mapping_tartu_streets/camera_{type}/predict/crops/car/{frame_id}{crop_index}.jpg`, which contains cropped vehicle images extracted from detections. Filenames follow the pattern `{frame_id}{crop_index}.jpg`, where crop_index is omitted for the first detection (line 0), "2" for the second detection (line 1), "3" for the third detection (line 2), etc (this is important to know for sequencing the images using DeepSort).

Under the path `predict/labels/` is stored YOLO format detection labels with one text file per frame.

Under the path `predict/images/` is stored full frame images corresponding to detections. This means the full video frame, not the cropped image (of the cars here).

Locally, under `scripts/` and `sampled_images/` are stored both the processing and training scripts, as well as the manually annotated images used for ResNet-18 training.

III. Single Image Classification

A. Dataset Cleaning

1) Image Filtering: Initial dataset filtering was performed in order to only retain those images which were at least of the size 50x50 pixels and had a width that was greater than its height (as most cars should adhere to this). This was done using shell commands (can be found in the GitHub).

2) Results: Running this shell command took around 1.5 days, and produced the following results:

Input images: 8,790,531
Filtered images: 4,731,786
Reduction rate: 46.18%

B. Front/Back Classification using ResNet-18

A binary classifier was trained to distinguish between front-facing and rear-facing vehicle crops.

1) Manual Labeling: A subset of ten thousand images were labeled using `label_back_of_car.py` where a label of 1 is the "back of the car" and 0 is an image that is "not of the back of a car" (front of car, side of car, non-car image etc). Hand-labeling the data was done over multiple days and is estimated to have taken a total time of around 6-8 hours.

2) Dataset Splitting: The model was trained with 80% of randomly selected data, and validated with the other 20%.

3) Model Architecture:

- ResNet-18 (ImageNet weights)
- Input size: 224×224
- Final layer: Linear(512, 2)

4) Training Configuration:

```
Optimizer: AdamW
Learning rate: 1e-4
Weight decay: 1e-4
Batch size: 32
Epochs: 8
Loss: Cross-Entropy
```

5) Inference Pipeline: Batch inference was done using the script `infer_all_front_back.py`.

6) Results: Training of the model took around 20 minutes on a Apple M4 Pro chip, and the best model produced the following results on the validation set:

```
Best validation accuracy: 95.20%
Precision: 93.00%
Recall: 91.51%
F1: 92.25%
```

Running this best model on the full dataset using GPU took approximately one day and led to 1,385,302 images "passing" the model (images of the back of cars), and was a reduction from the previous shell script of around 70%. This approximately 70% also matched the ratio of non-back to back of cars seen in the hand-labeled dataset.

IV. Schema Definition & JSON Annotation Format

A. Label Schema

The labeling schema is defined in `jupyter/labeling_schema.py`:

```
TURN_SIGNAL_LABELS = ["left", "right", "none",
    "both"]
TAIL_LIGHT_LABELS = ["on", "off", "not_visible
    "]
```

B. JSON Annotation Format

A Python script for labeling (`scripts/hand_label_python.py`) was created. This was created for single-image labeling of ten thousand images, and was later abandoned once images were sequenced, as deciding an image state based on one image is very difficult. Hand-labeled annotations for single-image testing are stored in JSON format with the following structure:

```
[
  {
    "image": "relative/path/to/crop.jpg",
    "turn_signal": "left|right|none|both|
       unclear",
    "tail_light": "on|off|not_visible|unclear"
  }
]
```

V. Single-Image Classification

A. OpenAI Testing Setup

To establish a baseline for large language model (LLM) vision capabilities on this task, I evaluated OpenAI's GPT-4o and GPT-4o-mini models on a stratified sample of 130 hand-labeled images. This experiment assessed whether foundation

models could perform zero-shot turn signal classification without task-specific training. This included tail light classification as well; however, the focus was later decided to not focus on this aspect as tail lights are required to be on in Estonia. Moreover, often in images, tail lights were too dim to accurately determine their state, and likely the labeling here was incorrect.

Turn Signal	Tail Light	Sample Count
none	off	30
left	off	20
left	on	15
left	unclear	6
right	off	20
right	on	15
right	unclear	10
none	on	7
none	unclear	7
Total		**130**

TABLE I
SAMPLE COUNTS FOR COMBINATIONS OF TURN SIGNAL AND TAIL LIGHT STATES.

This sampling strategy overrepresented challenging cases (unclear labels, left/right signals).

Images were encoded as base64 and submitted to the OpenAI API with structured JSON output format ($response\_format = "type" : "json\_object"$). All experiments used temperature=0 for deterministic responses. Both GPT-4o (full model) and GPT-4o-mini (smaller, faster variant) were evaluated.

Three prompts were tested to assess the impact of instruction clarity and label set specification.

Prompt 1 (Baseline)

```
You are an advanced image analysis model. Look
     at the car image and determine:
1. turn_signal - one of: left, right, unclear,
     none, both
2. tail_light - one of: on, off, unclear

Return only valid JSON in this format:
{
  "turn_signal": "left",
  "tail_light": "on"
}
```

This prompt included all possible labels, including "both" for hazard lights, with minimal task description.

Prompt 2 (Simplified Label Set)

```
You are an advanced image analysis model. Look
     at the car image and determine:
1. turn_signal - one of: left, right, unclear,
     none
2. tail_light - one of: on, off, unclear

Return only valid JSON in this format:
{
  "turn_signal": "left",
  "tail_light": "on"
}
```

The "both" option was removed to test whether a reduced label set would improve classification accuracy. This modification was motivated by the fact that there were very few crops in the "both" category, as "both" was difficult to distinguish from the tail lights simply being on in a single image.

Prompt 3 (Task-Specific Definitions)

```
You are an advanced image analysis model. Look
     at the car image and determine:
1. turn_signal - one of: left, right, unclear,
     none
2. tail_light - one of: on, off, unclear
A turn signal should be lit up and indicating
    which direction the car is planning to
    turn
A tail light is on if lights other than the
    turn signal are on.
Return only valid JSON in this format:
{
  "turn_signal": "left",
  "tail_light": "on"
}
```

This prompt added explicit definitions to clarify the distinction between turn signals and tail lights, addressing potential ambiguity when both may be illuminated simultaneously.

All results are discussed in section VII (Results and Analysis).

VI. TEMPORAL IMAGE CLASSIFICATION

Due to the difficulty to hand label single-images with turn signals, I decided to sequence the frames. Tracking, or sequencing, is the process of following an object (in this case, a car) over multiple frames, and ordering this output.

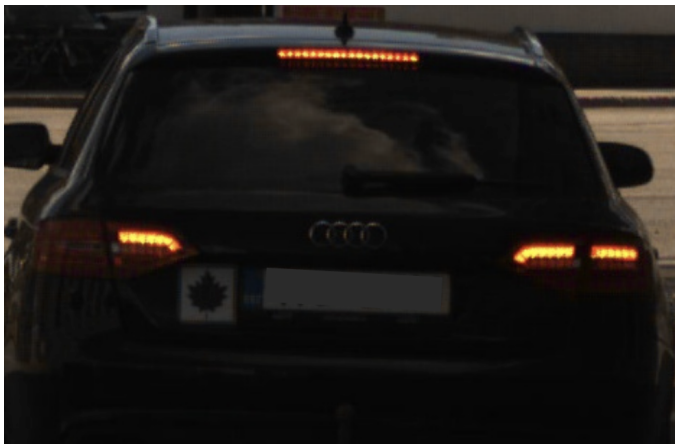

Fig. 1. Example of a difficult to label image.

A. Detection CSV Consolidation

DeepSORT (Deep Simple Online and Realtime Tracking) is an object tracking algorithm that extends the original SORT algorithm by incorporating appearance information through a deep neural network [1]. This allows it to maintain consistent identities for objects even during occlusions or when objects cross paths. In practice, each row in the CSV corresponds to a

detected object in a single frame, including the frame number, bounding box coordinates, detection confidence, and an object ID if it has already been tracked.

In order to create a CSV that would work for DeepSort, I created the script `prepare_detections_from_yolo_txt.py`. This script takes the YOLO ouput bounding boxes, class labels, and confidence scores and converts them into the format expected by DeepSORT.

1) Implementation Details: The script begins by extracting frame IDs from the video or image sequence to keep track of which detections belong to which frame. It then converts the bounding box coordinates provided by the YOLO detector (which are normalized relative to the image size) into absolute pixel coordinates.

2) Coordinate Conversion: YOLO outputs bounding boxes in the format of normalized center coordinates and width/height relative to the image size. To convert them to pixel coordinates and the top-left/bottom-right format expected by DeepSORT, the following transformations are applied:

```
x_center = cx * img_width
y_center = cy * img_height
box_width = w * img_width
box_height = h * img_height

x1 = x_center - box_width / 2
y1 = y_center - box_height / 2
x2 = x_center + box_width / 2
y2 = y_center + box_height / 2
```

Output Schema:

```
sequence,frame_id,crop_path,frame_path,width,
    height,
class_id,score,x1,y1,x2,y2
```

B. DeepSORT Output

Metric	Value
Total tracks	40,872
Mean track length	26.1 frames
Median track length	11.0 frames
Max track length	1,756 frames

TABLE II
DEEPSORT TRACKING SUMMARY STATISTICS.

C. Sequence Sampling

Turn signals exhibit a characteristic periodic behavior, typically blinking at frequencies between 1.0 and 2.5 Hz (60-150 blinks per minute) as mandated by automotive safety regulations. In contrast, tail lights and brake lights remain continuously illuminated when active, producing a steady-state intensity profile. Looking at the images (10 FPS video frames) and experimenting with different sampling rates, I determined that sampling every fourth image reliably would show an on and an off image one after another. While I did do research into the regulations about the frequency of flashing for turn signals, I found that turn signals can vary from car to car (and if one bulb on a side is dead, many vehicles will intentionally blink the remaining signal at a higher rate due to thermal flasher relay that depends on electrical load, adding another source of variation), so every fourth image was chosen, and worked quite well.

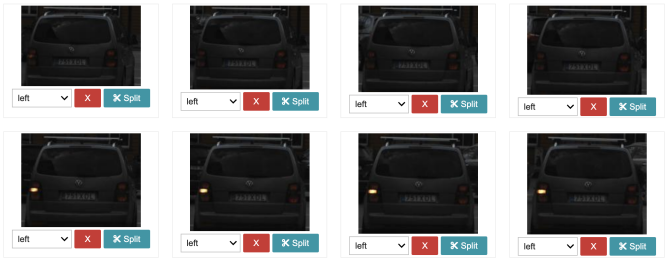

Fig. 2. Example of every fourth image flashing on and off when looking at every image in a sequence.

D. OpenAI Performance on multiple images

Having established baseline performance on single images (VII), the evaluation was extended to temporal sequences to assess whether foundation models could use frame-to-frame context for improved turn signal detection.

Sequences were constructed using the sampling strategy described above. Rather than submitting individual frames, all frames in a sequence were arranged into a spatial grid (left-to-right, top-to-bottom order) and presented as a single composite image to the vision model. This grid-based approach enables the model to observe temporal evolution while remaining within the single-image API constraints.

A single prompt template was used for temporal sequence evaluation, as preliminary experiments and the stronger performance of the computational heuristic method (described below) suggested diminishing returns from extensive prompt engineering and resource use. The prompt is shown below:

```
Analyze this grid of N sequential car images (
    left-to-right, top-to-bottom).

Task: Determine turn signal status:
- ``left'': Left turn signal blinking
- ``right'': Right turn signal blinking
- ``hazard'': Both signals blinking
- ``none'': No turn signals active

Key points:
- Turn signals blink (on/off pattern across
    frames)
- Look for amber/orange lights
- Hazard = both sides blink together
\end{quote}
```

The model was instructed to respond in structured JSON format. Due to the superior performance of the computational heuristic method and API cost considerations, foundation model evaluation was limited to a subset of 10 sequences sampled from the full test set (128 images total, most sequences including 13 images per sequence). This limited evaluation served to assess whether foundation models could provide

value beyond the heuristic approach, rather than establishing comprehensive performance metrics.

Both GPT-4o-mini and GPT-4o were evaluated, with a 1-second delay between API calls to respect rate limits. All requests used temperature=0.1, and `response_format={ "type": "json_object" }`.

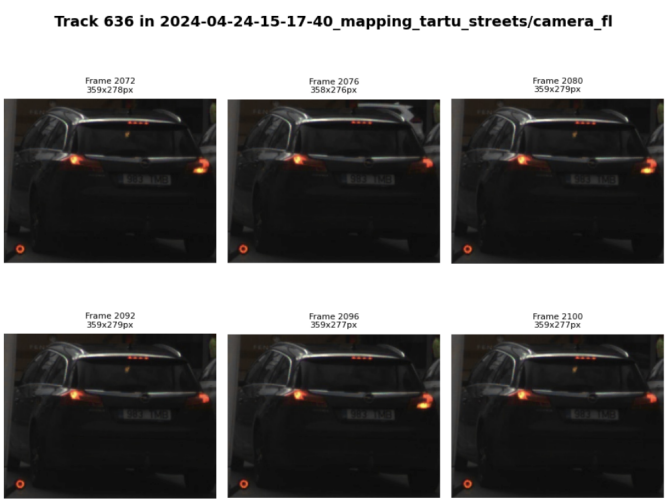

Fig. 3. Sequenced images of flashing right turn signal.

E. Turn Signal Heuristic Model

Beyond zero-shot foundation models, I developed a computational heuristic method to detect turn signals without machine learning. This approach leverages the inherent temporal and spectral properties of turn signal behavior to distinguish active signals from static tail lights.

The method begins by isolating the yellow-orange spectral band characteristic of turn signal emissions. Images are first converted from RGB to HSV (Hue-Saturation-Value) color space, which provides perceptual uniformity and separates chromatic information from luminance. A color mask is applied to extract pixels within the hue range of 15°-35° (yellow-orange), with saturation values exceeding 100 (on a 0-255 scale) to exclude pale or washed-out regions, and value thresholds above 100 to reduce ambient reflections. This color-selective filtering produces a binary mask.

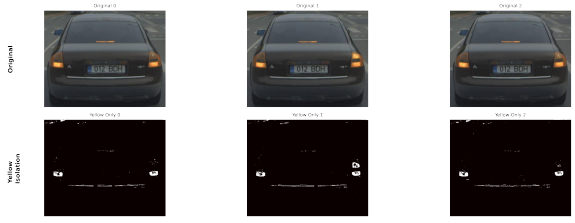

Fig. 4. Masking bright yellow color channels.

For each frame in a sequence, the sum of masked pixels provides a scalar intensity measure representing the total yellow light present. This intensity time series captures the temporal evolution of potential turn signals across the sequence. By sampling frames at a known rate (every fourth image after subsampling from the original 10 FPS video), a construct uniformly sampled signal suitable for frequency domain analysis was constructed.

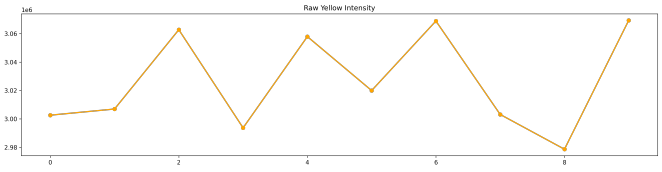

Fig. 5. Raw yellow intensities over frames.

To detect periodicity in the intensity time series, a Fast Fourier Transform (FFT) was used, which decomposes the signal into constituent frequency components. The intensity series is first normalized to allow the FFT to capture oscillatory behavior rather than absolute brightness. The FFT produces a spectrum of frequency components, from which positive frequencies are extracted and peaks are identified within the expected turn signal frequency band (1.0-2.5 Hz). A signal is classified as periodic if the peak power within this band exceeds three times the mean spectral power, which was chosen to distinguish true periodic signals from noise.

To distinguish between left and right turn signals, the method performs spatial analysis by defining regions of interest (ROIs) corresponding to the left and right sides of the image. For a rear-facing vehicle crop, the left ROI encompasses the leftmost 40% of the image width, while the right ROI covers the rightmost 40%, both vertically bounded to the lower 60% of the image where turn lights typically appear. Intensity time series are extracted independently for each ROI, and the standard deviation of each series quantifies the temporal variability (i.e., blinking activity) in that spatial region. A higher standard deviation indicates more pronounced intensity fluctuations, characteristic of an active turn signal.

The final classification follows a hierarchical decision tree:

1) **Global Periodicity Check:** If the full-image intensity series exhibits no periodicity and low variance (standard deviation less than 5% of mean intensity), classify as none (no active turn signal).
2) **Activity Threshold:** If blinking is detected but the maximum ROI activity falls below a threshold (which was determined from a test set), classify as none.
3) **Bilateral Comparison:** Compare left and right ROI activity levels:
   - If the ratio of minimum to maximum activity exceeds 0.7, both sides are blinking similarly, classify as hazard.
   - If left activity exceeds right activity, classify as left.
   - Otherwise, classify as right.

Some advantages and disadvantages to this method are mentioned below.

Advantages:

- Zero training data required
- Interpretable decision making based on physical signal properties
- Computationally efficient (no neural network inference)
- Generally robust to appearance variation (vehicle color, make, distance)

Limitations:

- Sensitive to threshold tuning (activity threshold, periodicity detection)
- Requires sufficient temporal coverage (minimum sequence length $\sim$10 frames)
- ROI localization assumes standardized vehicle orientation
- Fails for non-standard lighting configurations or severe occlusion

VII. Results and Analysis

A. Single-Image API

Accuracy of each model and prompt is displayed below:

Model	Prompt	Accuracy	Left	Right	None
gpt-4o-mini	prompt 1	44%	36%	55.56	28%
gpt-4o	prompt 1	63%	58%	80.00	21%
gpt-4o-mini	prompt 2	45%	34%	55.56	43%
gpt-4o	prompt 2	65%	61%	80.00	29%
gpt-4o-mini	prompt 3	40%	15%	60.00	50%
gpt-4o	prompt 3	66%	61%	82.22	28%

TABLE III
OVERALL TURN SIGNAL ACCURACY AND PER-CLASS ACCURACY FOR OPENAI API.

GPT-4o substantially outperformed GPT-4o-mini (70% vs. 40%), which shows that model capacity matters for this task. Prompt variations have only a marginal effect on overall accuracy. Removing the both label (Prompt 2) and adding explicit task definitions (Prompt 3) lead to modest improvements in certain per-class accuracies, but do not fundamentally alter model behavior. This suggests that the primary limitation is not prompt ambiguity, but rather the intrinsic difficulty of distinguishing turn signals in static images. These results seem to indicate that single-frame classification is insufficient for reliable turn signal detection in real-world conditions, regardless of model scale or prompt engineering.

B. Multiple-Image API

Model	Sequence	Acc.	Left	Right	None
GPT-4o-mini	temporal-grid	40%	40%	40%	0%
GPT-4o	temporal-grid	70%	100%	40%	0%

TABLE IV
TURN SIGNAL CLASSIFICATION ACCURACY ON TEMPORAL SEQUENCE EVALUATION.

The introduction of temporal context via grid-based image sequences insignificantly improves overall accuracy for GPT-4o from approximately 66% (single image) to 70%; however, the limited sequences makes this result unreliable. One observation from these results is the inability of both foundation models to correctly classify sequences in which no turn signal is active, resulting in an accuracy of 0% for the none class. This consistent failure indicates a systematic bias toward predicting the presence of a turn signal when processing vehicle imagery. This may arise from multiple factors, including the lack of "prompt engineering" (this was due to trying to limit costs), inherent challenges in distinguishing inactive turn signals from static tail lights, and the foundation model not being able to look at a sequence/grid of images and understand the temporal context.

C. Heuristic Method

In contrast to foundation model approaches, the computational heuristic method achieves an overall accuracy of 80%, outperforming both single-image and temporal foundation model evaluations.

Class	Accuracy
Left	0.57
None	0.83
Hazard	0.76
Right	0.61

TABLE V
PER-CLASS ACCURACY AND NUMBER OF EVALUATION WINDOWS.

These results are from running the heuristic method on 212,919 hand-checked images, encompassing 1010 sequences. In this hand-labeled sequence set, 183,964 of the images were classified as "none", 17,142 as left, 11,667 as right, and 146 as hazard.

VIII. Discussion

Recent advances in large-scale foundation models have led to their widespread adoption across a variety of perception tasks, often with the expectation that general-purpose models can replace task-specific pipelines with minimal engineering effort. While these models have demonstrated impressive performance on static image understanding and zero-shot classification benchmarks [2], the results of this project show some limitations when such models are applied to temporal perception tasks, specifically in autonomous driving.

These findings suggest that the tested foundation models lack an explicit mechanism for reasoning about temporal periodicity, a feature of turn signals. While the models can visually detect illuminated regions and reason semantically about vehicle components, they appear unable to distinguish static illumination from intentional signaling. The grid-based representation of time, while intuitive to humans, does not provide a sufficiently structured temporal signal for the models to perform reliable frequency-based reasoning. Importantly, these results do not imply that foundation models are unsuitable for autonomous driving perception more broadly. Rather, they show the risk of applying such models naively to tasks.

IX. Conclusion

This work presents an experiment for creating a pipeline for large-scale vehicle turn signal detection from dashcam footage,

spanning data collection, filtering, tracking, and classification. Through this, I demonstrate that turn signal detection is inherently a temporal problem that cannot be reliably solved using static imagery alone. Zero-shot foundation models, while impressive in many vision tasks, exhibit critical failure modes in this domain. In contrast, the presented computational heuristic model achieves superior accuracy, while still not being robust when compared to hand-labeled images.

X. Future Work

Several directions emerge from this work that could further improve turn signal detection and expand the utility of the dataset. This study primarily evaluated image-based foundation models with limited temporal reasoning capabilities. An extension is to investigate video-native foundation models, such as the NVIDIA Cosmos model family, which are trained on video [3]. These models may better capture temporal patterns inherent in turn signal behavior, potentially addressing some of the limitations observed in the grid-based image representations. This work also focused on rear-facing vehicle crops; however, front-facing turn signals also convey valuable intent information, particularly in intersections and roundabout traffic scenarios.

The dataset produced by this provides a foundation for training task-specific detection models. Future work includes using the labeled crops and temporal sequences to train object detection or segmentation models such as YOLO or Grounding DINO.

References

[1] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017. [Online]. Available: https://arxiv.org/abs/1703.07402

[2] R. Ramachandran, A. Garjani, R. Bachmann, A. Atanov, O. F. Kar, and A. Zamir, "How well does gpt-4o understand vision? evaluating multimodal foundation models on standard computer vision tasks," 2025. [Online]. Available: https://arxiv.org/abs/2507.01955

[3] NVIDIA, :, A. Azzolini, J. Bai, H. Brandon, J. Cao, P. Chattopadhyay, H. Chen, J. Chu, Y. Cui, J. Diamond, Y. Ding, L. Feng, F. Ferroni, R. Govindaraju, J. Gu, S. Gururani, I. E. Hanafi, Z. Hao, J. Huffman, J. Jin, B. Johnson, R. Khan, G. Kurian, E. Lantz, N. Lee, Z. Li, X. Li, M. Liao, T.-Y. Lin, Y.-C. Lin, M.-Y. Liu, X. Lu, A. Luo, A. Mathau, Y. Ni, L. Pavao, W. Ping, D. W. Romero, M. Smelyanskiy, S. Song, L. Tchapmi, A. Z. Wang, B. Wang, H. Wang, F. Wei, J. Xu, Y. Xu, D. Yang, X. Yang, Z. Yang, J. Zhang, X. Zeng, and Z. Zhang, "Cosmos-reason1: From physical common sense to embodied reasoning," 2025. [Online]. Available: https://arxiv.org/abs/2503.15558