Can Small Quantized VLMs Drive? An Experimental Evaluation of Small Quantized VLMs for Autonomous Driving

Samson Mathew
Undergraduate Student
Department of Computer Science
Hofstra University
Hempstead, NY, USA
smathew5@pride.hofstra.edu

Zonghua Gu
Department of Computer Science
Hofstra University
Hempstead, NY, USA
zonghua.gu@hofstra.edu

***Abstract*—Recent advancements in Foundation Models (e.g., Vision Language Models, VLMs) have sparked strong interest in their applications in Autonomous Driving. However, their practical adoption is hindered by challenges in deployment and real-time inference of large VLMs on in-vehicle computing platforms with limited hardware resources, as they are quite resource-demanding due to their sheer size. DriveBench is a recently developed large-scale benchmark dataset, designed to evaluate VLM reliability for Autonomous Driving. In this study, we evaluate a number of well-known small VLMs on the dataset, and attempt to answer the research question *Can small and quantized VLMs drive?* Although the general conclusion may be negative, this work aims to provide a useful reference for researchers and practitioners for evaluating the trade-offs when selecting an appropriate VLM for Autonomous Driving.**

***Index Terms*—Foundation Models, Vision Language Models, Model Quantization, Autonomous Driving**

I. Introduction

Foundation Models (FMs) are Deep Learning models that are pre-trained on large unlabeled datasets through self-supervision and then fine-tuned for different downstream tasks, including Large Language Models (LLMs), Vision Language Models (VLMs), Diffusion Models, etc. FMs bring generalist knowledge gained during pre-training to enable human-level intelligence. VLMs are multimodal FMs that take image and text inputs, and generate image or text outputs, with a wide range of applications such as Question Answering, Sentiment Analysis, Image Captioning, Object Recognition, Semantic Segmentation, and Image/Video Generation. Recently, researchers have been exploring the application of VLMs to Autonomous Driving (AD). VLMs promise unified perception, reasoning, and decision support by combining visual understanding with natural language. Their reasoning and interpretive capabilities allow them to work at a high level for richer scene understanding and decision support to improve safety and efficiency across a wide range of driving scenarios, especially the corner cases where training data are sparse. This makes VLMs appealing for AD tasks spanning perception, prediction, planning, and explanation [1], [2].

A key challenge of practical deployment of large VLMs is their extremely high demand for compute and memory resources. Model size may range from large-scale models with over a trillion parameters, e.g., GPT-4, to small-scale models with a few billion parameters. For example, serving a 175 billion model requires at least 350 GB of GPU memory. (The largest single GPU memory currently is 80 GB (NVIDIA A100). Multi-GPU systems can aggregate to hundreds of GBs through NVLink.) The sheer size of large VLMs poses significant hurdles to their local deployment and efficient inference on resource-constrained in-vehicle hardware platforms.

Fig. 1 illustrates deployment of VLMs for AD in cloud vs. in vehicle. Large VLMs can be deployed in the cloud, and only small-scale VLMs can be deployed in the vehicle. Cloud deployment has fast inference speed, but may experience long and unpredictable wireless network latency, even with today's 5G and 6G wireless technologies. In-vehicle deployment suffers from slow inference speed due to limited hardware resources such as GPU availability, processor speed, memory size, etc. Cloud deployment may be practical for certain non-time-critical tasks, but in-vehicle deployment is preferred for achieving real-time inference at high frame rates in safety-critical systems such as AVs. As closed-loop control systems in an AV have stringent real-time constraints to ensure safety, VLMs, whether in-cloud or in-vehicle, are not ready to be used to replace traditional closed-loop controllers. Instead, they are typically used as a slow system that runs in parallel with the fast system that runs the perception-planning-control loop at a fast rate (say 30 Hz), forming a dual-system architecture. Inspired by Kahneman's dual-process theory, the fast system provides rapid, reactive outputs, and the slow system conducts deeper reasoning, contextual analysis, and complex or novel situation handling, typically with higher computational cost and slower throughput. These systems may operate synchronously or asynchronously, sharing representations and informing each other's outputs. While this architecture helps

This work was supported by the Advanced Summer Program in Research (ASPiRe) program at Hofstra University.

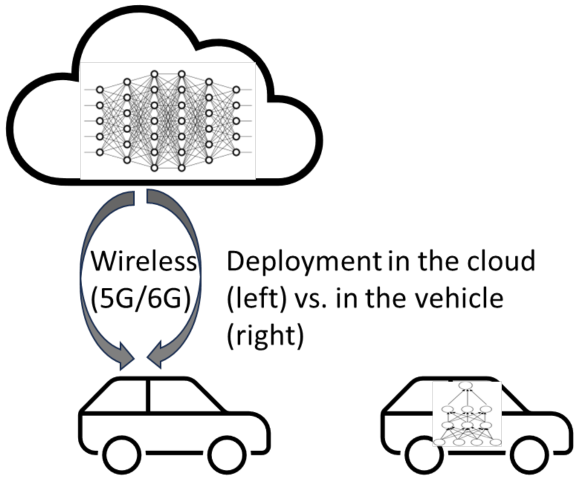


Fig. 1: Cloud deployment vs. local in-vehicle deployment.

to make the timing requirements (deadlines) less stringent for VLM inference, it is still necessary to let the VLMs run periodically and in real-time.

Since only small and efficient VLMs can be practically deployed in-vehicle, we pose the research question: *Can small and quantized VLMs drive?*, i.e., can small-scale, weight-quantized VLMs that can be practically deployed on an in-vehicle hardware platform perform well despite their small size and high efficiency? In this paper, we consider eight small-scale VLMs and their 4-bit quantized versions, and measure their performance on an extensive AD dataset, in order to answer the above research question.

This paper is structured as follows: We introduce background and related work in Section II; report our methods in Section III; present the experimental results in Section IV; and conclusions and future work in Section V.

II. Background and Related Work

A. VLMs for AD

Application scenarios of VLM for AD include perception and scene understanding [3], trajectory prediction, planning and decision-making [4], human–vehicle interaction [5], policy adaptation and rule compliance [6], etc. General-purpose VLMs can be used directly "out-of-the-box" for AD tasks, or they can be used after fine-tuning with datasets in the target application domain. For maximum performance and efficiency, many specialist VLMs for AD [1] have been developed. (We limit our attention to general-purpose VLMs in this paper, and leave the evaluation of specialist VLMs as part of future work.)

There are a number of datasets/benchmarks for AD. Two notable recent datasets for VLM evaluation are Driving with language (DriveLM) [3] and DriveBench [7]. DriveLM proposes graph VQA as a proxy for human-like reasoning with a VLM baseline that supports end-to-end policy learning, by instantiating graph-structured QA spanning perception, prediction, and planning built atop the nuScenes dataset [8] and the CARLA driving simulator. DriveBench [7] bulds upon DriveLM-nuScenes, and provides a reliability- and grounding-oriented benchmark across 17 conditions (clean, corrupted, and "no image"), revealing gaps in multimodal reasoning and robustness of VLMs for safety-critical systems. VLMs are shown to produce plausible yet weakly grounded answers, especially under degraded or missing visual inputs, underscoring the risks in applying VLM to AD domain. We use the DriveBench dataset for testing and performance evaluation in this paper (and plan to use the DriveLM dataset for model fine-tuning as part of future work).

B. Model Quantization

Model compression techniques aim to reduce the size, computation, and memory footprint of deep learning models while minimizing performance degradation. Well-known model compression techniques for LLMs/VLMs [9] include quantization, pruning, knowledge distillation, low-rank factorization, Neural Architecture Search, sparse models/Mixture-of-Experts. Model quantization is the process of reducing the precision of weights and/or activations to improve efficiency in terms of memory and inference speed. Model quantization techniques can be categorized into Post-Training Quantization (PTQ) vs. Quantization-Aware Training (QAT), or weight-only vs. weight+activation quantization.

In this paper, we use the BitsAndBytes library [10], which provides efficient 8-bit and 4-bit PTQ of model weights. BitsAndBytes "on-the-fly" quantization works by storing model weights in low precision (e.g., 4-bit or 8-bit) in memory and then dynamically dequantizing them just before computation during inference or training. This keeps memory usage low but still allows higher-precision math operations (FP16/FP32) when running matrix multiplications. PTQ with BitsAndBytes can improve computation efficiency by reducing memory footprint and data transfer overhead for faster model loading and weight movement, and enabling larger batch sizes. However, even though weights are stored in 4-bit, they are dequantized to higher precision (FP16/FP32) during computation (even on GPUs equipped with an INT8 Tensor Core such as A100 and H100), hence the matrix (GEMM) operations are still higher precision and not accelerated, and the efficiency improvement from saving memory and bandwidth is limited. (This is confirmed by our own timing measurements.) Our focus in this paper is to gauge how 4-bit quantization affects model performance, not inference time, but note that significant inference speedups may be achieved with appropriate hardware support.

III. Method

A. Selected Models

We select 8 VLMs for our experiments, based on the following selection criteria: open source; small size (as there is no universal definition of small VLMs, we select models with 12 billion parameters or less); well-known models from major companies, downloadable from Huggingface. We list the selected models in Table I, including the model name on Huggingface, the number of parameters, and amount of computation for one forward inference measured in FLOPs. For each VLM, we choose the instruction-tuned version. (Instruction tuning is a supervised learning technique used to refine VLMs by training them on labeled datasets that pair instructional prompts with desired outputs.) For each model, we also consider the 4-bit quantized models using on-the-fly weight quantization through the BitsAndBytes library [10], so a total of 16 VLMs are evaluated. The 4-bit quantized models are named by appending "-4bit" to each model name, e.g., llava-1.5-13b-4bit. We used the tool *calflops* [11] to obtain the number of parameters and FLOPs of each model, using the default input size of batch size 1 and sequence length 128, represented as (1, 128) tokens per input[1]. We adopt a batch size of 1 during inference, reflecting the real-time nature of AD tasks, where each input is processed to completion before the next input.

TABLE I: VLMs considered in this paper.

Model Name	Params	FLOPs
llava-1.5-13b [12]	13.35 B	3.29 TFLOPS
llava-1.5-7b [12]	7.06 B	1.69 TFLOPS
phi-3-vision-128k-instruct [13]	4.15 B	959.42 GFLOPS
phi-3.5-vision-instruct [13]	4.15 B	959.42 GFLOPS
gemma-3-4b-it [14]	4.3 B	993.8 GFLOPS
gemma-3-12b-it [14]	12.19 B	3.01 TFLOPS
Qwen2.5-VL-3B-Instruct [15]	3.75 B	794.83 GFLOPS
Qwen2.5-VL-7B-Instruct [15]	8.29 B	1.82 TFLOPS

B. Hardware Platform

All experiments are performed on the Star cluster (https://starhpc.hofstra.io/). a High-Performance Computing system at Hofstra University. Each compute node contains an AMD EPYC 7513 Processor, and 8 SXM NVIDIA A100 GPUs with 1024 GiB total memory (16 x 64GiB DIMM DDR4).

C. Performance Metrics

We adopt the DriveBench [7] dataset for vision-based AD, where each input consists of images from 6 cameras surrounding the vehicle: Front left, Front, Front Right, Back left, Back, Back Right. Similar to DriveBench [7] and DriveLM [3], we mainly consider the GPT score as the metric for evaluating the quality of the VLM output[2]. GPT Score is an LLM-as-a-judge evaluation approach that uses an LLM to assign a quantitative score (0-100) to generated text. We adopt OpenAI GPT-5-mini for GPT Score evaluation. GPT-5-mini is a compact version of GPT-5, released in August 2025, designed to handle lighter-weight reasoning tasks[3]. The model prompt consists of detailed rubrics that account for answer correctness, coherence, and alignment of explanations with the final answer, and we omit the details here which are described in [7].

Based on each original clean image, DriveBench provides 15 different corruption types, including weather conditions (Brightness, Dark, Fog, Snow, and Rain), external disturbances (Water Splash and Lens Obstacle), sensor failures (Camera Crash, Frame Lost, and Saturate), motion blurs (Motion Blur and Zoom Blur), and data transmission errors (Bit Error, Color Quant, and H.265 Compression). In addition, they include a black "noimage" input, i.e., when the visual information is absent and the car is essentially "driving blind". They observed that some models performed well even for the "noimage" input, suggesting that the model response might be based on majority biases (e.g., Going Ahead, the most common action in most driving scenarios) instead of camera input. For each model, we compute its *Average GPT Score* for each of the 17 input types by averaging the GPT scores across all 1,261 questions spanning four driving tasks of perception, prediction, planning, and behavior. We further compute its *Overall Average GPT Score* by averaging across all 17 input types, as an overall performance metric for each model.

IV. Experimental Results

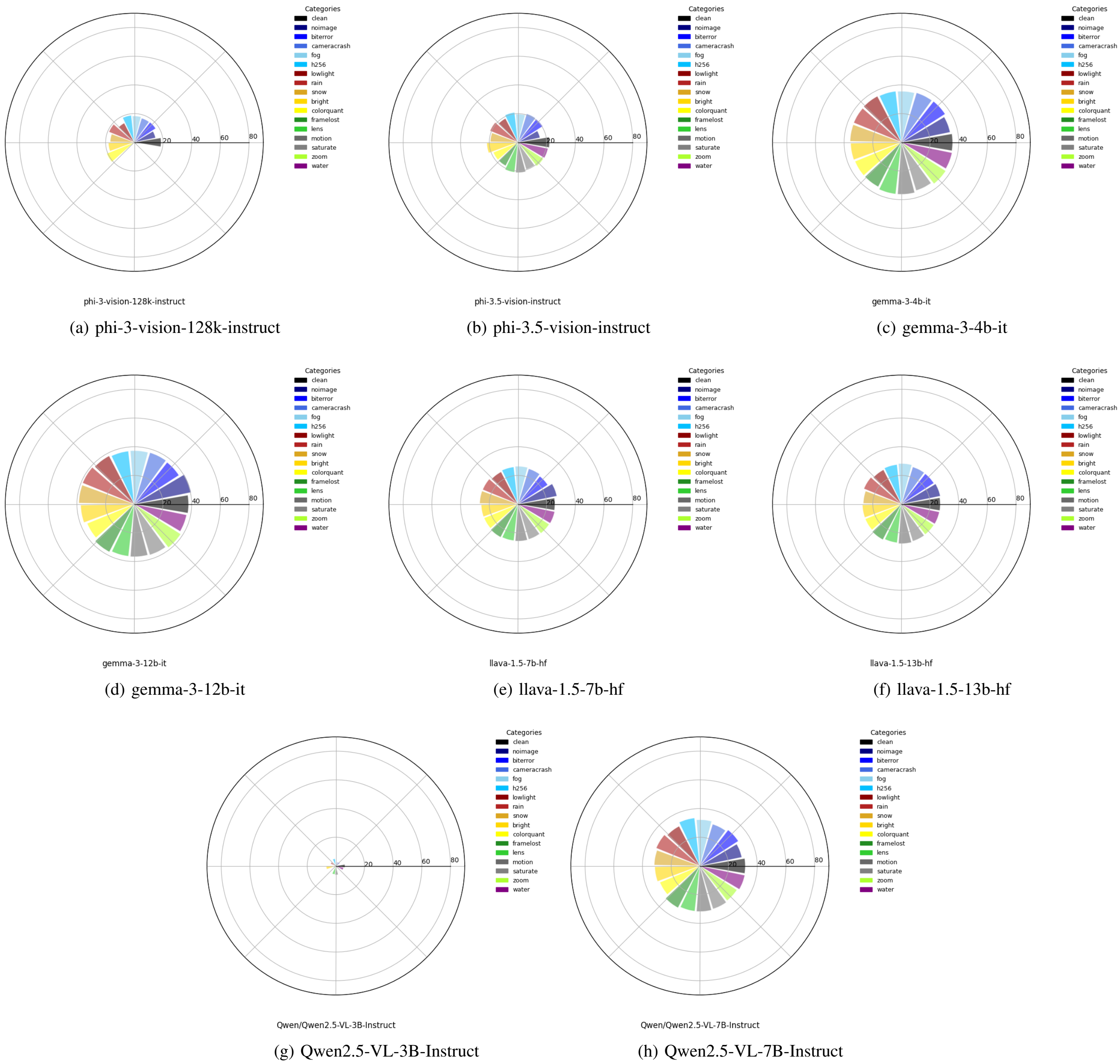
We plot Average GPT Scores of each model for the 17 input types as radar graphs (or polar graphs). Fig. 2 shows the results for full-precision models, and Fig. 3 shows results for 4-bit quantized models. (Note that phi-3-vision-128k-instruct in Fig. 2 has some missing data, since we were unable to complete the experiments before the deadline, but we will update it in the final version.) To facilitate high-level comparison among the different models, we plot the Overall Average GPT Score for each model in Fig. 4. We make the following observations from these figures:

- While larger models generally perform better, model performance is more correlated with model series (e.g., the Gemma-3 model series has the best overall performance) than with different model sizes with a series. (One exception is the Qwen2.5-VL series, where the 3B-Instruct model has significantly worse performance than the 7B-Instruct model.)
- For most models, model quantization has the effect of reducing performance, but the performance reduction is generally not very large. (We used a simple PTQ tool BitsAndBytes in this paper, and we expect the performance gap to be even smaller with sophisticated QAT algorithms.)

[1]The FLOPs numbers are intended to give a rough estimate of the compute workload, not an exact measure, intended to compare the relative computation requirement among different models, since 1) the actual batch size and number of input tokens may be different from (1, 128), as the same input may be encoded into different number of tokens by different VLMs; 2) calflops computes FLOPs based on model architecture and standard floating-point operations assuming full precision weights, but 4-bit models store weights in reduced precision. While quantization reduces the memory footprint and improves runtime efficiency, the theoretical FLOPs based on linear algebra operations stay the same since matrix shapes do not change.

[2]While some traditional language metrics based on pattern-matching between predicted responses and ground-truth answers, such as BLEU [16] and ROUGE-L [17], are also used in [7], they are shown to poorly reflect underlying key information, and GPT scores are shown to be a better metric overall.

[3]We did not use the full version of GPT-5 for budget/cost reasons, since we found that GPT-5 and GPT-5-mini give similar results for selected samples, so GPT-5-mini is adequate for our purposes.



(a) phi-3-vision-128k-instruct (b) phi-3.5-vision-instruct (c) gemma-3-4b-it

(d) gemma-3-12b-it (e) llava-1.5-7b-hf (f) llava-1.5-13b-hf

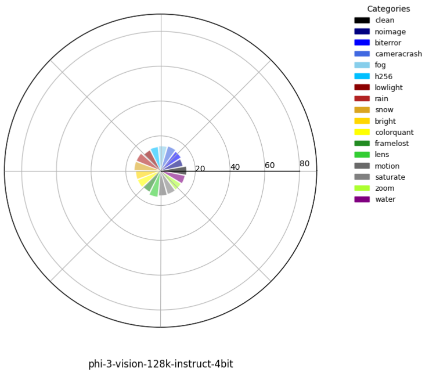
(g) Qwen2.5-VL-3B-Instruct (h) Qwen2.5-VL-7B-Instruct

Fig. 2: Radar graph of Average GPT Score for each full precision model.

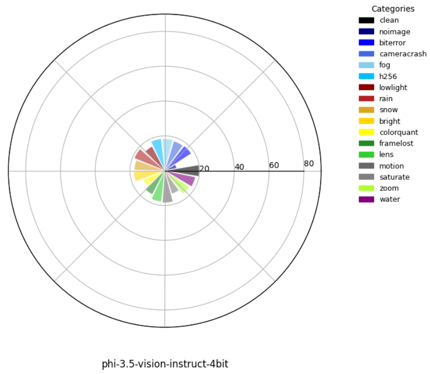
- Surprisingly, some 4-bit models achieve slightly improved performance compared to their full-precision counterparts, e.g., Gemma-3 series and Qwen2.5-VL-3B-Instruct. We do not have a good explanation for this phenomenon, except for the inherent non-determinism of large generative AI models.
- Among all 16 model variants, gemma-3-12b-it-4bit has the highest Overall Average GPT Score of 41.2, and Qwen2.5-VL-3B-Instruct has the lowest score of 3.1. Upon closer examination, we find that Qwen2.5-VL-3B-Instruct (and its 4-bit version) provides blank responses to many prompts, thus receiving a score of 0 for them.
- Even for the highest-performing model, the Overall Average GPT Score of 41.2 is far below the bar for effective driving, hence none of the 16 models is good enough for practical AD tasks.

While models considered in this paper have some overlaps with the models included in the DriveBench paper [7], our results are not comparable with the GPT scores in [7] for two reasons:

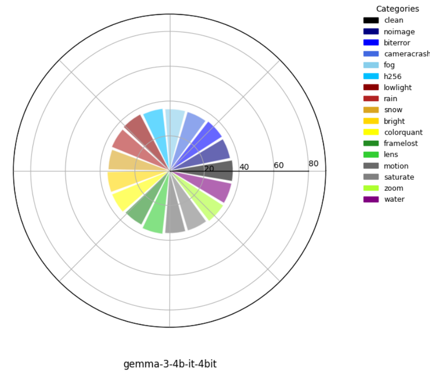
- We use the newest version of ChatGPT (GPT-5-mini) for GPT Score evaluation, whereas [7] uses GPT-3.5-turbo. They give significantly different scores for the same input.



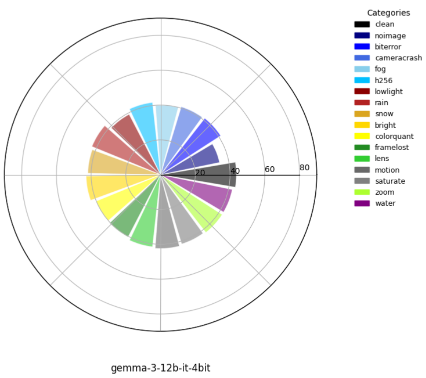
(a) phi-3-vision-128k-instruct-4bit



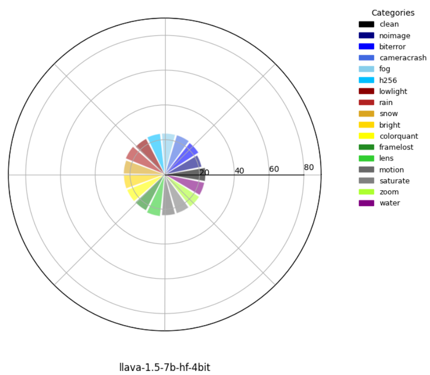
(b) phi-3.5-vision-instruct-4bit



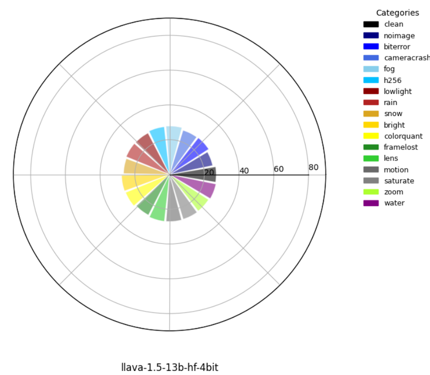
(c) gemma-3-4b-it-4bit



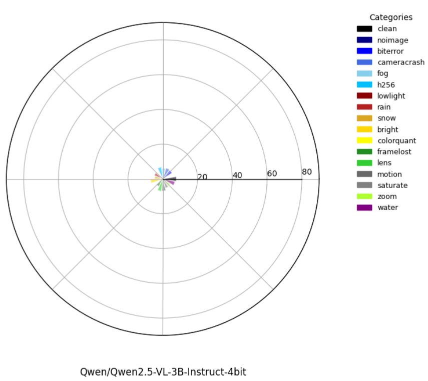
(d) gemma-3-12b-it-4bit



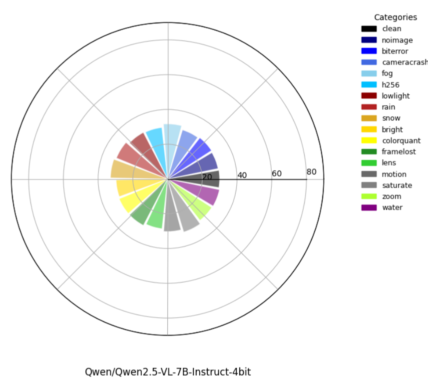
(e) llava-1.5-7b-hf-4bit



(f) llava-1.5-13b-hf-4bit



(g) Qwen2.5-VL-3B-Instruct-4bit



(h) Qwen2.5-VL-7B-Instruct-4bit

Fig. 3: Radar graph of Average GPT Score for each model for each 4-bit quantized model.

In our experiments, we found that GPT-3.5-turbo frequently hangs during evaluation of some models with poor output quality, e.g, with repeated phrases, but GPT-5-mini does not have this issue.

- The radar graphs in Figure 25 in [7] are actually the GPT scores for the Planning task, not the multi-task averages as stated in the paper, since the plotted values are inconsistent with the average values derived from Table 2 in [7].

V. Conclusions and Future Work

In this paper, we evaluated a diverse set of small and quantized VLM models on a comprehensive dataset DriveBench, across clean and corrupted conditions for 4 common driving tasks. The answer is broadly negative at present: out-of-the-box small and quantized VLMs are generally not sufficiently reliable to be used for AD tasks safely. Nevertheless, certain VLMs (e.g., Gemma-3 series) achieve good performance despite their small size, and are strong candidates for continued research to improve their performance and efficiency. Model quantization may result in a small reduction in performance,

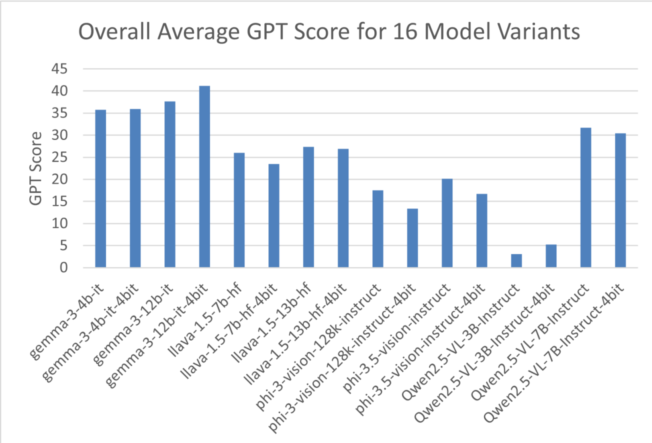


Fig. 4: Overall Average GPT Score for each model, obtained by further averaging the scores across all 17 input types for each Radar graph in Figs. 2 and 3.

but the performance reduction is generally insignificant, and sometimes it may even help to improve performance. Given proper hardware support, model quantization can result in significant inference speedups, hence it is a promising technique for improving model efficiency with small performance degradation.

Open challenges and directions include:

- Domain alignment with fine-tuning: Model fine-tuning of VLMs on domain-specific datasets can help improve grounding and reduce hallucinations for that domain, e.g., driving tasks such as perception, prediction, and planning. We plan to consider Parameter Efficient Fine-Tuning (PEFT) techniques like LoRA (Low-Rank Adaptation) [18] for fine-tuning LLMs/VLMs without retraining all their weights.
- Specialist models vs. general-purpose models: while this paper focused on general-purpose VLMs, we plan to consider specialist VLMs in the future, which can achieve good performance with relatively small sizes [1].
- Efficient perception: Visual token pruning and salience-guided attention can reduce computing time while prioritizing safety-critical cues, e.g., foreground objects such as cars and pedestrians should be given higher priority than background objects such as trees, grass and the sky [19].
- Vision-Language-Action (VLA) [2]: VLA is the next-generation evolution beyond VLM in AD, extending VLM by coupling vision-language understanding with actionable driving control and enabling a more complete and end-to-end framework for AD.

Acknowledgments

The authors would like to thank the developers of DriveBench for their help and support in resolving some technical issues.

Declaration

I acknowledge the use of AI tools, including ChatGPT and Grammarly, to help improve the grammar of this paper.

References

[1] X. Zhou, M. Liu, E. Yurtsever, B. L. Zagar, W. Zimmer, H. Cao, and A. C. Knoll, "Vision language models in autonomous driving: A survey and outlook," *IEEE Transactions on Intelligent Vehicles*, 2024.

[2] S. Jiang, Z. Huang, K. Qian, Z. Luo, T. Zhu, Y. Zhong, Y. Tang, M. Kong, Y. Wang, S. Jiao *et al.*, "A survey on vision-language-action models for autonomous driving," *arXiv preprint arXiv:2506.24044*, 2025.

[3] C. Sima, K. Renz, K. Chitta, L. Chen, H. Zhang, C. Xie, J. Beißwenger, P. Luo, A. Geiger, and H. Li, "Drivelm: Driving with graph visual question answering," in *European conference on computer vision*. Springer, 2024, pp. 256–274. [Online]. Available: https://opendrivelab.com/DriveLM

[4] C. Pan, B. Yaman, T. Nesti, A. Mallik, A. G. Allievi, S. Velipasalar, and L. Ren, "Vlp: Vision language planning for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14 760–14 769.

[5] T. Deruyttere, S. Vandenhende, D. Grujicic, L. Van Gool, and M.-F. Moens, "Talk2car: Taking control of your self-driving car," *arXiv preprint arXiv:1909.10838*, 2019.

[6] B. Li, Y. Wang, J. Mao, B. Ivanovic, S. Veer, K. Leung, and M. Pavone, "Driving everywhere with large language model policy adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14 948–14 957.

[7] S. Xie, L. Kong, Y. Dong, C. Sima, W. Zhang, Q. A. Chen, Z. Liu, and L. Pan, "Are vlms ready for autonomous driving? an empirical study from the reliability, data, and metric perspectives," *arXiv preprint arXiv:2501.04003*, 2025. [Online]. Available: https://drive-bench.github.io

[8] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[9] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang, "A survey on model compression for large language models," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 1556–1577, 2024.

[10] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm. int8 (): 8-bit matrix multiplication for transformers at scale," *arXiv preprint arXiv:2208.07339*, 2022.

[11] xiaoju ye. (2023) calflops: a flops and params calculate tool for neural networks in pytorch framework. [Online]. Available: https://github.com/MrYxJ/calculate-flops.pytorch

[12] H. Liu, C. Li, Y. Li, and Y. J. Lee, "Improved baselines with visual instruction tuning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024, pp. 26 296–26 306.

[13] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann *et al.*, "Phi-4 technical report," *arXiv preprint arXiv:2412.08905*, 2024.

[14] G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière *et al.*, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

[15] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang *et al.*, "Qwen2. 5-vl technical report," *arXiv preprint arXiv:2502.13923*, 2025.

[16] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[17] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[18] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[19] J. Cao, Q. Zhang, P. Jia, X. Zhao, B. Lan, X. Zhang, X. Wei, S. Chen, Z. Li, Y. Wang *et al.*, "Fastdrivevla: Efficient end-to-end driving via plug-and-play reconstruction-based token pruning," *arXiv preprint arXiv:2507.23318*, 2025.

[20] A. Marafioti, O. Zohar, M. Farré, M. Noyan, E. Bakouch, P. Cuenca, C. Zakka, L. B. Allal, A. Lozhkov, N. Tazi *et al.*, "Smolvlm: Redefining small and efficient multimodal models," *arXiv preprint arXiv:2504.05299*, 2025.

APPENDIX

We provide a representative example of input (images and prompt) and outputs by different models. The MCQ text prompt is: "What is the moving status of object <c2, CAM_BACK,0.4610,0.5841>? Please select the correct answer from the following options: A. Going ahead. B. Turn left. C. Turn right.", and the ground truth answer is "C". (Note that only the back camera image CAM_BACK is used in this prompt, and the other 5 images are ignored.) The object referred to in the prompt is a car, highlighted in a red circle in Fig. 5. However, one can observe that the scenario is ambiguous: the car is not stopped at a traffic light; it is driving on a road bending right, so the car can be reasonably said to be either going ahead, or turning right.

Here are outputs from the different VLMs:

- *gemma-3-4b-it*: The moving status of object <c2,CAM_BACK,0.4610,0.5841> is A. Going ahead. Explanation: In the CAM_BACK image, there is a construction crane moving from left to right. This indicates that the object is moving ahead[4].
- *gemma-3-12b-it*: The correct answer is A. Going ahead. Explanation: Based on the CAM_BACK image, the object at coordinates (0.4610, 0.5841) appears to be a car moving straight ahead on the road. There is no indication of a left or right turn. Therefore, the most appropriate description of its movement is "Going ahead.".
- *llava-1.5-7b-hf answer*: A. Going ahead.
- *llava-1.5-13b-hf answer*: A. Going ahead.
- *Qwen2.5-VL-3B-Instruct*: B. Turn left.
- *Qwen2.5-VL-7B-Instruct*: A. Going ahead. The object at <c2,CAM_BACK,0.4610,0.5841> appears to be a vehicle moving forward on the road, as indicated by its position and the direction of the arrow on the road. There is no indication of a turn left or right. Therefore, the correct answer is A. Going ahead.

In addition, we show an interesting output from a very small VLM with only 256 million parameters, SmolVLM-256M-Instruct [20] (We did not include this model in our experiments due to its small size and poor performance.): " C. Turn right. Answer: C. Turn right. Explanation: The image shows a grassy area with a tree on the left side of the image. The tree has a few branches and leaves, indicating it is in a season with green foliage. The grass is green and appears healthy, suggesting it is well-maintained. The presence of a tree suggests that the area is likely a park or a green space, which is a common feature in urban areas. The image does not show any vehicles or pedestrians, which are common indicators of a safe and well-maintained environment. The image does not show any vehicles or pedestrians, which are common indicators of a busy or crowded area. The image does not show any people or animals, which are common indicators of a safe and clean environment...(additional repetitive sentences omitted.)"

Even though it gave the correct choice of C, the explanation focuses on irrelevant objects such as trees and grass, and it incorrectly states that "The image does not show any vehicles or pedestrians." There are additional repetitive sentences that are omitted due to space constraints. This failure case is likely due to its small size, and lack of driving-related data in its training dataset.

[4]This explanation does not make sense, since there is no construction crane in the Back image. There is one in the Front left image, but there is no logical correlation between its position and the car's movement direction in the Back image.

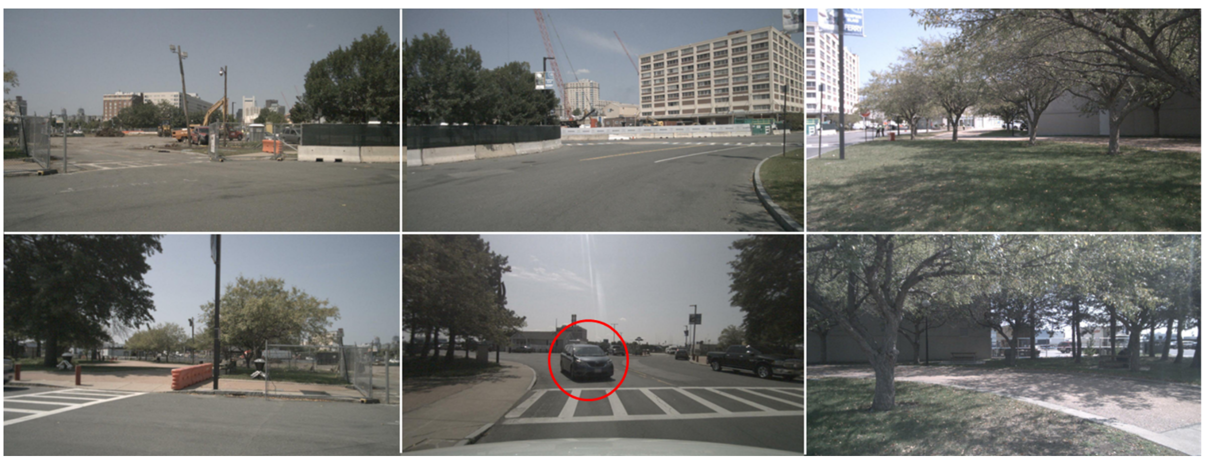


Fig. 5: Input images from 6 cameras surrounding the vehicle: Front left, Front, Front Right, Back left, Back, Back Right.