JOURNAL OF UNDERGRADUATE RESEARCH

# Battle of the YOLOs — Augmenting and Benchmarking Fisheye Camera YOLO Models for Traffic Intersection Object Detection

**J.P. Fleischer[1], Julian Dominguez[1], Artur Kumik[2], Sargam Thakur[1], Gregor von Laszewski[3], and Sanjay Ranka[1]**

[1]*Herbert Wertheim College of Engineering, University of Florida*
[2]*College of Liberal Arts and Sciences, University of Florida*
[3]*Biocomplexity Institute and Initiative, University of Virginia*

Mentors: Gregor von Laszewski, Ph.D.[3]
Sanjay Ranka, Ph.D.[1], Department of Computer and Information Science and Engineering

Abstract

Our data pipeline, which leverages a computer vision model, can ingest fisheye camera footage and compute statistics such as car volume and the near miss rate per intersection. However, accurate metrics are dependent on the model's class detection accuracy. As new versions of Darknet/YOLO (You Only Look Once) are released and changes to the intersection are completed, the model must be retrained and benchmarked to improve accuracy. We report on a new, integrated process that automates model training from annotation to deployment, producing a YOLOv4 model with 99.10% mAP (mean average precision) across all classes within a fisheye traffic camera environment. The training is containerized and portable across computer environments, including consumer-grade personal computers and high-performance computing clusters.

*Keywords:* computer vision, high-performance computing, neural networks, pedestrian safety

## Introduction

The field of urban planning and traffic engineering is intertwined with enhancing motorist and pedestrian safety; initiatives such as the United States Department of Transportation's Vision Zero aim to achieve zero fatalities within transportation systems. To enhance safety, engineers will implement countermeasures within signalized intersections. However, while counting the rate of car accidents can give some insight into countermeasure effectiveness, determining what contributes to a near miss (also referred to as a conflict) can better signal patterns that contribute to potential crashes.

Identifying near misses accurately is dependent on a detector model that can correctly classify and localize objects within the traffic camera feed. The coordinates of these detections

are then used to determine speed and potential collision with other objects. Since the model is crucial for safety analysis, benchmarking the training and performance of the model in a variety of different configurations, including hardware and configuration parameters, is vital. Factors that influence training and performance include the graphics processing unit (GPU), central processing unit (CPU), and YOLO version configuration. However, to easily conduct benchmarking across these environments, a portable containerization method is required that dynamically compiles Darknet, which is a framework written in C++ that conducts YOLO model training.

The completion of such a method will facilitate observing the impact of model training on GPU readings, including temperature, as well as the total training time required. A sufficiently portable method can be executed on a range of computers, including high-performance computing (HPC) clusters that run Linux at various universities. However, it must also be able to run on consumer-grade GPUs and machines with Windows or macOS operating systems. These benchmarks will grant researchers not only a high-level survey of the fastest and most powerful hardware configurations currently available, but also a method for computer vision modeling on accessible hardware. This method will then enable researchers to quickly train and assess new models for traffic object detection.

Using such a method, this paper attempts to answer the following questions: **Do multiple GPUs increase the speed of Darknet training? Which YOLO configuration is the most accurate against a validation set?**

## Literature Review

### Methodologies that survey near misses and accidents in traffic

Studies in recent years tend to favor the use of learning models with varying levels of supervision to classify objects and/or incidents to eliminate the time-consuming data annotation stage. Most of these studies use various YOLO models.

Studies not using AI typically use statistical analysis techniques such as Kalman filtering to track vehicles or predict their motion (Sharma et al., 2019), (Bewley et al., 2016). The technique has been used to solve the velocity of a target item's motion (Bewley et al., 2016). This method has also been used to extract the backgrounds of traffic images to assess vehicle density (Balcilar

& Sonmez, 2008). One tool within the pipeline, named FastMOT, combines Kalman filtering with detection to achieve accurate object tracking metrics (Yang, 2020).

In prior studies that did not use AI to streamline object and incident detection, simple data collection and analysis methods have been used to evaluate near miss incidents. In a study by Siregar et al., trained human observers obtained near miss accident data at a particular traffic junction (2018). This data was used to categorize conflicts by vehicle types and the types of maneuvers most common to each conflict type (crossing, weaving, merging, diverging). However, this method of analysis of traffic incidents has quickly become replaced by techniques not reliant on the human eye.

**Pedestrian fisheye camera model**

In recent years, numerous machine learning models analyzing fisheye cameras have been created, with many focused on identifying pedestrians and vehicles. These models try to use the wide angle that is characteristic of many fisheye cameras, while trying to minimize distortion using various techniques, to achieve useful and accurate results.

Most recently, scientific studies have focused on the use of machine learning methods for pedestrian detection. In a notable study by Chiang et al. (2021), a machine learning model was implemented, using YOLOv2 and v3, along with a new technique of using composite images of perspective views. This new technique aims to minimize the distortion of the fisheye camera by separating the image into smaller sub-images. The results return mean Average Precisions (mAPs) which are generally higher than other studies tackling similar problems and suggest improvements and developments in the field.

**Darknet YOLO Comparison**

1) Comparison of YOLO Versions

YOLO has evolved from the initial release (YOLOv1) to recent versions (YOLOv4, YOLOv7, YOLOv10, etc.). According to the Darknet FAQ, YOLOv3-tiny is typically considered faster than YOLOv4-tiny, although YOLOv4-tiny offers higher quality output, and YOLOv7-tiny tends to be slower with lower quality output compared to YOLOv3-tiny and YOLOv4-tiny (Charette, 2025). In addition, the original YOLO9000 paper (Redmon & Farhadi, 2016) and the YOLOv4 paper (Bochkovskiy et al., 2020) report improvements in accuracy and inference speed with each subsequent version.

2) Comparison Between Darknet and Other Frameworks (e.g., Ultralytics)

The Darknet framework is the original platform for YOLO; Darknet is highly optimized for NVIDIA GPUs using CUDA and is written in C++. Recent developments, such as the hank.ai fork, extend Darknet's compatibility by adding ROCm support for AMD GPUs. In contrast, newer YOLO frameworks like Ultralytics (e.g., YOLOv10) are implemented in PyTorch, providing a modular API and additional features (e.g., segmentation, pose estimation). However, benchmark tests indicate that Darknet processes frames significantly faster than Ultralytics' YOLOv10 models, which can be up to five times slower (Charette, 2022).

## Methodology

### Software Architecture

To create an accurate dataset for training and to benchmark the training pipeline, various open-source software tools were used. One software, DarkMark, is a frame-by-frame annotation tool that has been developed specifically for use with the Darknet neural network framework. This tool was used to create annotations of images because it stores data in a way that can be straightforwardly distributed to other HPC clusters using tools such as Globus. Globus is used to store and transfer large datasets. CVAT (Computer Vision Annotation Tool) is another existing data annotation tool that offers command-line and application programming interface (API) usage. However, DarkMark offers time-saving tools that will be expanded on in a later section. Cloudmesh-gpu is a tool used to run benchmarking, providing information about GPU time, temperature, and energy usage (von Laszewski, 2022). The information provided by the benchmark runs for different GPUs is critical to examine the practicality of certain GPUs for applications with different priorities, resources, and desired capabilities.

Docker is a software platform for containerizing applications, allowing them to run on different hardware machines without encountering errors related to package versions and installations. Apptainer, although also containerization software, is often used on HPC clusters instead of Docker. This is the first instance of containerized Darknet/YOLO for Docker and Apptainer, compatible with HPC machines. NVIDIA CUDA (Compute Unified Device Architecture) and cuDNN (CUDA Deep Neural Network) are required libraries for GPU utilization and are included within the Docker/Apptainer containers.

These tools are easily integrated into the software architecture, which is shown in Figure 1.

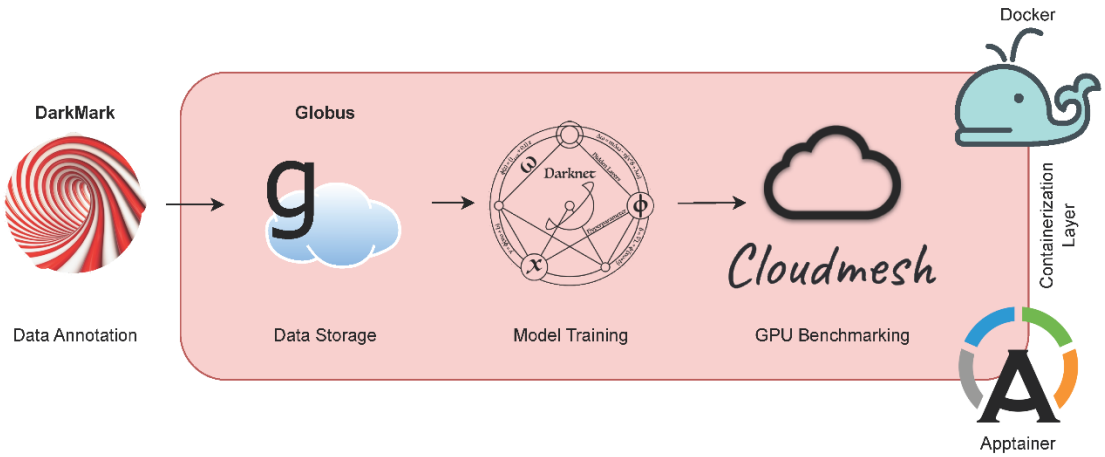


**Figure 1.** The data training process architecture from annotation to benchmarking.

Data annotation tools are still in their infancy; oftentimes the most popular platforms lack intuitive features and contain interfaces that require complex navigation. Some issues encountered while annotating over 30,000 images are:

1) There is no standard way to cleanly and efficiently handle the verification of automatic annotations.
    a. In DarkMark, additional features are needed such as mass-delete.
    b. Automatic annotations introduce jitter, or the slight movement of the bounding boxes across frames, to stationary objects.
2) Although a linear merging tool was beneficial in annotation, linearity does not conform with the exponential distortion effect caused by fisheye cameras.
3) Styles or criteria may not be consistent across annotators, leading to model confusion.
    a. Should far-away entities be annotated?
    b. What constitutes a “bus” or “truck”?
    c. Does a pick-up truck hauling a trailer constitute one truck, or should only the pick-up truck be marked?

The importance of creating a centralized documentation base to mitigate these issues and encourage collaboration across annotators cannot be overstated. Any annotation style will work if the annotations are consistent across the dataset. A documentation base for annotator reference can be created via Docsy (Hugo) or Bookstack, which are software used for keeping scientific notes.

While annotations must be consistent, they must be easily created with the use of helpful annotation software such as DarkMark. To further ease and accelerate the use of the software, the authors have contributed to the DarkMark codebase and created pull requests at https://github.com/stephanecharette/DarkMark/pulls?q=is%3Aclosed (#39, #46 and #47), adding the following features:

1) Merge mode - interpolate new marks between two preexisting marks across a gap of frames
2) Mass delete mode - delete an area of marks across a specified number of frames
3) Copy a mark forward a specified number of frames
4) Panning while zoomed in

These features facilitate the quick creation of quality training data to create an accurate model. For a satisfactorily performing model, all possible appearances of an object must be annotated, including in all possible locations. Equation 1 is derived to estimate *N*, where *N* is the Cartesian product of all possible features (appearance and location) within the intersection. In other words, *N* is the total number of annotated images required to cover all object appearances in various environments.

$$N = \underbrace{2}_{\text{day/night}} \times \underbrace{6}_{\text{intersections}} \times \underbrace{8}_{\text{car colors}} \times \underbrace{8}_{\text{car silhouettes}} \times$$

$$\underbrace{5}_{\substack{\text{classes} \\ \text{(car, truck, bus, pedestrian, motorbike)}}} \times \underbrace{4}_{\text{4 legs of intersection}} \times \underbrace{2}_{\text{pedestrian on scooter or bicycle}} \times \ldots \quad (1)$$

While other factors likely affect the necessary image count, such as specific times of day (early morning, noon, afternoon), *N* can be conservatively estimated to be in the realm of 30,000, which is the number resulting from Equation 1.

Developers can train a model using all annotated images or only a certain percentage (such as 80%). If the developer uses a percentage, the remaining images are part of a validation set that the training phase never sees. Then, validating the model informs the developer of the optimal parameters for tweaking the model's performance. However, if every image depicts a new feature that the model must learn, then hiding an image in the validation set will fail to teach that image's feature. Thus, the authors use an empty validation set, as training with a non-empty set can cause volatile, non-deterministic accuracy within a model. DarkMark provides and recommends the option to use 100% of images for training.

### Advantages of DarkMark Compared to CVAT

Table 1 summarizes the pros and cons of using DarkMark compared to CVAT.

**Table 1.** Feature comparison between DarkMark and CVAT.

| Feature | DarkMark | CVAT |
|---|---|---|
| Open-source | ✓ | ✓ |
| API | ✓ | ✓ |
| Automatic annotations | ✓ | ✓ |
| Faster and more reliable, less latency when navigating through frames | ✓ | |
| Easier to make code contributions due to cleaner codebase | ✓ | |
| Seamless training integration | ✓ | |
| Intuitive annotation tools such as mass-delete | ✓ | |
| Centralized storage, allowing easier collaborative annotating | | ✓ |
| Simpler build process | | ✓ |
| Polished interface | | ✓ |

DarkMark provides seamless training integration with Darknet by generating shell scripts to begin the training, as well as the necessary configuration files to train the neural network (Charette, 2025).

While CVAT has superior quality-of-life features for initial setup, which is a one-time occurrence, annotating is a repetitive process that happens on many images. Therefore, DarkMark is preferable due to its short latencies, convenient hotkeys, and better annotation features. Although these platforms are only a few years old, they will likely grow to adapt further to annotators' needs.

### Benchmarking across HPC Environments

Benchmarking was conducted on both the University of Florida (UF) and the University of Virginia (UVA)'s high-performance computing (HPC) clusters. It became clear that the two systems had key differences within their computing environments. These differences had to be accounted for to enable consistent, containerized benchmarking. For the UVA HPC, users are granted a particular number of compute hours that are paid for by a principal investigator's allocation. On the other hand, on the UF HPC, principal investigators pay for a set period under an allocation, where approved users may run jobs of any duration during that time. Both systems are contingent on available resources, i.e. the Slurm (job submission software for HPCs) queue

must not be full or in-use by other users. Although these HPCs differed in ways of access, they provided invaluable resources for training the model, such as the A100, V100, RTX 2080 Ti, and other GPUs.

Four characteristics must be considered when developing an environment-agnostic HPC benchmarking algorithm:

1) Partition names in Slurm are not consistent across HPCs.

2) Indices indicating which GPU is allocated to the Slurm job (using CUDA_VISIBLE_DEVICES environment variable) are sometimes set starting from 0, or they are set according to their index on the allocated Slurm node.

3) Some HPC clusters have login nodes with sufficient memory to build an Apptainer container, but some do not, requiring the build to be performed within the Slurm job (potentially using up significant job time).

4) Lmod, the module software commonly used within HPC clusters (McLay et al., 2011), will sometimes persistently make available required modules such as Apptainer, but sometimes HPC nodes require a module load with each new login.

This benchmarking algorithm (https://github.com/jpfleischer/chocolatechip/blob/ea02c42b9942fb18760021f46fbb5883d31a864e/spring2025/darknet/Makefile) accounts for these scenarios across UF's HiPerGator and UVA's Afton. However, rather than accounting for each scenario introduced by accommodating each institution's HPC cluster, it would be simpler for researchers if clusters all followed a common standard that ensures a reliable, consistent environment.

**Datasets**

The Lego Gears dataset, provided in the Darknet documentation pages, is a dataset featuring LEGO pieces to simulate a simple computer vision classification scenario. Such a small dataset serves as a suitable proxy to understand how larger ones, such as the Traffic Cars dataset, may function across computing environments. Table 2 illustrates the size difference of these datasets, with Traffic Cars having many more images of higher resolution. The network configuration must be of a high size; resizing the images would cause small objects, such as pedestrians, to

become pixelated such that they are unidentifiable. As a result, Traffic Cars takes much longer to train and requires more VRAM than a typical consumer-grade system provides.

**Table 2**: Darknet Dataset Comparison.

| Dataset | Hosted Location | Number of Images | Configuration | Average Training Time | VRAM Required |
|---|---|---|---|---|---|
| Lego Gears | Darknet Documentation Website | 90 | 224x160 pixels YOLOv4-Tiny | 9 minutes (on consumer-grade GPU) | 500MB-1.5GB |
| Traffic Cars | Globus | 28,871 | 960x736 pixels YOLOv4-Full | 1 day 20 hours (on HPC GPU) | 48GB |

## Results

### Lego Gears YOLOv4-Tiny Training Comparison

The bar chart in Figure 2 shows the results of a training benchmark done with Darknet on the Lego Gears dataset on different configurations.

Based on the results, it is more efficient to train small datasets on personal computers with fast storage capabilities and GPUs. The HPC clusters (Afton at UVA and HiPerGator at UF) were slower than the personal computers. However, while HPC machines train at a slower pace, they can train higher fidelity models such as YOLOv4-Full, whereas GPUs on personal machines lack the VRAM to do so.

Of the graphics cards tested, the best benchmark results occurred from the NVIDIA GeForce 3090 and NVIDIA A100 graphics cards. The other graphics cards, such as the 2080 Ti and Quadro, returned similar results, but ultimately returned slower benchmark times than the aforementioned cards.

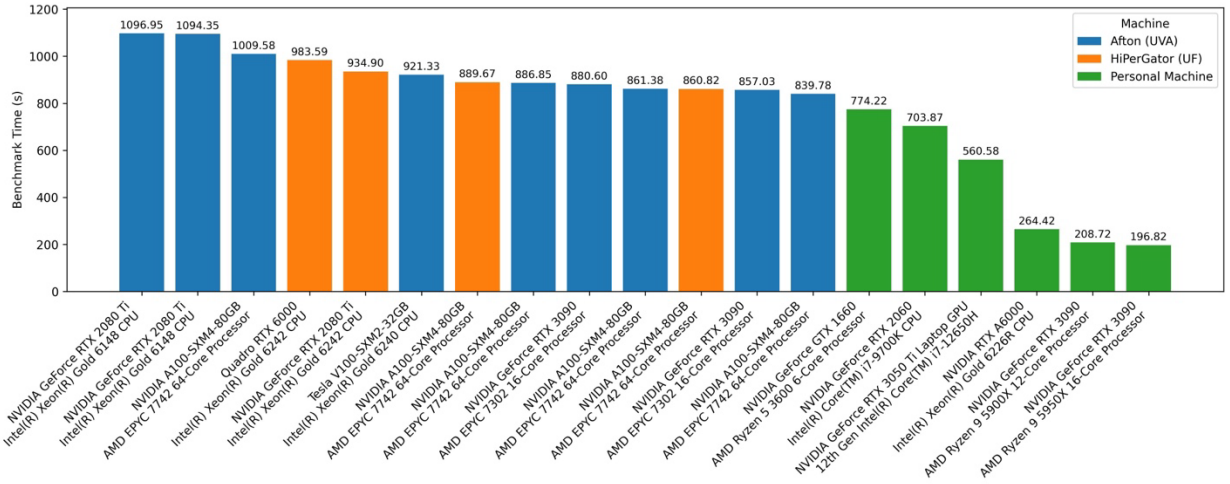


**Figure 2.** Benchmark results for single-GPU runs ranked by training time for Lego Gears dataset (YOLOv4-Tiny).

NVIDIA GeForce 3090 cards were available on both Afton and personal computers. These cards performed better compared to the other cards on both HPC clusters and personal computers, returning the quickest benchmark times.

**Table 3.** Comparison of GPU Specifications. (NVIDIA, n.d., 2022a, 2022b)

| | Power Draw | Memory | Memory Bandwidth | FP32 |
|---|---|---|---|---|
| A100 SXM | 400W | 80 GB | 2039 GB/s | 19.5 TFLOPS |
| 3090 | 350W | 24 GB | 936 GB/s | 35.58 TFLOPS |
| 2080 Ti | 250W | 11 GB | 616 GB/s | 13.45 TFLOPS |

In Table 3, the different GPUs used in benchmarking are compared, focusing on specifications that played a role in the variation of results. In accordance with the findings, the A100 and 3090 have much better specifications than the 2080Ti, resulting in a significant increase in performance.

In Figure 3, the GPU temperatures while running benchmark tests over time are shown. While HPC clusters (shown in blue) take a longer time to complete training compared to the personal computers (red), which is consistent with previous findings, they stay significantly cooler for the entire testing. This fact is emphasized by Figure 4, which compares temperatures before and after training. These lower temperatures are due to the advanced cooling solutions

present in HPC clusters. When benchmarking for short periods of time, high temperatures on personal computers are not a problem. However, when training for longer, HPC clusters can better handle the high temperatures. Trainings with multiple GPUs are also shown, which ran on MALTLab (the name of the authors' research lab) machines.

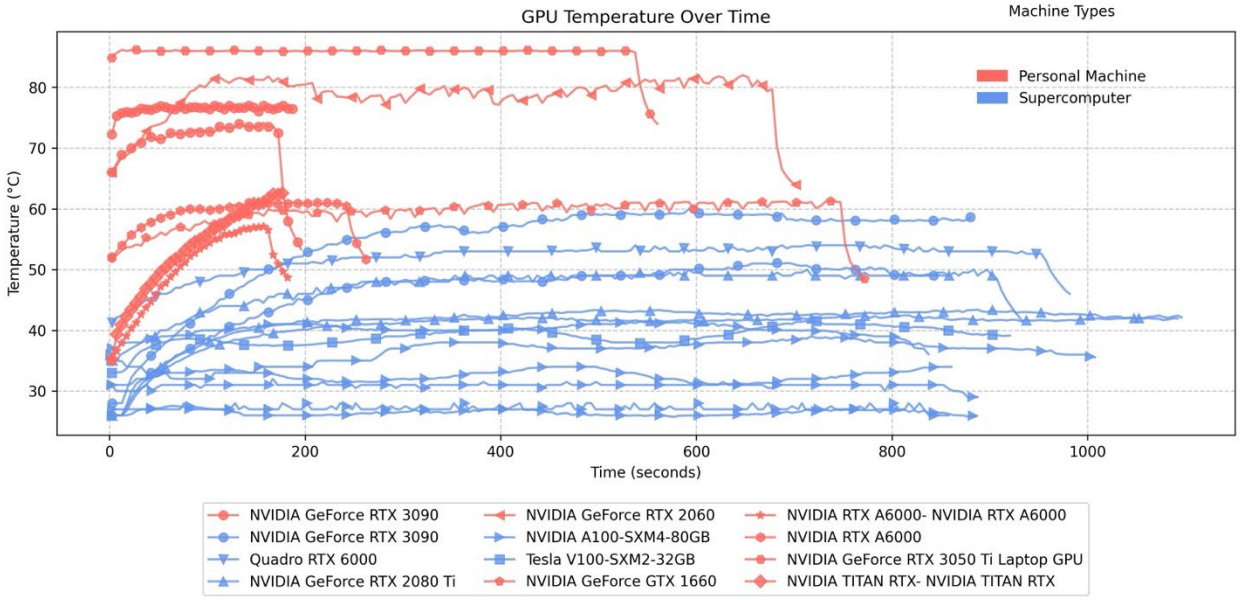


**Figure 3.** Temperature over time by GPU.

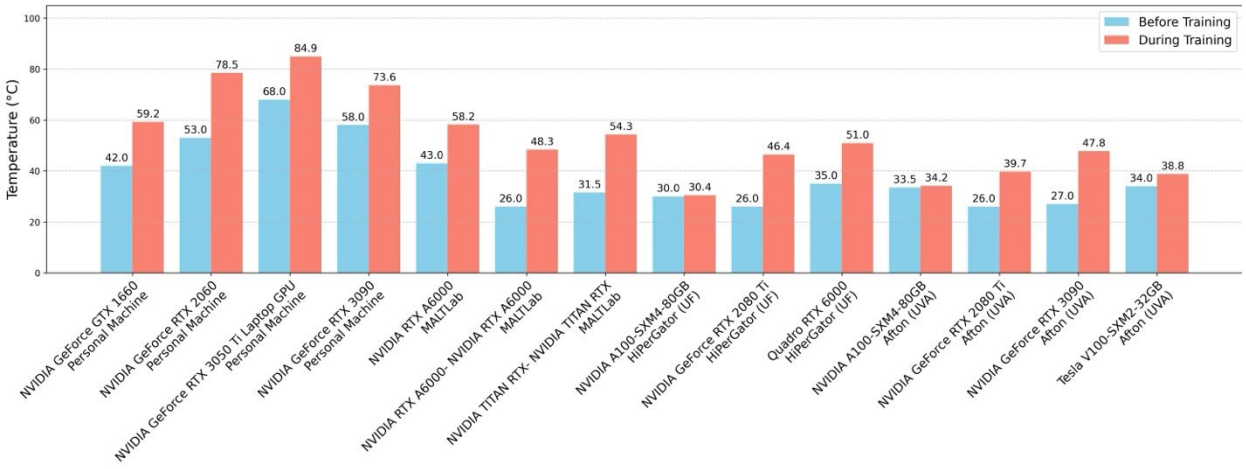


**Figure 4.** GPU temperatures before and during training.

The hottest system was the benchmark on an RTX 3050 Ti laptop. However, it should be noted that the starting temperature was over 70 degrees Celsius, which likely played a role in the high temperature during benchmarking.

The coolest system was the A100, which was tested on both HiPerGator and Afton. The A100 is one of the strongest GPUs in the HPC cluster benchmarking and stayed the coolest out of all the GPUs benchmarked.

Figure 5 illustrates the power draw before and during benchmark training. Although each system increases in power draw as training begins, which is expected due to the process demanding GPU use, the difference in power draw varies among GPUs. The smallest difference in power draw comes from the A100s, whereas the largest difference comes from the 3090 on a personal machine. While the jump in power draw was the greatest, this graphics card completed benchmarking in the shortest amount of time out of all completed benchmarks, hinting at a possible correlation between power draw and speed of completion.

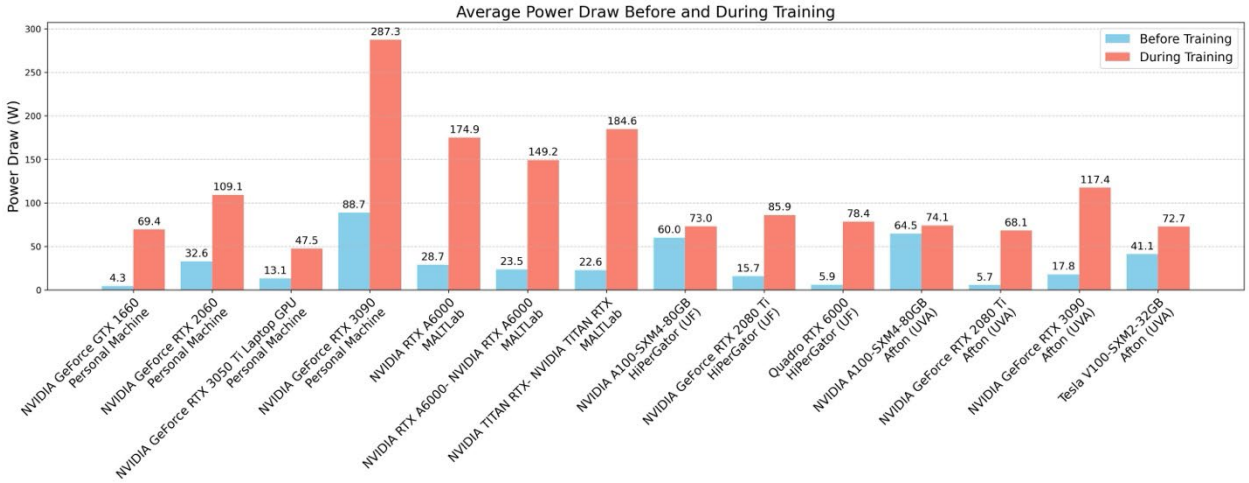


**Figure 5.** GPU Power Draw before and during training.

As shown in Figure 6, multiple GPUs significantly decrease benchmarking times. When training with multiple GPUs, batches are split among the GPUs, which decreases individual GPU load and the total time of benchmark tests. This emphasizes the benefits of using multiple GPUs when possible and convenient.

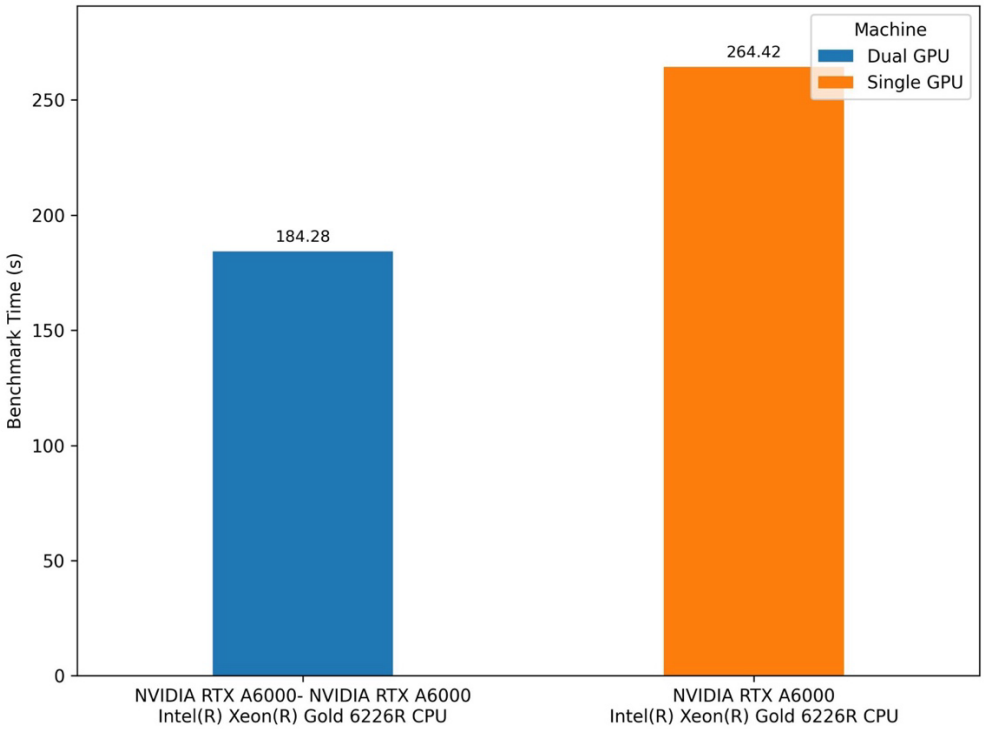


**Figure 6.** Single vs. Dual GPU Benchmark Times.

### Traffic Cars YOLOv4-Tiny versus YOLOv4-Full

Figure 7 demonstrates the difference between the YOLOv4-Tiny and YOLOv4-Full configurations when applied to a subset of the Traffic Cars validation dataset. YOLOv4-Tiny was chosen as v4 is recommended by Darknet documentation (Charette, 2025). Notably, pedestrians in groups are not properly detected with the YOLOv4-Tiny configuration. This is likely due to the smaller number of layers within the neural network; therefore, YOLOv4-Full is preferable when using the generated model in real-world analysis. While YOLOv4-Tiny models take less time to train, the importance of accurate detection is paramount in assessing near miss conflicts taking place within the intersection. For instance, a car detected twice (shown in Figure 7a) may be erroneously computed to be two separate cars intersecting each other.

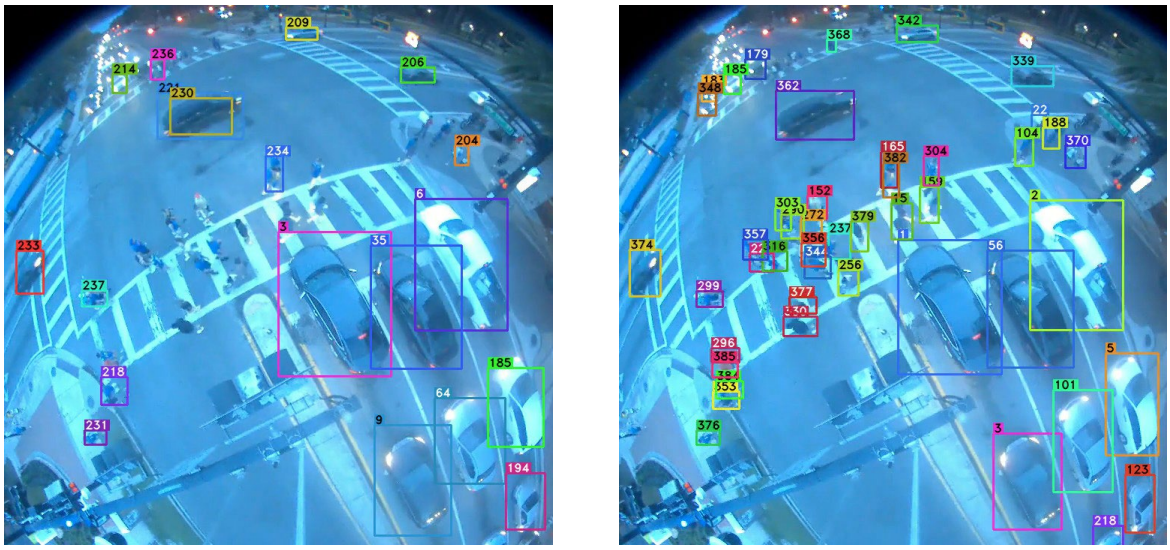

**Figure 7.** (a) YOLOv4 Tiny Comparison, (b) YOLOv4 Full Comparison.

In Figure 8, the graph shows the training process over iterations (number of batches) of the model, plotting the Mean Average Precision (red) and loss (blue). While the left plot demonstrates training on the Traffic Cars dataset with the YOLOv4-Tiny configuration, the right plot shows the YOLOv4-Full training. The model convergence, which is indicated by a drop-off in loss towards the final quarter of training, is also followed by a high mAP value that stays consistent. These patterns indicate successful model training that can immediately be used and deployed.

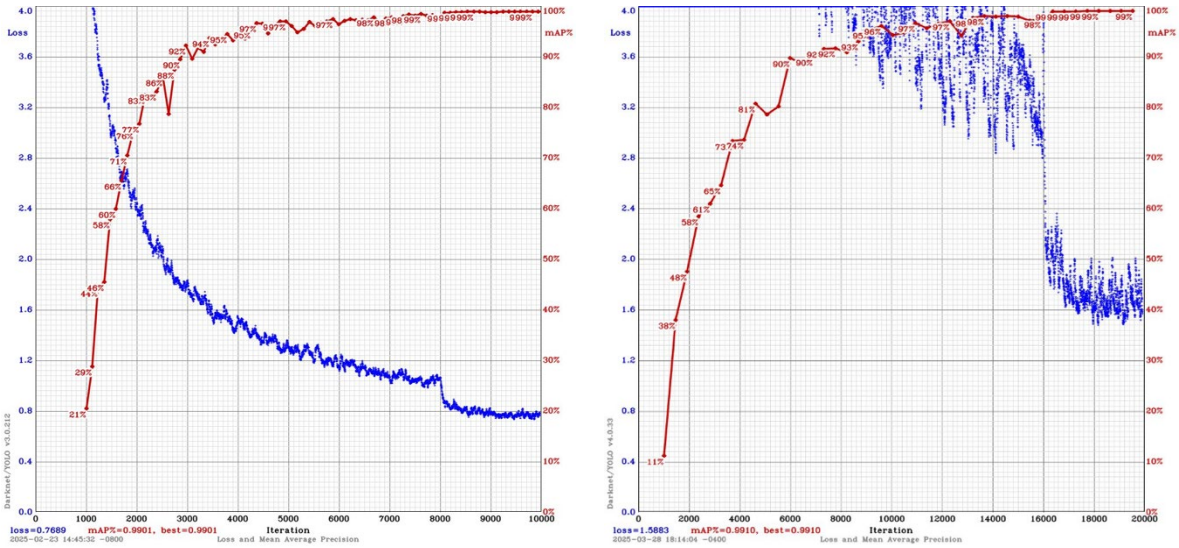

**Figure 8.** (a) YOLOv4 Tiny Comparison, (b) YOLOv4 Full Comparison.

## Conclusion

This paper has analyzed benchmarking runs across various GPUs and computing environments to reveal critical insights into the optimal configurations for training YOLO models within applications. The results suggest that the NVIDIA A100 and NVIDIA GeForce 3090 have the best performance, featuring high VRAM and processing power. HPC clusters are ideal for large-scale model training using large datasets and/or extensive training times. Personal machines showed more impressive timing results for running smaller datasets, though less thermally efficient. Additionally, the use of multiple GPUs demonstrated a significant reduction in model training times. The YOLOv4-Full model developed for the fisheye traffic camera environment achieved a 99.10% mAP across all classes. The model is preferable to the YOLOv4-Tiny configuration, especially pertaining to pedestrian detection. The containerization method for Darknet/YOLO provides an accessible avenue for researchers at other institutions to easily run and benchmark training processes for other datasets.

Future work will explore the use of other YOLO versions including YOLOv4-tiny-3L and YOLOv7. The algorithm can be further augmented to leverage DarkMark's API, generating the Darknet training files and switching configuration files for the ease of the developer.

## Acknowledgements

The authors would like to thank Stéphane Charette for providing benchmarks on a variety of powerful personal computers, as well as for promptly responding to inquiries about the Darknet/YOLO suite of software.

## References

Balcilar, M., & Sonmez, A. C. (2008). *Extracting vehicle density from background estimation using Kalman filter.* 2008 23rd International Symposium on Computer and Information Sciences, 1–5. https://doi.org/10.1109/ISCIS. 2008.4717950

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). *Simple online and realtime tracking.* 2016 IEEE International Conference on Image Processing (ICIP), 3464–3468. https://doi.org/10.1109/ICIP.2016.7533003 Bochkovskiy, A., Wang, C., & Liao, H. M. (2020). Yolov4: Optimal speed and accuracy of object detection. CoRR, abs/2004.10934. https://arxiv.org/abs/2004.10934

Charette, S. (2025). DarkMark. https://github.com/stephanecharette/DarkMark

Charette, S. (2025). Darknet faq [Most people should use YOLOv?-tiny. For example, YOLOv3-tiny is faster than YOLOv4-tiny, while YOLOv7-tiny offers lower quality output.]. https://www.ccoderun.ca/programming/darknet_faq/

Charette, S. (2025). Darknet/YOLO FAQ. https://www.ccoderun.ca/programming/darknet_faq/#optimal_network_ size

Charette, S. (2022). *Yolov10 vs. darknet/yolo comparison* [Benchmarks comparing training and inference times for YOLOv10, YOLOv3-tiny, and YOLOv4-tiny.]. https://www.ccoderun.ca/programming/darknet_faq/#configuration_template

Chiang, S.-H., Wang, T., & Chen, Y.-F. (2021). *Efficient pedestrian detection in top-view fisheye images using com- positions of perspective view patches.* Image and Vision Computing, 105, 104069. https://doi.org/10.1016/j.imavis.2020.104069

McLay, R., Schulz, K. W., Barth, W. L., & Minyard, T. (2011). *Best practices for the deployment and management of production hpc clusters*. State of the Practice Reports. https://doi.org/10.1145/2063348.2063360

NVIDIA. (n.d.). *2080ti specs.* https://www.nvidia.com/en-gb/geforce/graphics-cards/rtx-2080-ti/

NVIDIA. (2022a). *3090 specs.* https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/

NVIDIA. (2022b). A100 specs. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf

Redmon, J., & Farhadi, A. (2016). *YOLO9000: better, faster, stronger.* CoRR, abs/1612.08242. http://arxiv.org/abs/1612.08242

Sharma, A., Chakraborty, P., Hawkins, N., & Knickerbocker, S. (2019, March). *Automating Near-Miss Crash Detection Using Existing Traffic Cameras* (tech. rep. No. InTrans Project 17-619). Center for Transportation Research and Education, Iowa State University.

Siregar, M., Agah, H., & Hidayatullah, F. (2018). *Near-miss accident analysis for traffic safety improvement at a 'channelized' junction with U-turn.* International Journal of Safety and Security Engineering, 8, 31–38. https://doi.org/10.2495/SAFE-V8-N1-31-38

von Laszewski, G. (2022). *Cloudmesh GPU Monitor.* GitHub. https://github.com/cloudmesh/cloudmesh-gpu

Yang, Y. (2020, November). *FastMOT: High-Performance Multiple Object Tracking Based on Deep SORT and KLT* (Version v1.0.0). Zenodo. https://doi.org/10.5281/zenodo.4294717