International Journal of Secondary Computing and Applications Research

Volume 2, Issue 1

EDITOR-IN-CHIEF: DR. MARIA HWANG

APRIL, 2025

Proceedings of International Journal of Secondary Computing and Applications Research, Vol 2, Issue 1

April 21, 2025

Letter from the Editor-in-Chief

Welcome to the latest issue of the *International Journal of Secondary Computing and Applications Research.*

This volume highlights the creativity, rigor, and ambition of high school researchers exploring the frontiers of computing. From theory to application, the range of topics reflects the increasingly sophisticated role that young scholars are playing in shaping the future of the field.

We hope these papers spark new ideas, encourage collaboration, and reaffirm the immense potential of high school students as contributors to serious computing research. As Editor-in-Chief, I am proud to showcase the remarkable work of our authors and the growing community that supports them.

Thank you for reading– and we welcome your thoughts as we continue to grow.

Sincerely,
Maria Hwang
Editor-in-Chief

Volume 2, Issue 1 April 21, 2025
DOI: 10.5281/zenodo.15081073 | https://ijscar.org/pubs/volume2/issue1

Contents

Comparative Analysis of Vision Transformer and ResNet50 for Glaucoma Detection: Balancing Performance and Efficiency

Eric Hwang
09eric.hwang@dulwich-seoul.kr
Dulwich College Seoul
Seoul, Republic of Korea

Abstract

This study develops and evaluates the performance and computational efficiency of ResNet50 (CNN-based architecture) and Vision Transformer (Transformer-based architecture) for detecting glaucoma from fundus photographs. Glaucoma is one of the leading causes of irreversible blindness that affects millions of people around the world. Using deep learning methods, both models are trained to learn indicators of glaucomatous fundus photographs, such as thinning of the retinal nerve fiber layer (RNFL) and nasalization of blood vessels, to classify healthy and glaucomatous eyes. Gradient-weighted Class Activation Mapping (Grad-CAM) was used to interpret the model predictions by visualizing the regions of fundus photographs that contributed most significantly to the classification. With a publicly available dataset, we fine-tuned both models by leveraging transfer learning with a small learning rate (0.0001) on pre-trained layers. Both models were assessed with metrics such as accuracy, F1 score, inference time, throughput, and maximum GPU memory usage under controlled conditions. ResNet50 outperformed ViT, achieving higher accuracy (90.72% vs 87.64%) and an F1 score (0.9104 vs 0.8614) while being significantly more computationally efficient, with 68.48% faster inference and 50.52% lower GPU usage. These findings highlight the suitability of ResNet50 over ViT for use in resource-constrained medical settings to assist ophthalmologists diagnosing patients, as it effectively balances performance and efficiency.

Keywords

Artificial Intelligence, Computer Vision, Deep Learning, Neural Networks, Transfer Leanring, Medical Imaging, Glaucoma Detection

ACM Reference Format:
Eric Hwang. 2025. Comparative Analysis of Vision Transformer and ResNet50 for Glaucoma Detection: Balancing Performance and Efficiency . In *Proceedings of International Journal of Secondary Computing and Applications Research (IJSCAR VOL. 2, ISSUE 1).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.5281/zenodo.15149831

1 Introduction

Glaucoma is a chronic eye disease that is one of the leading causes of blindness and affects more than 80 million people around the world, with early diagnosis and treatment being the most important steps to prevent vision loss [7]. However, traditional approaches to manually diagnosing glaucoma can be time consuming and subject to high variability as they are based on subjective analysis [6]. As a result, these limitations can lead to delayed diagnosis, particularly in high-footfall and resource-constrained medical settings where ophthalmologists are in short supply.

With recent advancements in Artificial Intelligence (AI), Deep Learning (DL) models have emerged as transformative tools for medical imaging, offering accurate and time-efficient disease diagnosis. Architectures based on Convolutional Neural Networks (CNNs) have set the benchmark for medical image classification, while Vision Transformers (ViTs) [2] showcase the potential to outperform CNN-based architectures. Despite advancements in AI for healthcare, few studies have systemically compared CNNs and ViTs on both computational efficiency and performance for glaucoma detection. Along with performance itself, computational efficiency is also a crucial metric, often overlooked in previous research due to its focus on surpassing human expertise, particularly when evaluating a model's suitability to be used in resource-constrained medical environments, which is where the implementation of an automated diagnosis system should be of a higher priority.

Therefore, to address this gap, our study evaluates ResNet50 [4] and ViT using a collection of publicly available fundus photograph datasets. By leveraging transfer learning and fine-tuning both architectures, we rigorously analyze performance metrics (accuracy, F1 score) and computational efficiency metrics (inference time, throughput, maximum GPU memory usage) to determine their suitability for being implemented as an assistive tool for glaucoma diagnosis. Through this analysis, we establish a benchmark for deploying deep-learning tools for retinal imaging in healthcare, specifically tailored for glaucoma detection in diverse clinical settings with implications for broader medical applications.

2 Related Work

In the medical field, CNNs traditionally have performed very well in medical image classification and segmentation. In 2020, a comprehensive study by García et al. assessed two CNN architectures using a circumpapillary Optical Coherence Tomography (OCT) test dataset (glaucomatous: 93, healthy: 156 B-scans), concluding that fine-tuned CNNs, especially VGG architectures achieved a high area under the ROC curve [3]. Similarly, a study by Kulkarni and Ahmed evaluated a more exhaustive list of CNNs and histogram equalization techniques using a small (glaucomatous: 396, healthy: 309) dataset, with VGG-19 achieving the highest accuracy (97.9%) and ResNet architectures achieving the highest specificity score (98.4%) [8]. These studies highlight the effectiveness of CNN architectures in glaucoma detection.

IJSCAR VOL. 2, ISSUE 1

Recently, there has been active research on utilizing ViTs for medical imaging. A study by Nurgazin et al. demonstrated that ViT with ProtoNet, a few-shot learning algorithm, can achieve competitive performance against state-of-the-art CNN-based approaches in medical benchmark datasets [10]. More specifically for glaucoma detection, Hwang et al. demonstrated that ViTs perform comparably to CNNs in glaucoma detection using fundus photographs (total: 3,142), achieving a comparable area under the ROC curve and accuracy. They concluded that ViTs are preferable when prioritizing sensitivity over minimizing false positives [5].

Overall, the existing literature underscores the efficacy of both CNNs and ViTs in medical imaging, with emerging evidence suggesting that ViTs may offer competitive, if not superior, performance in medical imaging. While a comparative analysis of diagnostic performance exists such as [5], a comprehensive evaluation that incorporates computational metrics for glaucoma detection and considers the constraints of resource-limited clinical settings is lacking. Addressing this gap is crucial for informing the selection of appropriate models for deployment in varying healthcare environments.

Furthermore, although these studies demonstrate the effectiveness of CNNs and ViTs, their small dataset sizes limit their generalizability to larger, more diverse clinical populations. By leveraging a significantly larger collection of fundus photograph datasets (total: 7815 without sampling), this study addresses this limitation, providing a more rigorous evaluation of CNN and Vision Transformer (ViT) architectures. This larger dataset enables more robust performance comparisons and enhances the applicability of findings to real-world clinical settings.

Finally, while [5] used a collection of datasets separately for comparison and without balancing the dataset, which yielded valuable insights on architectural differences, we balanced the dataset using sampling techniques with data augmentations to ensure equitable representation of glaucomatous and healthy cases, reducing potential biases and improving the reliability of the model evaluation. This methodological enhancement strengthens the robustness of the results, making them more applicable to diverse clinical settings.

3 Methods

3.1 Data Collection and Preprocessing

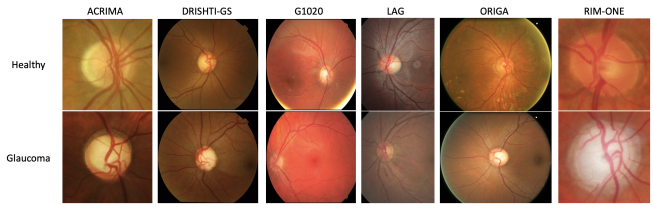


Figure 1: Sample Images of Fundus Photographs from Each Dataset

The dataset used for this study is a collection of 6 public fundus photograph datasets, downloaded from Kaggle [11] which consists of the following: ACRIMA, DRISHTI-GS, G1020, LAG, ORIGA, and RIM-ONE [1]. Using a collection of datasets increases the total dataset size and also increases data diversity which ensures that the models learn to generalize to variations such as differences in fundus photography protocols, imaging equipment, and patient demographics.

In total, There are 7815 images (healthy: 5002, glaucoma: 2813) of fundus photographs. Figure 1 demonstrates sample fundus photographs from each dataset. Since there is a dataset imbalance, with a higher number of healthy eye scans, random oversampling was applied to balance the dataset, preventing the models from becoming biased toward the majority class. This step is critical in the medical context because false negatives (failing to identify glaucoma correctly) can have severe consequences for patients as the early detection of glaucoma is essential. With random oversampling, the dataset becomes (healthy: 5002, glaucoma: 5002).

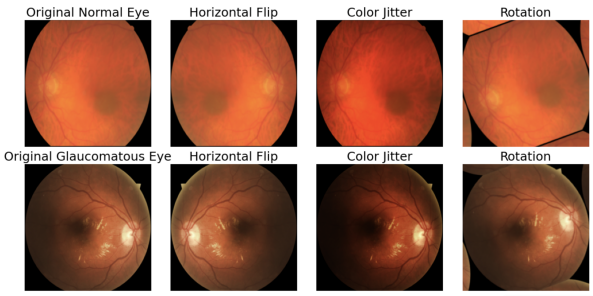


Figure 2: Sample Images of Data Augmentation Methods

As shown in Figure 2, the multiple data augmentation methods (horizontal flip, random rotation, and color jitter) were used to simulate real-world variations in retinal imaging conditions while also increasing the total dataset size which prevents overfitting. Horizontal flips simulate eye symmetry, rotations mimic misalignment during image capture, and color jitter compensates for variations in imaging equipment and lighting conditions. As a result, it makes the models more robust, which is especially important in resource-constrained environments where imaging quality may vary significantly.

Moreover, All images were resized to 224x224 pixels to standardize input dimensions which allows compatibility with ResNet50 and ViT architectures. Furthermore, the fundus photograph can still preserve the indicators of glaucoma, such as nasalization of the blood vessels and thinning of the retinal nerve fiber layer (RNFL) [9], while optimizing computational efficiency, a key consideration for deployment in resource-constrained medical environments. Finally, the dataset was randomly split into training data and validation data with a 75:25 ratio which allows sufficient data for training while having enough data for an unbiased validation to evaluate the metrics.

3.2 Model Selection and Modification

Since ResNet achieved the highest specificity in [3] and is widely researched in medical classification tasks such as ResNet-22 for breast cancer classification [14], ResNet50, an architecture more robust

Figure 3: ResNet50 Model Schematic (Illustration inspired from [12])

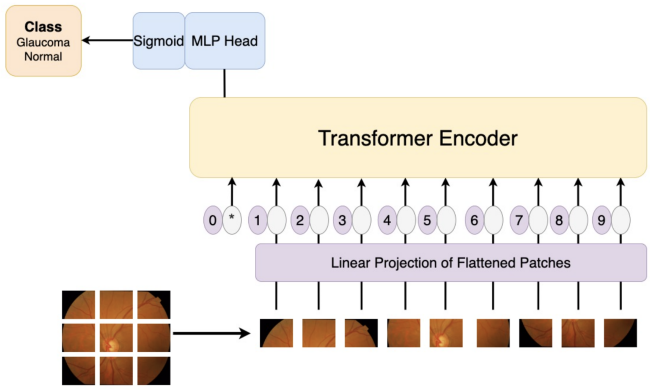


Figure 4: ViT Model Schematic (Illustration inspired from [2])

than the original ResNet34, was selected as a representative CNN model against ViT. This comparison highlights the strengths and limitations of convolutional and transformer-based architectures in glaucoma detection.

Both models were initialized with ImageNet pre-trained weights and then were trained with the glaucoma dataset to facilitate transfer learning. Their entire network was fine-tuned with a small learning rate to allow them to adapt to the specific task of detecting glaucoma from general image classification.

The activation function of the output layer was changed to a sigmoid function as the task is binary classification. Furthermore, it makes the models output a prediction between 0 and 1 for detecting glaucoma, which allows the model to be used as a screening tool or to assist ophthalmologists in diagnosing patients.

L2 Regularization (weight decay) was used as this value minimized overfitting by penalizing large weights which encourages simpler model representations. Regularization is particularly crucial in medical imaging tasks, where preserving the model's ability to generalize across diverse patient data is vital.

Furthermore, The number of epochs was set to 10 for both models as during preliminary experiments, ViT's training and validation loss began diverging, indicating overfitting, after the 10th epoch due to the relatively small dataset and transformer-based architecture's high capacity for pattern memorization. Hence, by limiting the number of epochs, we balanced the models' learning capacity and generalization.

3.3 Metrics

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy and F1 score were chosen as primary metrics for measuring model performance as accuracy provides an overall measure of each model's ability to accurately detect glaucoma from fundus photographs. Furthermore, the F1 score balances sensitivity and precision which is crucial in medical classification tasks.

- Inference: time taken for the model to make a prediction on a given input, measured in seconds.
- Throughput: number of predictions made by the model per unit of time, measured in images/second.
- Max GPU Memory Usage: the peak amount of GPU memory used during training, measured in MB.

On the other hand, Computational efficiency was assessed using inference time per image, throughput (images processed per second), and maximum GPU memory usage during inference. These metrics were chosen to evaluate the feasibility of deploying these models in medical environments with restricted computational resources or a high density of patients.

3.4 Controls

Both models were trained in equal hardware and software conditions. Both were run on Google Colaboratory with T4 GPU (High-RAM 51GB) as a hardware accelerator. Equal versions of Python (3.10.12) and Pytorch (2.3.1+cu121) were used to prevent discrepancies. Moreover, the same data set was used with the same ratio of train-validation split.

In terms of model design, the pre-trained weights for both models were generated using the ImageNet dataset which gives them a fair baseline to begin transfer learning. Both models were trained with Binary Cross Entropy Loss (BCELoss) as the loss function and Adam optimizer as the optimization algorithm to ensure they are both minimizing the same loss function and optimizing towards the same goal.

Overall, the following hyperparameters were kept constant for equal training conditions.

- Epoch: 6
- Learning Rate: 0.0001
- L2 Regularization - Weight Decay: 0.0001
- Batch Size: 32

4 Results

Epoch	Train Loss	Validation Loss	Accuracy	F1	Inference Time	Throughput	GPU usage
1	0.3341	0.3852	82.37%	0.8301	8.4035 s	297.61 img/s	782.34 MB
2	0.2243	0.2287	88.56%	0.8844	8.4699 s	295.28 img/s	782.04 MB
3	0.1764	0.2181	89.40%	0.8912	8.4621 s	295.55 img/s	782.04 MB
4	0.1602	0.2385	89.00%	0.8907	8.4657 s	295.43 img/s	782.04 MB
5	0.1392	0.2020	89.96%	0.8984	8.4778 s	295.01 img/s	782.04 MB
6	0.1116	0.2034	90.72%	0.9104	8.4417 s	296.27 img/s	782.04 MB

Table 1: Results of ResNet50 Training and Validation

Epoch	Train Loss	Validation Loss	Accuracy	F1	Inference Time	Throughput	GPU usage
1	0.4704	0.3387	84.69%	0.8340	26.85 s	93.14 img/s	1580.54 MB
2	0.2972	0.2621	87.60%	0.8693	26.80 s	93.32 img/s	1580.54 MB
3	0.2403	0.2207	88.64%	0.8822	26.83 s	93.21 img/s	1580.54 MB
4	0.2145	0.2359	88.68%	0.8801	26.86 s	93.13 img/s	1580.54 MB
5	0.1993	0.2588	88.04%	0.8781	26.86 s	93.12 img/s	1580.54 MB
6	0.1925	0.2612	87.64%	0.8614	26.75 s	93.49 img/s	1580.54 MB

Table 2: Results of ViT Training and Validation

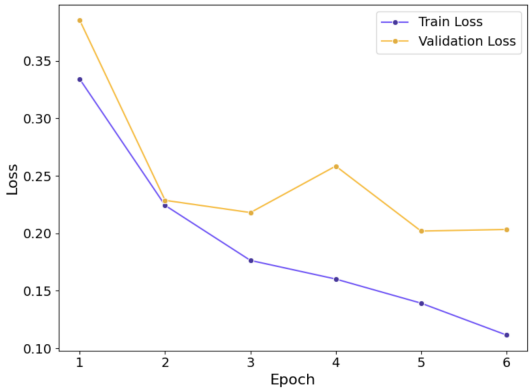


Figure 5: Train and Validation Loss over Epoch, ResNet50

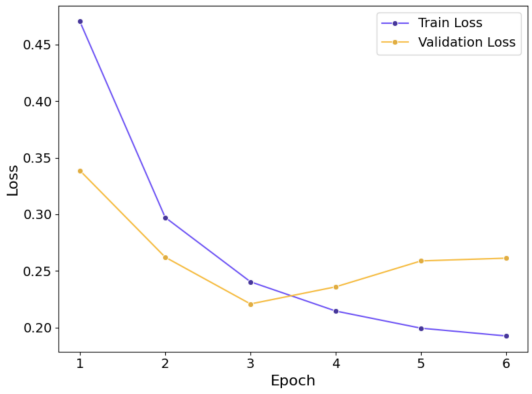


Figure 6: Train and Validation Loss over Epoch, ViT

5 Discussion

5.1 Training and Performance Analysis

For ResNet50, the training loss steadily decreases from 0.3341 to 0.1116 over 6 epochs, and the validation loss also decreases from 0.3852 to 0.2034 with only a slight increase from Epoch 3 to 4. Therefore, it indicates that ResNet50 generalized well without significant overfitting issues. For ViT, the training loss drops sharply from 0.4704 to 0.2972 over the first 2 epochs but decreases much more slowly after reaching the final training loss of 0.1925. Its validation loss initially decreases from 0.3387 to 0.2207 for the first 3 epochs but increases slightly afterward to reach 0.2612 at the final epoch. This indicates that ViT had slight overfitting at later epochs as validation loss starts to increase while train loss decreases but overall, it generalized quite well.

Furthermore, ResNet50 achieved a 90.72% accuracy for the final epoch with an F1 score of 0.8614. This demonstrates that the model had a very good performance as it predicted most of the dataset correctly without overly predicting one side, indicated by a high F1 score. Although ViT also had a strong performance with 87.64% accuracy and 0.8614 F1, ResNet50 exceeded ViT in performance metrics with 3.08% higher accuracy and 0.049 higher F1. It can also be observed that the accuracy of ViT stabilizes from Epoch 2 onward which implies that the model is converging where it is no longer learning more information from the given dataset. Lastly, the average of each computational metric over 6 epochs was calculated as shown in Table 3.

5.2 Computational Efficiency and Practical Implications

Model	Average Inference Time	Average Throughput	Max GPU Usage
ResNet50	8.45 s	295.86 img/s	782.09 MB
ViT	26.83 s	93.24 img/s	1580.54 MB

Table 3: Comparison of Computational Metrics

In general, ResNet50 was 68. 48% faster in inference making and predicted 217. 31% more images/second while using 50. 52% less GPU, outperforming ViT also in computational metrics. The difference between the two models was significant, where ViT was using approximately double the GPU memory as ResNet50 while making predictions much slower. Furthermore, ResNet50 even exceeded ViT in performance metrics.

The big gap in computational efficiency between ResNet50 and ViT highlights that ResNet50 is much better suited to be deployed in resource-constrained medical environments such as rural clinics and mobile diagnostic units or high-footfall environments. While ViT showcased promising results for diagnostic performance, its computational inefficiency directs to the need for further optimizing Transformer-based models for real-world applications.

The computational inefficiency of ViT is likely caused by the self-attention mechanisms which enables global feature extraction, but leads to quadratic complexity. This explains the significantly higher GPU memory usage and slower inference time of ViT as ResNet50 has more linear computation process. Furthermore, the ViT used in the investigation, ViT_B_16, has notably higher number of parameters with 86,567,656 [16] compared to 25,557,032 parameters [15] of ResNet50. This likely explains the slight overfitting for the last 3 epochs as it could've begun memorizing the dataset.

5.3 Grad-CAM Visualization

Figure 7 demonstrates the Gradient-weighted Class Activation Mapping (Grad-CAM) [13] visualization for ResNet50, mapping different regions on the fundus photograph that the model identified as most influential in detecting glaucoma. While the optical nerve head is visible on the activation map, the model appears to focus more on the surrounding peripapillary regions which suggests that it relies on subtle retinal features beyond the optical nerve head. This Grad-CAM visualization provides insights to the model's decision-making process. The focus on peripapillary regions in the Grad-CAM visualization could explain why ResNet50 outperformed ViT

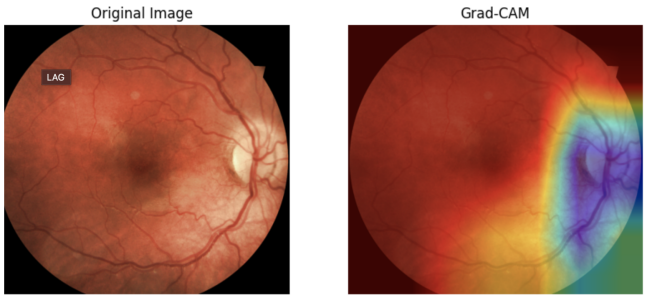


Figure 7: Grad-CAM Visualization of ResNet50

as it localized these small but critical indicators of glaucoma compared to ViT which architecturally focuses more on global patterns. Moreover, utilizing Grad-CAM visualizations for medical diagnosis have multiple benefits as it provide ophthalmologists visual justifications for the model's predictions, making it more effective for assisting medical diagnosis by human doctors.

Therefore, Figure 7 indicates that the CNN architecture of ResNet50 is better at extracting small features. Especially with ResNet allowing the layers to be much deeper than other CNNs by eliminating the vanishing gradient problem, it is clear that ResNet50 was better at identifying small indicators of glaucoma than ViT. Future work could involve validating Grad-CAM visualizations with expert annotations to confirm alignment with clinical markers of glaucoma.

5.4 Limitations and Future Directions

Despite utilizing a significantly larger dataset (total: 10004 after oversampling) with various augmentation methods to increase the size further, ViT's performance plateaued after Epoch 3, suggesting that the model may require even larger datasets or different augmentation strategies to fully converge and reach its potential. Future studies could explore training ViT on substantially bigger private datasets from multiple institutions or using synthetic data augmentations.

In addition, classifying glaucoma as binary oversimplifies the real-world problem of diagnosing glaucoma as it can be at different stages or conditions, requiring more elaboration than just binary prediction. Hence, developing multi-classification datasets for glaucoma detection could be crucial for identifying nuanced progression of the disease.

Future works should also investigate modifying the ViT architecture such as hybrid CNN-Transformer models that can capture both local and global images.

6 Conclusions

This study develops and evaluates the model performance and computational efficiency of ResNet50 and Vision Transformer for detecting glaucoma from fundus photographs. With a publicly available dataset, we fine-tuned both models by leveraging transfer learning with a small learning rate (0.0001) on pre-trained layers.

For diagnostic performances, ResNet50 achieved a 90.72% accuracy for the final epoch with F1 score of 0.8614. Although ViT also had strong performance with 87.64% accuracy and 0.8614 F1, ResNet50 exceeded ViT in performance metrics with 3.08% higher accuracy and 0.049 higher F1. Furthermore, in terms of computational efficiency, ResNet50 was 68.48% faster at making inferences and predicted 217.31% more images/second while using 50.52% less GPU, outperforming ViT also in the computational metrics. These findings highlight the suitability of ResNet50 over ViT for use in resource-constrained medical settings, where both reliability and efficiency are crucial, to assist ophthalmologists in diagnosing patients.

7 Acknowledgement

I would like to express my deepest thanks to Ms. Catherine Robinson for her guidance and support throughout the research. Her insights and feedback were greatly appreciated and helped me stay on track during key stages of my work.

References

[1] Francisco José Fumero Batista, Tinguaro Diaz-Aleman, Jose Sigut, Silvia Alayon, Rafael Arnay, and Denisse Angel-Pereira. 2020. RIM-ONE DL: A Unified Retinal Image Database for Assessing Glaucoma Using Deep Learning. *Image Analysis and Stereology* 39, 3 (2020), 161–167. https://doi.org/10.5566/ias.2346

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv preprint arXiv:2010.11929* (June 2021). https://doi.org/10.48550/arXiv.2010.11929 ICLR camera-ready version.

[3] Gabriel García, Rocío del Amor, Adrián Colomer, and Valery Naranjo. 2020. Glaucoma Detection From Raw Circumpapillary OCT Images Using Fully Convolutional Neural Networks. In *Proceedings of the 2020 IEEE International Conference on Image Processing (ICIP)* (Abu Dhabi, United Arab Emirates). IEEE, 2526–2530. https://doi.org/10.1109/ICIP40778.2020.9190916

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385* (December 2015). https://doi.org/10.48550/arXiv.1512.03385 Tech report.

[5] Elizabeth E. Hwang, Dake Chen, Ying Han, Lin Jia, and Jing Shan. 2023. Multi-Dataset Comparison of Vision Transformers and Convolutional Neural Networks for Detecting Glaucomatous Optic Neuropathy from Fundus Photographs. *Bioengineering (Basel)* 10, 11 (October 2023), 1266. https://doi.org/10.3390/bioengineering10111266

[6] Shajila Beegam M. K. and Mala Kalra. 2024. Leveraging CNN and Fundus Imaging for Enhanced Glaucoma Detection. *SN Computer Science* 5, 1 (2024), Article 1137. https://doi.org/10.1007/s42979-024-03527-4

[7] Mona Kaleem. n.d.. Glaucoma. [Online]. Available: https://www.hopkinsmedicine.org/health/conditions-and-diseases/glaucoma. Johns Hopkins Medicine. [Accessed: 2024-12-00].

[8] Ashish Kulkarni and H. Shafeeq Ahmed. 2024. A Comparative Study on Deep Convolutional Neural Networks and Histogram Equalization Techniques for Glaucoma Detection From Fundus Images. *medRxiv* (October 2024). https://doi.org/10.1101/2024.10.25.24316109

[9] Jordan Mandell, Benjamin Lin, and Ann Quan. 2024. *Evaluating the cup and disc in glaucoma.* https://eyeguru.org/blog/evaluating-cup-and-disc/ Accessed: 2024-12-01.

[10] Maxat Nurgazin and Nguyen Anh Tu. 2023. A Comparative Study of Vision Transformer Encoders and Few-shot Learning for Medical Image Classification. In *Proceedings of the 2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)* (Paris, France). IEEE, 2505–2513. https://doi.org/10.1109/ICCVW60793.2023.00265

[11] Ayush Roy. 2023. Glaucoma Classification Datasets: DRISHTI-GS, RIM-ONE, ACRIMA, ORIGA, and G1020. https://www.kaggle.com/datasets/ayush02102001/glaucoma-classification-datasets?resource=download Accessed: 2024-01-01.

[12] Srinivas Rahul Sapireddy. 2023. *ResNet-50: Introduction.* https://srsapireddy.medium.com/resnet-50-introduction-b5435fdba66f Accessed: 2024-12-00.

[13] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2019. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision* 128, 2 (2019), 336–359. https://doi.org/10.1007/s11263-019-01228-7

[14] Yiqiu Shen, Nan Wu, Jason Phang, and Jungkyu Park. 2020. An interpretable classifier for high-resolution breast cancer screening images utilizing weakly supervised localization. *arXiv preprint arXiv:2002.07613* (February 2020). https://doi.org/10.48550/arXiv.2002.07613

[15] Torchvision Contributors. 2023. *torchvision.models.resnet50.* https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html
[16] Torchvision Contributors. 2023. *torchvision.models.vit_b_16.* https://pytorch.org/vision/main/models/generated/torchvision.models.vit_b_16.html

Received 01 January 2025; accepted 27 March 2025

Co-occurrence of Extreme Heat and Pollution in the Southwest United States

Arjun Maganti
arjunmaganti2008@gmail.com
Basis Independent Silicon Valley
San Jose, California, USA

Zachary Espinosa
zespinosa97@gmail.com
University of Washington
Seattle, Washington, USA

ABSTRACT

In this research paper we examine the co-occurrence of extreme heat and humidity in the Southwest United States between 2003 and 2023. Using simple statistical regression analysis, time series analysis, and spatial compositing, we demonstrate that humidity, when coinciding with extreme heat, tends to exacerbate air pollution. Specifically, we find that humidity and heat have the largest impact of PM d <2.5μm and carbon monoxide (CO) with a lesser impact on other pollutants. We hypothesize the hygroscopic nature of PM2.5 particles and the reactive CO molecules with NOx and VOC compounds lead to amplified reaction rates during extreme heat and humidity. Next, we examine the meteorological conditions associated with the co-occurrence of extreme heat and humidity in Los Angeles and Sacramento. We demonstrate that heat waves in Los Angeles are associated with strong northwesterly winds, originating in the Mojave Desert, that are associated with anomalous warm atmospheric heat transport. This is also associated with decreased on-shore winds from the Pacific Ocean, which typically have a cooling effect. Decreased onshore winds resultantly lead to an exacerbation of heat conditions. These findings can contribute to further prevention of possible wildfire events caused by extreme heat conditions, and also lead to further protections against cases of extreme pollution in the southwestern United States.

KEYWORDS

co-occurrence of extreme heat and pollution, climate change, Southwest United States, compound events, air pollution

ACM Reference Format:

Arjun Maganti and Zachary Espinosa. 2025. Co-occurrence of Extreme Heat and Pollution in the Southwest United States. In *Proceedings of International Journal of Secondary Computing and Applications Research (IJSCAR VOL. 2, ISSUE 1).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.5281/zenodo.14988723

1 INTRODUCTION AND RELATED WORK

The southwestern United States is known for an abundance of wildfires ranging from man-made camping accidents to large scale forest fires. Despite decades of research, there is still much to improve in our understanding and preparation for wildfires. One particular area of research that requires further exploration is the co-occurrence of extremes events. For example, many fires, such as the Dixie and August Complex fires, co-occur during significant heat waves. Recent scientific study has shown that the frequency of co-occurring extreme heat, wildfires, and air pollution will increase due to anthropogenic global warming [1] [10] [11]. Here, we focus on examining the impact of co-occuring extreme temperature and humidity on air pollution.

There are several decades worth of literature examining the physical drivers and impacts of heat waves. Extreme heat can have profound negative impacts on human society, including affecting economic productivity, human mortality, and agricultural yield. These have shown that the frequency and intensity of extreme heat will increase due to global warming from the anthropogenic emission of greenhouse gases. Such studies have been conducted documenting all continents and parts of the globe, confirming that these effects have global consequences [4] [12]. These studies provide valuable insight into the long term effects that heat waves have on the environments of places rather than just the effects they have on humans.

Similar to extreme heat, air pollution has been shown to be exacerbated by anthropogenic emissions of greenhouse gases and other pollutants. A study by Schnell et al. 2017 [12] analyzed co-occurrences of extreme temperature, surface ozone, and particulate matter over the eastern United States. The study concluded that temperature was a key factor to the increased levels of ozone and PM2.5 in the atmosphere. Pollution levels have also increased in countries such as the United States, Portugal, and the United Kingdom [11] [8]. Pollution is a human health risk and can impact respiratory and cardiac-associated mortality. As a result, examining the co-occurrence of extreme heat and pollution can have implications in health research as well [2].

As such, our research looks to further this goal by analyzing the impact of humidity on the co-occurrence of extreme heat and air pollution in the southwest United States. By analyzing the effect of heat and humidity on the anomalous occurrence of pollutants, we use a composite map analysis to understand the difference in pollutant occurrence over the western United States during humid and dry heat waves. We also use a time series analysis to perform a case-study of several historical extreme heat events. The aim of this study is to improve our understanding of how pollution differs between dry heat waves and humid heat waves in the Southern western United States.

2 METHODOLOGY

2.1 Dataset

This study uses the CAMS (Copernicus Atmosphere Monitoring Service) global reanalysis dataset, which provides comprehensive

IJSCAR VOL. 2, ISSUE 1, April 2025,

https://doi.org/10.5281/zenodo.14988723

information about atmospheric composition, including temperature, humidity, and pollutants. CAMS is widely recognized in the field of atmospheric science for its high-resolution global data and assimilation of satellite observations. It was selected for its reliability, accessibility, and relevance to studying heatwave and pollution interactions.

We downloaded data from the CAMS global reanalysis dataset, covering January 1, 2003, through December 31, 2023. The spatial domain spans latitudes 32°N to 42°N and longitudes 114°W to 124°W, with each grid point spaced 0.7 degrees apart in both latitude and longitude. For each grid point and time step, the following variables were collected:

- Vector wind speeds (`u10` and `v10`).
- 2-meter temperature (`t2m`).
- Mean sea level pressure (`msl`).
- Concentrations of particulate matter with diameter <2.5 μm (`pm2p5`).
- Total column concentrations of carbon monoxide (`tcco`), methane (`tc_ch4`), nitrogen dioxide (`tcno2`), and ozone (`gtco3`).
- Total column water vapor (`tcwv`).

This produced a 3-dimensional dataset, structured with x and y coordinates for spatial positions, variable values for each of the chosen parameters, and a z-axis representing dates from 2003 to 2023. All variables were numerical, allowing for comprehensive statistical and graphical analyses. The NetCDF (nc) dataset was converted into a pandas DataFrame for efficient manipulation and visualization.

Table 1: List of CAMS variables and their abbreviations used in the study.

Name	Abbreviation
10-meter U wind component	u10
10-meter V wind component	v10
2 meter temperature	t2m
Mean sea level pressure	msl
Particulate matter d $\leq$ 2.5 μm	pm2p5
Total column carbon monoxide	tcco
Total column methane	tc_ch4
Total column Nitrogen Dioxide	tcno2
GEMS Total Column Ozone	gtco3
Total column vertically-integrated water vapor	tcwv

2.2 Preprocessing

Firstly, the preprocessing stage involved narrowing down the dataset to focus on relevant parameters, specifically anomalies in temperature and humidity during the months of May through September. The goal was to identify patterns in heatwaves and humidity, particularly during summer months leading into wildfire season. We use the 95th percentile of 2-meter temperature values during the months of May, June, July, August, and September to extract heat waves from our data set. The purpose of this specifically is to extract the warmest days possible across the time period, and to specifically extract summer days leading into wildfire season as these days have the most relevance to overall pollution values. Similarly, we use the 90th percentile of water vapor during these same months to find the set of most humid days during this timespan. This process creates two datasets consisting of the 95th percentile of 2-meter temperature values for a location, and the 90th percentile of total column water vapor values for that same location. Intersecting the two sets gives us one combined dataset of the most hot and humid days for a specific x and y grid point.

2.3 Methods

Most formal research done in this topic uses a combination of mathematical models or neural network models to effectively predict such cooccurrences. These studies are out of the scope for this paper. Due to this, our paper will use a combination of two methods: time series analysis and composite map analysis.

Time series analysis is used to investigate the temporal relationship between heat waves, humid events, and pollutant levels. For this, anomalies in temperature and humidity were extracted for specific heat wave and humid wave events within a 10-day frame. Temperature or humidity served as causation variables, which were plotted on individual axes. Concurrently, pollutant levels, such as particulate matter (PM2.5) or total column ozone, were plotted on a dual y-axis within the same graph. This dual y-axis approach creates overlapping line charts, where one line represents temperature or humidity, and the other line represents the pollutant of interest. This visualization effectively highlights the temporal correlation and potential lead-lag relationships between these variables. Such analysis provides a detailed understanding of how individual heatwave or humid wave events impact pollutant levels in the atmosphere across the selected geographic region.

Composite map analysis was performed using the Cartopy library in Python to explore the spatial dynamics of atmospheric variables during heatwave and humid wave events. By mapping wind vectors, pressure gradients, and gradients of other atmospheric variables, we generated averaged maps that provide a comprehensive snapshot of the meteorological and atmospheric conditions during these events. Additionally, difference maps were created to isolate the effects of humidity during heatwaves. By calculating the differences in atmospheric conditions between hot days and humid-hot days, we were able to further investigate the influence of humidity on pollutant levels, such as ozone and particulate matter, during extreme weather events. These spatial analyses allow us to examine not only the localized effects of heatwaves and humidity but also broader patterns that may emerge across the study region.

3 RESULTS

3.1 Composite Map Analysis

The composite map analyses are presented in Figures 1–12. Figures 1–6 focus on Los Angeles, while Figures 7–12 highlight Sacramento. Each figure consists of three panels:

(1) The left panel represents the variable's average anomaly during hot days (95th percentile of `t2m`).
(2) The middle panel shows the anomaly during hot and humid days (95th percentile of `t2m` and 90th percentile of `tcwv`).
(3) The right panel illustrates the difference between the hot and humid day anomalies and the hot day anomalies.

Figures 1–6 display the behavior of six atmospheric variables in Los Angeles during anomalous events: 2-meter temperature (`t2m`), particulate matter (`PM2.5`), carbon monoxide (`tcco`), methane (`tc_ch4`), nitrogen dioxide (`tcno2`), and ozone (`gtco3`).

Los Angeles, California (34.05 N, 118.25 W)

Figure 1: 2-meter temperature

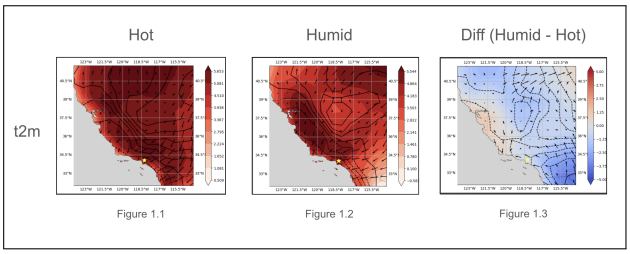


Figure 2: Particulate Matter d < 2.5µm

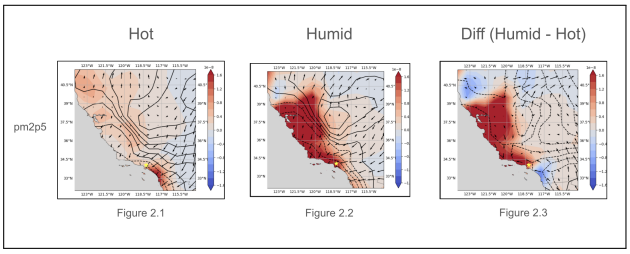


Figure 3: Total Column Carbon Monoxide

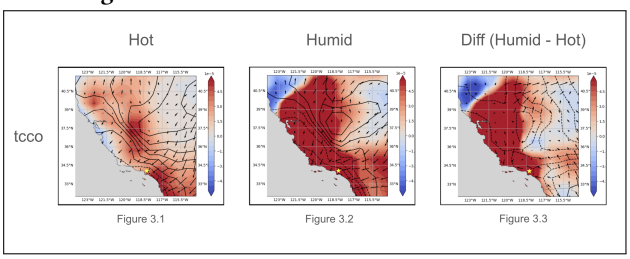


Figure 4: Total Column Methane

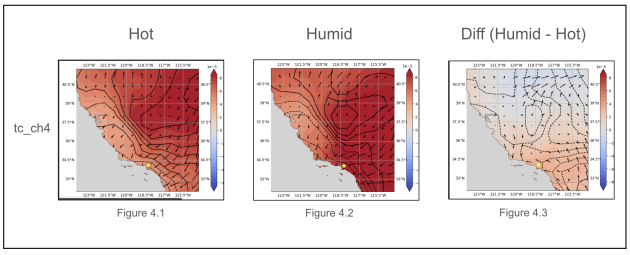


Figure 5: Total Column Nitrogen Dioxide

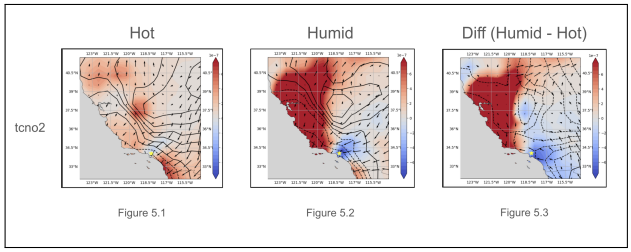
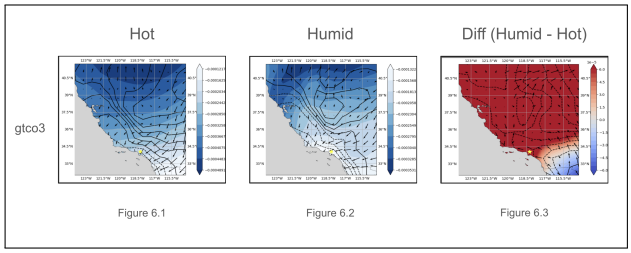


Figure 6: Total Column Ozone



In Los Angeles, composite map analyses reveal notable patterns for the six atmospheric variables during anomalous events. The 2-meter temperature (t2m, Figure 1) shows minimal differences between hot and hot-humid days, as the anomalies are inherently tied to the hot-day criteria, with slightly lower values on humid days due to stricter filtering. Particulate matter (PM2.5, Figure 2) exhibits significant positive anomalies on humid days, with widespread increases across the western portion of the state and the Central Valley, compared to the more localized anomalies near Orange County and San Diego on hot days. Carbon monoxide (Figure 3) displays substantially higher anomalies on humid days, with positive differences observed across the region except for inland desert areas, which remain largely unaffected. Methane (Figure 4) anomalies are generally small but consistently positive, with hot days contributing to markedly elevated levels compared to average conditions. Nitrogen dioxide (Figure 5) presents a more complex trend: hot days show negative anomalies near Los Angeles, whereas humid days result in significant positive anomalies extending north through the San Francisco Bay Area and Central Valley, as captured in the difference map. Lastly, ozone (Figure 6) is the only variable negatively correlated with heat, with hot days resulting in reduced ozone levels. However, humid days mitigate this reduction, leading to less pronounced negative anomalies and slightly higher ozone levels than on hot days alone.

Sacramento, California (38.58 N, 121.50 W)

Figure 7: 2-meter temperature

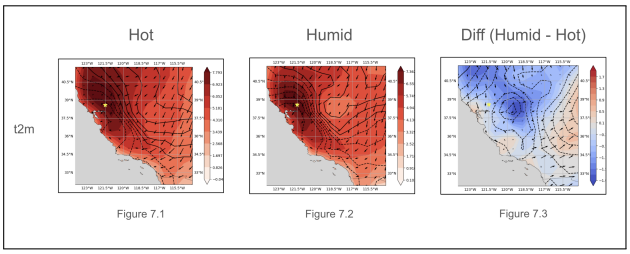


Figure 7.1 Figure 7.2 Figure 7.3

Figure 8: Particulate Matter d < 2.5µm

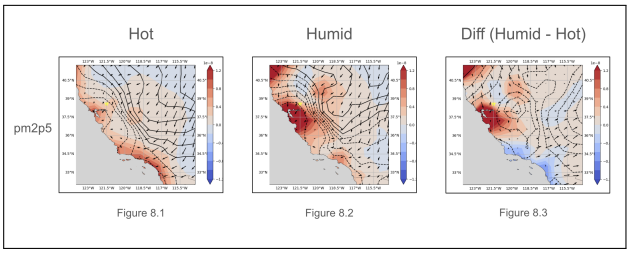


Figure 8.1 Figure 8.2 Figure 8.3

Figure 9: Total Column Carbon Monoxide

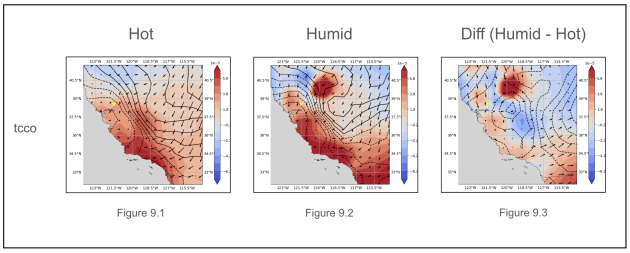


Figure 9.1 Figure 9.2 Figure 9.3

Figure 10: Total Column Methane

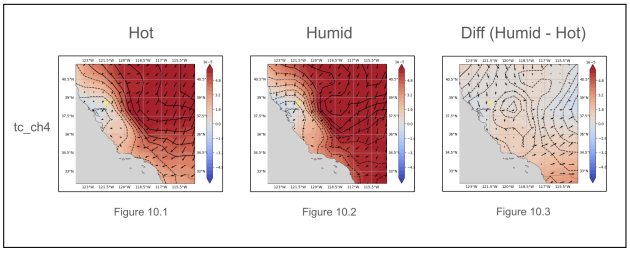


Figure 10.1 Figure 10.2 Figure 10.3

Figure 11: Total Column Nitrogen Dioxide

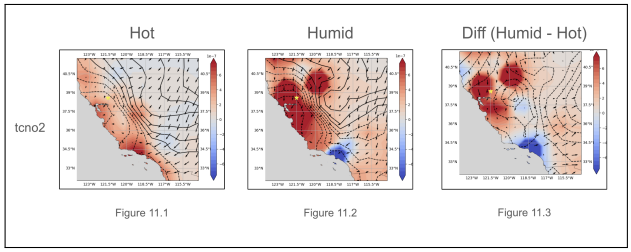


Figure 11.1 Figure 11.2 Figure 11.3

Figure 12: Total Column Ozone

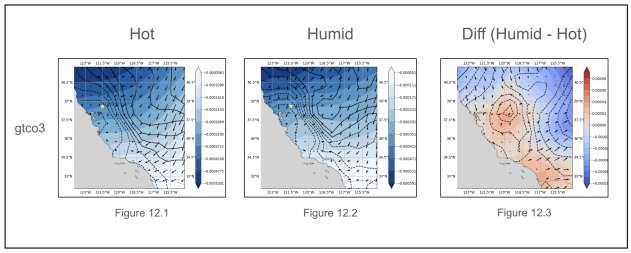


Figure 12.1 Figure 12.2 Figure 12.3

In Sacramento, the trends in composite map analyses largely align with those observed in Los Angeles, with some regional distinctions due to its inland location. Temperature anomalies (t2m, Figure 7) show negligible differences between hot and hot-humid days. Particulate matter (PM2.5, Figure 8) demonstrates a similar trend to Los Angeles, with significantly higher anomalies on humid days, particularly across the Central Valley and surrounding regions. Carbon monoxide (Figure 9) anomalies in Sacramento are smaller on humid days compared to Los Angeles at the center of axis, reflecting its inland position, but still show positive differences, indicating the influence of humidity. Methane (Figure 10) follows the same pattern as in Los Angeles, with modest positive differences and elevated anomalies on hot days. Nitrogen dioxide (Figure 11) anomalies differ slightly, as coastal areas show negative differences, while inland areas, including Sacramento and regions to the north and east, exhibit positive differences on humid days, compared to Los Angeles which still exhibits negative anomalies. Ozone (Figure 12) trends are consistent with Los Angeles, with lower anomalies on hot days and less pronounced negative anomalies on humid days, suggesting a similar mitigating effect of humidity on ozone reductions caused by heat.

3.2 Time Series Analysis

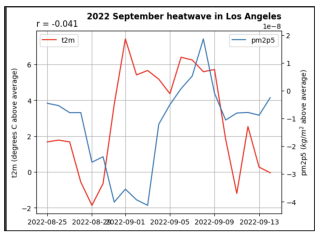


Figure 13.1

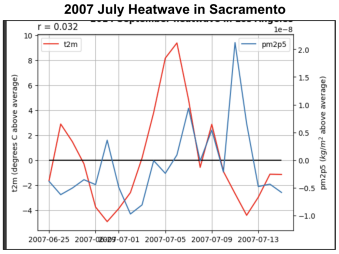


Figure 13.2

To further investigate the lead-lag relationship between temperature and pollutants, specific examples provide valuable insights. Figures 13.1 and 13.2 illustrate two notable heatwaves: the September 2022 heatwave in Los Angeles and the July 2007 heatwave in Sacramento. Both events exhibit significant temperature anomalies, with peak temperature variations ranging from 7 to 9°C (approximately 14 to 16°F) above typical values. A clear correlation is observed with particulate matter (PM2.5), as PM2.5 concentrations surge approximately 2 to 3 days following the peak of the heatwaves. These instances are representative of common heatwave patterns, and by aggregating multiple cases, we identify a general trend linking temperature anomalies to pollutant fluctuations. Although natural variation in PM2.5 and other pollutants exists, these examples demonstrate that substantial heat anomalies play a significant role in shaping pollutant levels, thus providing a clearer understanding of their dynamic relationship.

4 DISCUSSION

Humidity's variability has significant impacts on multiple pollutants as discussed in the Los Angeles and Sacramento examples. The factors influencing specific variables' variations result from unique, but linked chemical processes that occur in the atmosphere. As such, each variable must be studied individually to examine the nature of our results.

4.1 PM2.5

In the instance of PM2.5, the size is the key factor why it is significantly higher on humid days. Hygroscopy is a process where a particle readily bonds with other particles around it without assistance of added pressure, heat, or circulation. PM2.5 particles are naturally very hygroscopic, and bond with the increased water vapor in the atmosphere, thus leading to larger PM2.5 molecules. These are detected as higher values by montoring equipment [2]. Similarly, aerosols, small bundles of solids and liquids suspended in the air, tend to increase during humid events. Thus, previously non PM2.5 matter can be converted into matter that is detected and classified as PM2.5, thus increasing its detections.

4.2 Carbon Monoxide

Carbon monoxide also showed a significant increase on humid days. The main attributing factor is the way this variable was measured. The full name of the variable is total column carbon monoxide, which refers to the total amount of carbon monoxide present in a specific vertical shaft of the atmosphere from the ground to the top of the atmosphere. As a result, mixing within a column is not a viable explanation, although humidity does often lead to thermal inversions that would allow for restrained column mixing. Instead, a viable explanation could be the low rates of oxidation. CO typically reacts with hydroxyl radicals (OH) in the atmosphere which allows its concentration to disperse over time. Water vapor, however, is a reacting agent that can exacerbate this process, resulting in larger CO values. Similarly, CO is a common byproduct of many other pollutants' reactions, like Volatile Organic Compounds (VOCs) and Nitric Oxide compounds (NOx) [14] [6].

4.3 Methane

Methane has much smaller, but still positive anomalies on humid days than hot days. Similar to CO, the measurement of methane is throughout the total atmospheric column, eliminating mixing and the elevation of a measurement from a factor. Consequently, the lack of dispersion of methane is attributed to a variety of factors. Heat waves are generally associated with high pressure systems, and moist air can trap methane in stagnant air masses, causing its measurements to increase on humid days. As well as this, similar NOx and VOC reactions can cause methane not to react with hydroxyl groups as much as they would on a hot, dry day due to the introduction of water vapor [5]. In a similar case to carbon monoxide, methane would increase, but not at the same rate due to its reactions tending to be slightly less frequent. A more biological explanation is also plausible. Humidity can sometimes lead to increased moss, fungi, and plants grown in soil, contributing to the flow of groundwater and reservoirs. This water can lead to methane-producing bacteria that also contribute, although on a smaller scale, to methane in the atmosphere.

4.4 Nitrogen Dioxie

Nitrogen dioxide (NO2) shows a negative correlation with humidity in the center of axis for Los Angeles, but not for Sacramento. The negative correlation is due to the transformation of NO2 into a different compound, nitric acid (HNO3). In the atmosphere, NO2 reacts with hydroxyl (OH) molecules to form nitric acid, and elevated humidity levels act as a key catalyst in this reaction. As a result, this essentially drains most NO2 from the atmosphere. Nitrogen dioxide is also part of the NOx family of gases, and this reduction in gases can also lead to the exacerbating effects of carbon monoxide and methane in the atmosphere. In contrast, the positive anomalies observed in Sacramento can be attributed to the influence of local atmospheric conditions, such as lower humidity levels and the proximity of the regio to agricultural areas, which can contribute to increased NO2 emissions.

4.5 Ozone

Although ozone has a large difference between humid heat and dry heat, all composited anomalies are negative. Ozone negatively correlates with heat. Ozone is typically associated with nitrogen dioxide, which means that NO2 and O3 reactions are used to regenerate NO and oxygen gas, leading to the reduction of both gases. This is known as the NO2-O3 equilibrium shift [6]. Despite this, the increase in ozone is evident. A plausible explanation for this

circumstance is that increased water vapor plays a role in this reaction. Water vapor can cause the formation of peroxy radicals (RO2) along with typical oxygen gas. As a result, atomic oxygen can combine with the oxides in these radicals to reform O3, thereby regenerating it at the expense of nitrogen dioxide, which is not regenerated at the same rate.

5 CONCLUSION AND LIMITATIONS

5.1 Summary

In this paper we examined the co-occurrence of extreme heat and humidity in the Southwest United States between 2003-01-01 and 2023-12-31. We use a combination of simple statistical regression analysis, time series analysis, and spatial compositing to demonstrate that humidity, when coinciding with extreme heat, tends to exacerbate air pollution. Specifically, we find that humidity and heat have the largest impact of PM d <2.5µm and carbon monoxide (CO), with a lesser, but still significant impact on ozone (O3), nitrogen dioxide (NO2), and methane (CH4). We hypothesize that humidity has a larger impact of CO and PM2.5 because the hygroscopic nature of PM2.5 particles and the reactive CO molecules with NOx and VOC compounds lead to exacerbated results especially in extreme heat and humidity. Additionally, we examine the meteorological conditions associated with the co-occurrence of extreme heat and humidity in Los Angeles and Sacramento. We demonstrate that heat waves in Los Angeles are associated with strong northwesterly winds, originating in the Mojave Desert, and anomalous warm atmospheric heat transport due to large pressure gradients between said areas. This is also associated with decreased on-shore winds from the Pacific Ocean, which typically have a cooling effect, but are non-present in these situations due to the reversal of the winds. Similar studies have been conducted for other regions of the globe [3], however, to the best of our knowledge, this is the first study systematically examining the co-occurrence of extreme heat, humidity, and pollution in the Southwestern United States.

5.2 Limitations

There are several limitations to this study and avenues for additional analysis. First, we do not examine the robustness of our heatwave definition. We define heatwaves using a 1-day exceedance of the historical 95th percentile of 2-meter temperature. While this quantile-based measurement of heatwaves is standard in past studies, most studies require that this occurs for 3 or more consecutive days. The definition used here simplifies the complexity of heatwaves and overlooks multi-day thresholds that are more robust. If given additional time, we would explore the robustness of our results using various heatwave definitions from other studies. Second, we would like to more thoroughly examine the meteorological conditions associated with the co-occurrence of extreme heat, humidity, and pollution. For example, we may investigate the impact of regions with climatological inversions (e.g., the Central Valley of California and Los Angeles) and compare rural versus urban centers. Finally, we do not control for the impact of wildfires, which may influence the results of this study. Wildfires can both trigger heatwaves and be exacerbated by them, and while humidity can play a role, it is unlikely to be the only contributing factor. The significance of humidity relative to other factors cannot be fully determined in this study. Further research should isolate the independent effects of various factors, including wildfires, to increase the robustness and accuracy of studies on this topic.

REFERENCES

[1] A. AghaKouchak, F. Chiang, L. S. Huning, C. A. Love, I. Mallakpour, O. Mazdiyasni, and M. Sadegh. 2020. Climate extremes and compound hazards in a warming world. *Annual Review of Earth and Planetary Sciences* 48, 1 (2020), 519–548.

[2] Célia Alves et al. 2023. PM2.5 chemical composition and health risks by inhalation near a chemical complex. *Journal of Environmental Sciences* 124 (2023), 860–874.

[3] Alan I. Barreca. 2012. Climate change, humidity, and mortality in the United States. *Journal of Environmental Economics and Management* 63, 1 (2012), 19–35.

[4] J. Clark, J. T. Abatzoglou, N. J. Nauslar, and A. M. Smith. 2020. Verification of red flag warnings across the northwestern US as forecasts of large fire occurrence. *Fire* 3, 4 (2020), 60.

[5] Christopher D. Holmes et al. 2013. Future methane, hydroxyl, and their uncertainties: key climate and emission parameters for future predictions. *Atmospheric Chemistry and Physics* 13, 1 (2013), 285–302.

[6] Justyna Jaroszyńska-Wolińska. 2010. The reaction mechanism of ozone with the NO and NO2 oxides. *Journal of Molecular Structure: THEOCHEM* 952, 1-3 (2010), 74–83.

[7] S. Jodzis and K. Baran. 2022. The influence of gas temperature on ozone generation and decomposition in ozone generator. How is ozone decomposed? *Vacuum* 195 (2022), 110647.

[8] Egide Kalisa et al. 2018. Temperature and air pollution relationship during heatwaves in Birmingham, UK. *Sustainable Cities and Society* 43 (2018), 111–120.

[9] J. Li, X. An, Q. Li, C. Wang, H. Yu, X. Zhou, and Y. A. Geng. 2022. Application of XGBoost algorithm in the optimization of pollutant concentration. *Atmospheric Research* 276 (2022), 106238.

[10] E. Mario, L. Raffaele, C. Onofrio, C. S. J. Maria, B. Valentina, G. Vincenzo, and S. Giovanni. 2024. Coupling heat wave and wildfire occurrence across multiple ecoregions within a Eurasia longitudinal gradient. *Science of the Total Environment* 912 (2024), 169269.

[11] J. Parente, M. G. Pereira, M. Amraoui, and E. M. Fischer. 2018. Heat waves in Portugal: Current regime, changes in future climate and impacts on extreme wildfires. *Science of the Total Environment* 631 (2018), 534–549.

[12] J. L. Schnell and M. J. Prather. 2017. Co-occurrence of extremes in surface ozone, particulate matter, and temperature over eastern North America. *Proceedings of the National Academy of Sciences* 114, 11 (2017), 2854–2859.

[13] Yi Sun et al. 2020. Examining the joint effects of heatwaves, air pollution, and green space on the risk of preterm birth in California. *Environmental Research Letters* 15, 10 (2020), 104099.

[14] Wenjing Wang et al. 2023. Volatile organic compound emissions from typical industries: Implications for the importance of oxygenated volatile organic compounds. *Atmospheric Pollution Research* 14, 1 (2023), 101640.

[15] Xiaorui Zhang et al. 2022. Observed sensitivities of PM2.5 and O3 extremes to meteorological conditions in China and implications for the future. *Environment International* 168 (2022), 107428.

Received 02 January 2025; accepted 22 February 2025

A Study of Drift Variability in Crazyflie Nano-Quadcopters by Pitting in-built Logging Parameters against Meticulous Measurement Techniques

Vihaan Bhaduri
Saratoga High School
Saratoga, California, USA
vihaan.bhaduri@gmail.com

Mark Santolucito
Barnard College, Columbia University
New York, USA
msantolu@barnard.edu

Abstract

Drift is one of the most unpredictable challenges when operating nano-quadcopters like Crazyflie. Caused by various factors including propeller orientation, battery positioning, and hardware manufacturing variations-drift significantly impacts flight predictability, particularly in tight spaces such as narrow hallways, crowded rooms, and public areas. While a permanent solution to drift may not exist—especially when it arises from inherent hardware variations such as mechanical inconsistencies and alignment inaccuracies—understanding its effects is crucial for flight precision, safety, and the mitigation of unintended flight path deviations. Moreover, drift analysis is essential in environments lacking precise positioning systems. Addressing drift directly within flight operations provides a more effective and broadly applicable solution to this mechanically induced issue. In this case study, we examined the typical drift patterns observed in Crazyflie quadcopters, assessing their scale and comparing differences across model versions. This analysis can help researchers determine whether these drones offer the precision required for their applications or if additional stabilization measures, such as a lower-level Proportional-Integral-Derivative (PID) controller, are necessary.

Keywords

Nano-quadcopters, Crazyflie, mechanical variability, drone drift

ACM Reference Format:
Vihaan Bhaduri and Mark Santolucito. 2025. A Study of Drift Variability in Crazyflie Nano-Quadcopters by Pitting in-built Logging Parameters against Meticulous Measurement Techniques. In *Proceedings of International Journal of Secondary Computing and Applications Research (IJSCAR VOL. 2, ISSUE 1).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.5281/zenodo.14988716

1 Introduction

Drift is a critical factor in all aspects of flight, as its impact on trajectory can compromise safety and precision. In our study, we focused on mitigating hardware- and mechanically-induced drift in the Crazyflie quadcopter series from Bitcraze—an issue possibly stemming from slight imperfections in the drone's construction. Though seemingly minor, such variations can lead to unintended deviations, increasing the risk of collisions with surrounding obstacles. As we advance toward autonomous drone operation, refining even the smallest flight details is essential to achieving an optimally designed drone. An ideal build could potentially eliminate drift, but it is very difficult to achieve. Since drift is undeniable, minimizing it is crucial for maintaining a drone's structural integrity and ensuring it can reliably complete tasks without sustaining damage. This would enable precise navigation through tight spaces, where accuracy is vital for the drone's operation. In summary, the key contributions of our work are:

(1) Graphing data from logger module vs. manually measured drift data to determine whether the logger module provided by Bitcraze is capable of generating accurate location data. The logger data proved to be quite erratic and often incorrect, and drift was undeniably visible in all models of the Crazyflie.
(2) Cross-evaluating model performance with Crazyflie 2.1+. This model performed better than Crazyflie 2.1 model in terms of drift but certainly showed its significant presence affecting all aspects of flight.
(3) Validating whether having a patterned floor would affect the quadcopter's optical position analysis module perform better, thereby reducing drift. The experiment did not yield a positive outcome.

2 Related Work

Although drift is essential in all forms of aerial flight, there is minimal research available on drift analysis and its implications on a quadcopters. A significant and manually created drift is explored in [5]. By cutting the propellers and analyzing the effects this had on the drone, the authors were able to better understand the significance of drift on a quadcopter. Another work, [6], related to drone drift in fact argues that the concept is often negligible. Built-in and pre-implemented flight algorithms were utilized to reduce the effect of drift in this paper. The paper mentions the use of IMUs (Inertial Measurement Units) when conducting short flight experiments where drift is insignificant. However, in situations where mechanical drift is high, the simple use of an IMU is problematic.

Nano-Unmanned Aerial Vehicles (UAVs) have the potential to be one of the most useful forms of autonomous drones. Whether they are used for traffic patrolling, hallway monitoring, or other purposes, they can play a significant role in shaping the future. However, their creation is not simple. Many challenges, such as memory problems and their short lifetimes, pose difficulties. In [7], the use of the Parallel Ultra-Low Power Shield (PULP-Shield) as well as the convolutional neural network, PULP-DroNet, is able to both

IJSCAR VOL. 2, ISSUE 1

calculate the steering angle of the drone as well as the drone's crash probability, resulting in a functional automated nano-quadcopter.

This drone was created by adding a PULP-Shield on top of the Crazyflie 2.0's body. The PULP-Shield consists of a PULP-GAP8 processor, composed of a fabric controller and a cluster, as well as other parts. A fabric controller is similar to a microprocessor, and the cluster is used to accelerate parallel code running on the processor. The PULP-Shield's use is mainly related to input filtration. The image stream coming from the camera is offloaded to the Crazyflie's hardware, where the drone's microcontroller unit (MCU) then offloads much of the intensive visual navigation workload to the PULP accelerator. This filtered input is then sent to the PULP-DroNet model.

Although this automated nano-quadcopter is functional, it still lacks the necessary robustness needed for real-life application. As seen in the video provided in the paper, the drone is unable to keep a straight and steady path, moving at a slow pace of 1.5 m/s while maintaining a height of 0.5 m/s. Furthermore, the PULP-DroNet model is trained on an outdoor dataset, making it difficult to perform with environmental variations. In our work, we addressed this challenge by performing our experiments in an enclosed garage space.

When regarding all forms of aerial flight, flight precision cannot be trivialized. A high precision model could save battery life, increase efficiency, and reduce crash hazard. When regarding work similar to that conducted by ETH Zurich [7], drift analysis can be essential to further progress in this space. Positioning alternatives, although feasible as possible solutions, are both less effective and unreasonable in the context of experimental flight. Drone drift analysis when utilized effectively reduces flight risks. Moreover, in the new domain of swarm flight [2, 4, 8], drone drift could potentially create accidents between drones. If a quadcopter veers off its flight path, it could easily collide with other neighboring drones, leading to crashes.

A potential solution to drift mitigation is seen within the context of runtime monitoring, a technique often used to detect threats to a system. As demonstrated in [1] this research domain has huge potential in drone autonomy and flight enhancement. The authors used RTLola, a stream-based specification language that helped develop a dynamic monitoring framework for the DLR ARTIS (Autonomous Robotics and Technology for Intelligent Systems), a UAV. RTLola is both an expressive and reliable specification language, meaning it is able to analyze the health status of a UAV during flight. The use of the specification language was heavily present in the work demonstrated, helping UAVs on reconnaissance missions. Based on real-time data analysis, a UAV would make decisions centered on ensuring its safety. The aforementioned details of RTLola can possibly be utilized for direct drift correction. The safety validation system can be changed to prevent excessive displacement from the flight path. These technologies are essential to alert the system of any errors so it can correct itself.

3 Hardware Stack

The Crazyflie 2.1/2.1+ is built from a variety of different microprocessors and chips in order to create a lightweight and prototype-friendly nano-quadcopter. The essential components in use for our work were part of Bitcraze's Flow Deck V2. The navigation sensor is composed of two main components: the VL53L1x ToF sensor and the PMW3901 optical flow sensor. The former is used for precise measurement in the drone's upwards movement (traditional, '+z' axis) while the latter measures movement relative to the ground. The sensor calculates the change in pixels between frames. We collected data from flight trials on both the Crazyflie 2.1 and Crazyflie 2.1+ models in order to analyze drift. Some of the data we were able to collect also allowed us to help determine the performance capability of the optical flow sensor under specific conditions, analyzing whether a patterned or netural floor would improve the sensor's estimations.

Figure 1: Enclosed garage setup to minimize environmental factors

4 Methodology

To collect sufficient data to prove drift variability, we conducted a series of five different styles of flight experiments (in-place, forward, backward, rightward, and leftward) as shown in Fig. 2 using two different drone models: Crazyflie 2.1+ and Crazyflie 2.1. We used Crazyflie 2.1+, one of Bitcraze's latest releases, as our base model. The lightweight drone was assembled with the Flow Deck v2 attached to its underside. This deck helped the drone understand its location and generate coordinates. We used these coordinates to calculate the net drift of the drone during flight and used Crazyflie's motion commander ('mc') module for command and control.

Drift measurements were performed in an enclosed garage to minimize the effect of environmental factors such as wind and erratic air currents and external temperature variations, as shown in Fig. 1. In all our experiments, we measured between two points, a take-off and a landing point. Ideally, with zero drift, these two points should coincide for in-place (hover) flights and should follow perfect grid geometry for right and left lateral flights and forward and backward flights. We kept a distance of two meters between the takeoff and the ideal landing points in each of the four flight types, as depicted in Fig. 2.

To calculate the net drift caused by the flight, we measured the displacements 'x' and 'y' from the takeoff location to the landing position. While we performed manual measurements, the Bitcraze

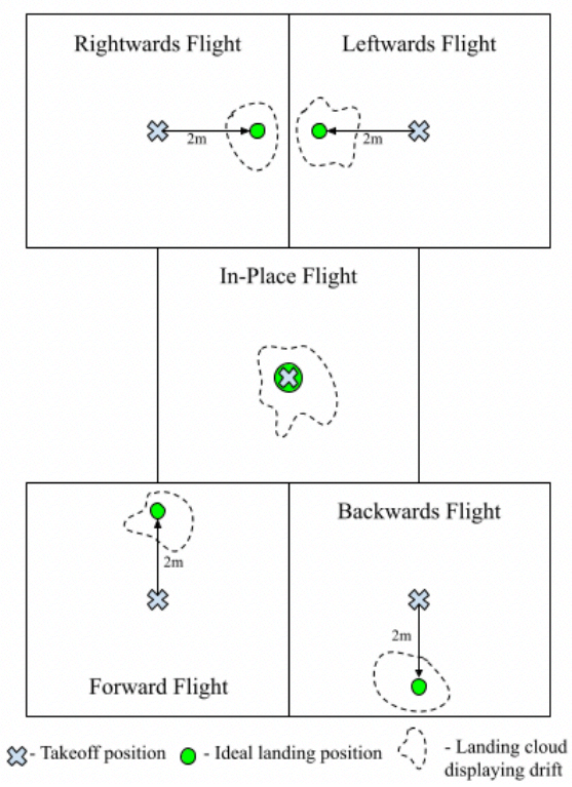


Figure 2: Five different drone flights - ideal vs actual

logger module would also record its own interpretations of drift. We used three essential log variables:

`state.position.x, state.position.y, state.position.z`

from the Bitcraze position framework to record the drone's location coordinates. This was later used to calculate drift from the logger module. We faced a challenge matching the coordinate axes between Bitcraze's logger system axes convention and the traditional coordinate axes used in our manual measurements. Our traditional coordinate system assumed the drone's front/nose aligned with the positive 'y' direction. The Bitcraze logger system differed from this convention by using unconventional axes as: right (traditional '+x' axis) → '-y' axis, forward (traditional '+y' axis) → '+x' axis, left (traditional '-x' axis) → '+y' axis, backward (traditional '-y' axis) → '-x' axis. We later adjusted the data to conform to our interpretation of a logical and traditional axes system.

We divided the flight into two categories, in-place flight and path-based flight. The in-place flight involved a simple movement:

`mc.up(1.5, velocity=1.5)`

This would elevate the drone 1.5 meters upward (traditional '+z' axis), hover for 10 seconds, and then descend. In the path-based flight, we measured drift variability while the Crazyflie was in motion across four different simple paths - lateral left, lateral right, forward, and backward. We positioned the drone to be oriented such that its direction of flight would lead it to its ideal landing spot (ex: for forward flight, we placed the drone with its nose facing forward). We followed the same flight motion as the in-place flight, raising the drone to a height of 1.5 meters, where it would hover for 3 seconds. Then we would use Bitcraze's motion commander module to fly the drone two meters in the path direction (left, right, forward, backward). The quadcopter would then finally descend.

In total, we conducted 50 drift variability tests of in-place flight using the Crazyflie 2.1+ model and 25 tests for each category of path-based flights: forward, backward, rightward, and leftward. We also performed 40 in-place flight drift variability tests on the Crazyflie 2.1 model that helped us compare the variability between models and conclude that the drift variability is undeniable and real. There was no set guidance behind choosing the 50 and 40 iteration numbers. To balance human time and effort spent conducting such meticulous experiments and measurements, we used our engineering discretion to finalize the test iteration numbers that we considered reasonable to help us empirically conclude drift variability.

Moreover, we performed an additional 25 in-place flight runs to test the impact of propeller replacement on drift. We swapped the pair of propellers, which came with the Crazyflie 2.1 bundle, with replacement propeller parts.

Lastly, we conducted tests to prove that a checkered pattern on the floor would not affect the perception of movement by the PMW3901 optical flow sensor, an integral part of the Crazyflie flow deck module. We created a nonuniform grid pattern on the floor and performed a series of 10 in-place flight tests.

5 Results

5.1 Graphical Interpretation

The results of our collected data were represented in three different ways (see Fig. 6 - Fig 13). The (a) graphs for each mini-dataset composed of a scatter plot of both the logger and manually interpreted data. Points relating to the same data are connected in the graph. The (b) and (c) graphs represent a running average plot with the blue graph displaying the x-drift while the orange graph indicating the y-drift. The (b) graphs represented the trend lines for the data collected by the logger, while the (c) graphs represented manually collected data. The inability for the trend lines to plateau emphasizes the presence of inherent variability in drift.

All outliers in our dataset were ignored while plotting the graphs; otherwise, they would make them visually unreadable. These outliers were a result of the erratic output of the crazyflie's logger module.

In-place flight is the most accurate measurement of drift, as shown in Fig. 6. It does not involve any potential confounding variables that Crazyflie's motion commander module might have. Hence, we used in-place flight as our baseline for drift comprehension and analysis.

Crazyflie 2.1+: Depicted in Fig. 6, Fig. 7, Fig. 8, Fig. 9, and Fig. 10.
Crazyflie 2.1: Depicted in Fig. 11.
Crazyflie 2.1 with new propellers: Depicted in Fig. 12.
Crazyflie 2.1+ with a patterned floor: Depicted in Fig. 13, Fig. 5.

5.2 Threats to Validity

Our experiments, despite their rigorous nature and proof, have the potential to be limiting due to small mechanical discrepancies between quadcopter builds. However, the general conclusion on

the omnipresence and potentially detrimental drift is undeniable across drones.

Internal Validity: The quadcopters that we used in our experiments could potentially have had mechanical defects. This could include motherboard variations, warped propellers, or misaligned battery installation. Despite our thorough balancing of the propellors, and precise installation of the quadcopter components, a chance of mechanical misalignment remained unavoidable that could have affected the drift slightly.

External Validity: There is no guarantee; in fact, it is nearly impossible to duplicate this study. However, a replication of this work is certainly possible. While it may not yield the exact results presented in this paper, it will likely show a similar trend and conclusion, unless it is inherently addressed in future Crazyflie models. Every quadcopter is made uniquely, and thus drift is more noticeable in some than in others. This generalization may apply to all miniature drone models, not just the Crazyflie series by Bitcraze. However, drift can potentially be mitigated, regardless of the type of quadcopter. We proposed a few options in Section 6.1 (Future Work).

5.3 Analysis

Our different flight trials gave us a varying set of data points. However, all plots indicated one key element: drift is evident in all our unique flight types - dominant in Crazyflie 2.1 and a little less dominant in Crazyflie 2.1+.

The Crazyflie logger module and the motion commander module also presented inaccurate results. The data from the logger module often deviated drastically from the ground truth values in both the 'x' and 'y' directions. The use of the logger module when attempting to control flight is problematic, as its inaccurate positional estimates would make the flight adjustment process difficult. Furthermore, the motion commander module often was unable to move the drone as expected. For instance, with the Crazyflie 2.1+, in-place (hover) flight deviated by around 10-15 centimeters in both the 'x' and 'y' directions; rightward and leftward lateral flights deviated by around 50 centimeters in the 'y' direction. Even forward and backward flights demonstrated significant drifts with an average variation of (-7.51 cm, 17.14 cm) for forward flight and (-0.34 cm, -38.40 cm) for backward flight from the ideal. Our trials for in-place flight in Crazyflie 2.1 gave us an average deviation of (42.46 cm, -53.62 cm) when taking off from (0, 0).

In addition, we witnessed an inconsistency in drift when we replaced the propellers of the Crazyflie drone as shown in Fig. 3. When the counterclockwise rotating propellers were replaced with a new pair, the average coordinates for in-place flight was (-17.30 cm, 46.93 cm) as shown in Fig. 12 graph. Thus, any form of mechanical deviation like a propeller replacement, would alter the flight trend, making the drone's flight unpredictable. This can pose a major handicap for precise navigation and flight safety. Consistency is hard to come by in the Crazyflie models. Even without a change in flight type or mechanical components, the drone would still not produce repeatable results. This can be seen by the often fluctuating running mean plots shown in Fig. 4(b).

The variability within each of the datasets is concerning and can be seen in the (b) and (c) graphs throughout Fig. 6 - Fig 13 in details. To aid visualization, a summary graph showing this variability is

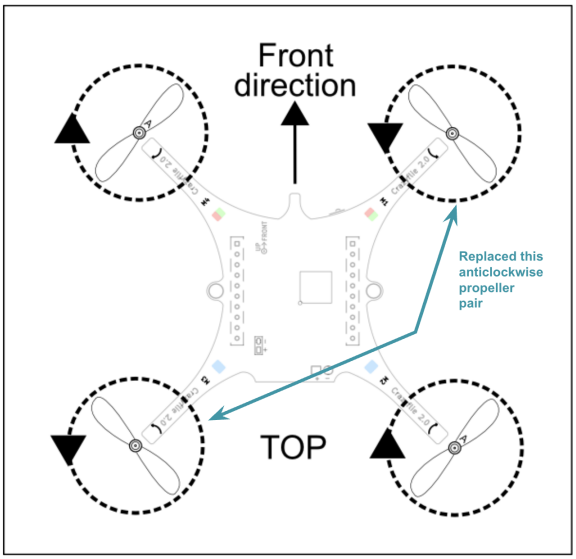


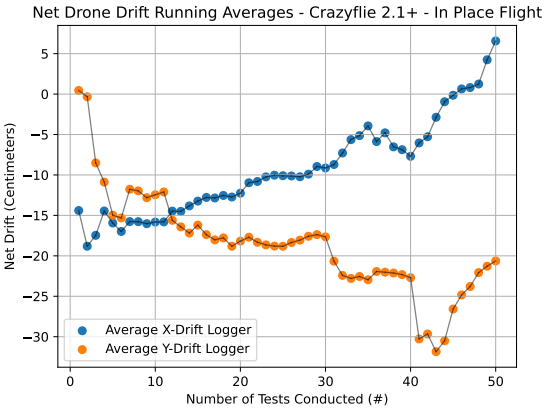
Figure 3: Replaced anticlockwise propeller pair (Adapted from Bitcraze Getting Started Guide [3])

presented in Fig. 4. The difference between flight drift indicates an inconsistency in flight. This implies that drift can affect a quadcopter in a different way each flight. For instance, one flight may cause the drone to drift by 10 cm in the rightwards direction while the next flight may cause a 10 cm drift in the leftwards direction. It is possible that a simple human touch to a propeller or to the drone's framework, altering drift's effect.

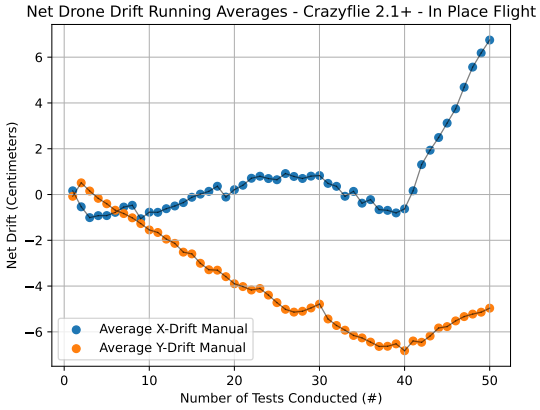
To ensure minimal to no drift and achieve precise flight behavior when the PMW3901 optical sensor of the Crazyflie's flow deck module detects a patterned or textured floor, we conducted an experiment using a custom nonuniform checkered pattern on the floor, as shown in Fig. 5. This setup allowed us to evaluate the optical sensor's effectiveness in stabilizing the Crazyflie's motion and if it could generate consistent optical flow data for better trajectory control and drift. However, the results in Fig. 13 showed no visible improvement, as drift persisted. This indicates that the checkered floor pattern did not enhance the performance of the PMW3901 optical flow sensor or effectively reduce drift in the Crazyflie's flight.

6 Conclusions

We empirically demonstrated that drift in Crazyflie quadcopters is undeniable. We also showed the inaccuracies in Crazyflie's inherent logger module and how it differed from our meticulous manual measurements. We demonstrated that by changing mechanical parts like propellors or the Crazyflie model to the latest one failed to eliminate or drastically reduce drift. Lastly, we proved that a checkered pattern on the floor yielded no noticeable reduction in drift or flight behavior. Since drift is unavoidable, it can be detrimental to flight safety and precise navigation. Hence, in our future work, we explore potential methods to combat drift and improve flight precision, safety, and reliability.



(a) Logger module: Running Average Plot



(b) Manual measurement: Running Average Plot

Figure 4: Summary plots showing variability of datasets Average Point for Logger Module Data: (6.57 cm, -20.64 cm); **Average Point for Manual Msmnt. Data:** (13.50 cm, -9.93 cm)

6.1 Future Work

This drone analysis has paved the way for critical improvements in Crazyflie drone flight trajectories. To counteract mechanically and externally induced drift, we can implement a simple program to stabilize flight. By continuously monitoring the drone's position, we can confine its movement within a predefined boundary, preventing excessive displacement. This ensures precise, stable, and controlled flight, enhancing overall performance and reliability.

Another effective way to counteract drift is by adjusting the drone's flight perspective based on a detailed drift analysis. For example, if the drone consistently drifts 'x' centimeters to the right when flying forward, a program can be designed to compensate for this deviation, ensuring a more accurate trajectory. However, this auto-correction is only viable in scenarios where the drift pattern remains consistent. For inconsistent drift, a more complex program can be designed to compensate and correct the flight path on the fly. Using any of these, the impact of drift on a quadcopter can be effectively mitigated.

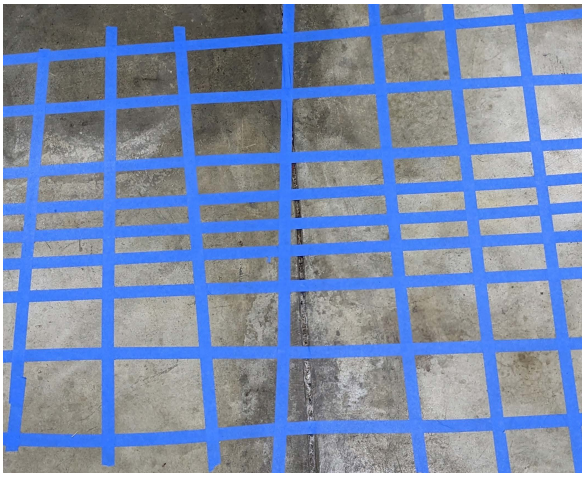


Figure 5: Nonuniform checkered pattern on floor

Furthermore, our work with Crazyflie models has opened the door to a wide range of innovative possibilities. One promising avenue is the implementation of a runtime monitoring framework like RTLola on small-scale drones, such as those used in this study. This integration could enhance safety mechanisms, allowing nano-quadcopters to operate beyond traditional boundaries while minimizing the risk of damaging expensive equipment. Additionally, the modular design of these drones makes them easily adaptable, enabling cost-effective prototyping and rapid development of new technologies.

References

[1] Baumeister, J., Finkbeiner, B., Schirmer, S., Schwenger, M., Torens, C.: Rtlola cleared for take-off: monitoring autonomous aircraft. In: Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part II 32. pp. 28–39. Springer (2020)

[2] Biediger, D., Mahadev, A., Becker, A.T.: Investigating the survivability of drone swarms with flocking and swarming flight patterns using virtual reality. In: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). pp. 1718–1723. IEEE (2019)

[3] Bitcraze: Getting started with the crazyflie 2.0 or crazyflie 2.1(+), https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/

[4] Lehto, M., Hutchinson, B.: Mini-drones swarms and their potential in conflict situations. In: 15th international conference on cyber warfare and security. vol. 12, pp. 326–334 (2020)

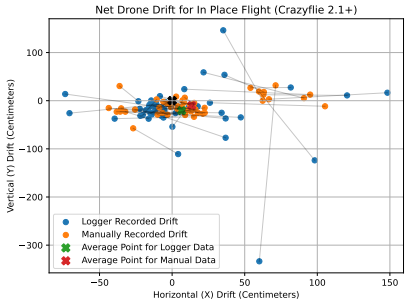
[5] Masalimov, K., Muslimov, T., Kozlov, E., Munasypov, R.: Crazypad: A dataset for assessing the impact of structural defects on nano-quadcopter performance. Data **9**(6), 79 (2024)

[6] Örnhag, M.V., Persson, P., Wadenbäck, M., Åström, K., Heyden, A.: Trust your imu: Consequences of ignoring the imu drift. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4468–4477 (2022)

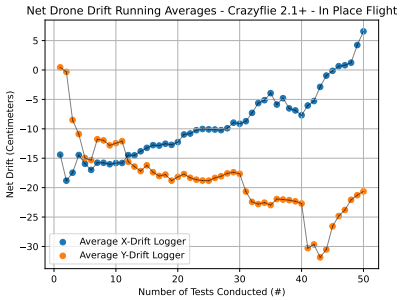
[7] Palossi, D., Conti, F., Benini, L.: An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-uavs. In: 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS). pp. 604–611. IEEE (2019)

[8] Zhou, X., Wen, X., Wang, Z., Gao, Y., Li, H., Wang, Q., Yang, T., Lu, H., Cao, Y., Xu, C., et al.: Swarm of micro flying robots in the wild. Science Robotics **7**(66), eabm5954 (2022)

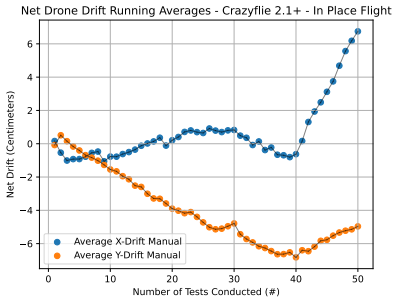
Received 06 January 2025; accepted 25 March 2025



(a) In Place Drift Plot

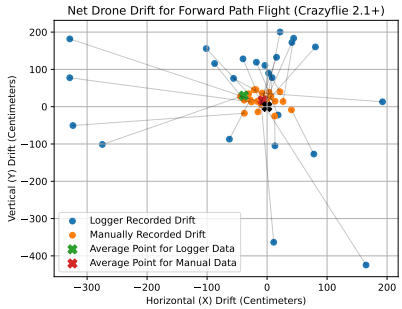


(b) Logger module: Running Avg. Plot

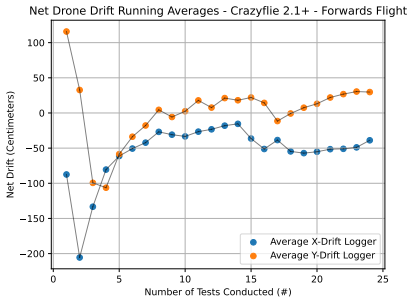


(c) Manual measurement: Running Avg. Plot

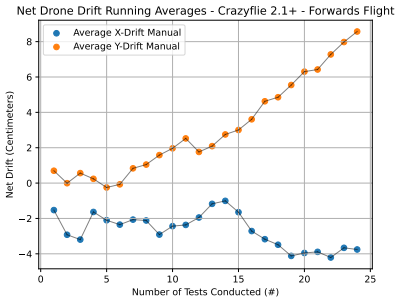
Figure 6: Crazyflie 2.1+ Plot - In-Place Flight
Outliers: None
Average Point for Logger Data: (6.57 cm, -20.64 cm); **Average Point for Manual Data:** (13.50 cm, -9.93 cm)



(a) In Place Drift Plot

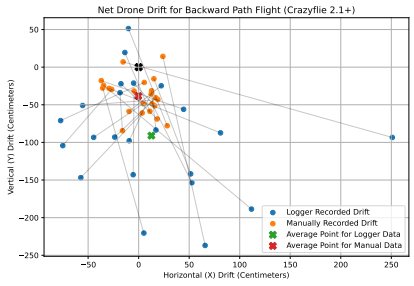


(b) Logger module: Running Avg. Plot

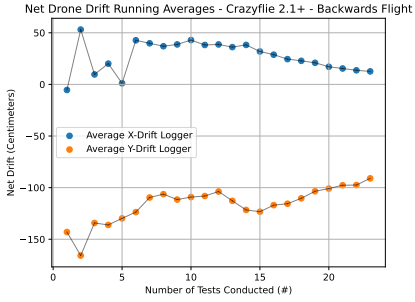


(c) Manual measurement: Running Avg. Plot

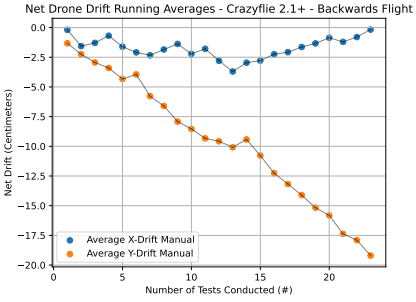
Figure 7: Crazyflie 2.1+ Plot - Forward Path Flight
Outliers: 1st Data Point Collected - (2.56 cm, 22.55 cm, 0.01 cm)
Average Point for Logger Data: (-38.86 cm, 29.74 cm); **Average Point for Manual Data:** (-7.51 cm, 17.14 cm)



(a) In Place Drift Plot

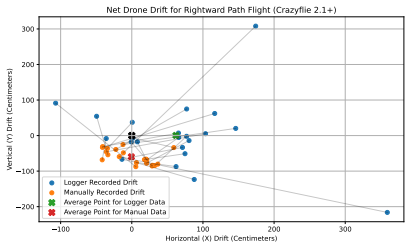


(b) Logger module: Running Avg. Plot

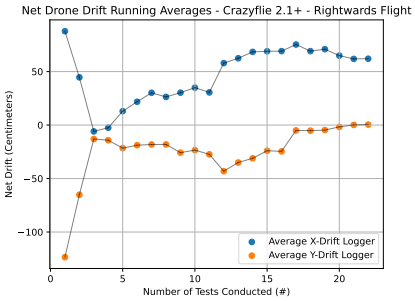


(c) Manual measurement: Running Avg. Plot

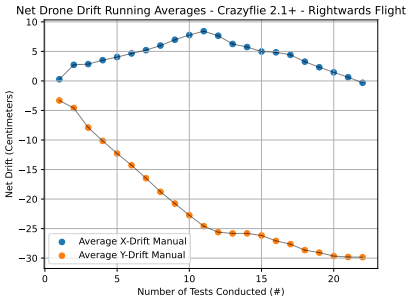
Figure 8: Crazyflie 2.1+ Plot - Backward Path Flight
Outliers: 1st Data Point: (-78.98 cm, -69.41 cm, 0.005 cm); 22nd Data Point: (-29.00 cm, 1.50 cm, 0.004 cm)
Average Point for Logger Data: (12.61 cm, -91.01 cm); **Average Point for Manual Data:** (-0.34 cm, -38.40 cm)



(a) In Place Drift Plot

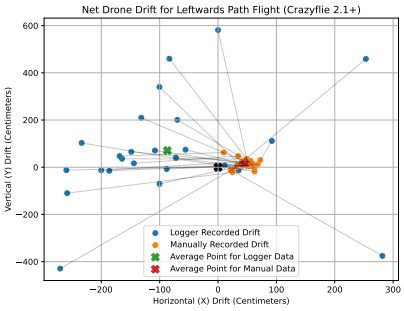


(b) Logger module: Running Avg. Plot

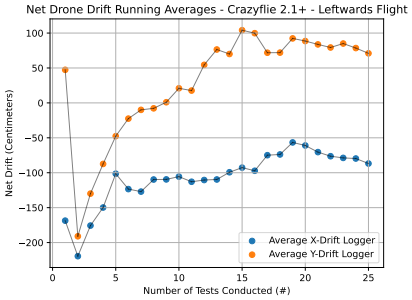


(c) Manual measurement: Running Avg. Plot

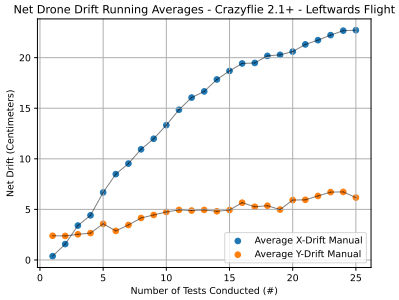
Figure 9: Crazyflie 2.1+ Plot - Rightward Path Flight
Outliers: 1st Data Point: (4.80 cm, 0.03 cm, 0.01 cm); 6th Data Point: (-5.71 cm, 1.98 cm, 0.008 cm); 12th Data Point: (-7.06 cm, 3.35 cm, 0.01 cm)
Average Point for Logger Data: (62.04 cm, 0.46 cm); **Average Point for Manual Data:** (-0.63 cm, -59.69 cm)



(a) In Place Drift Plot

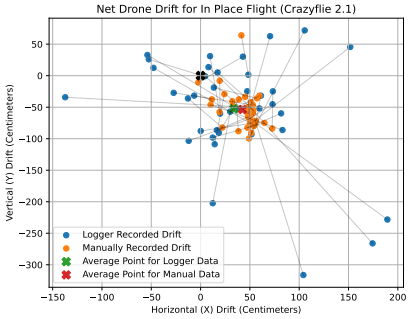


(b) Logger module: Running Avg. Plot

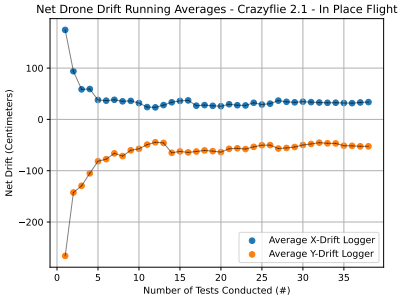


(c) Manual measurement: Running Avg. Plot

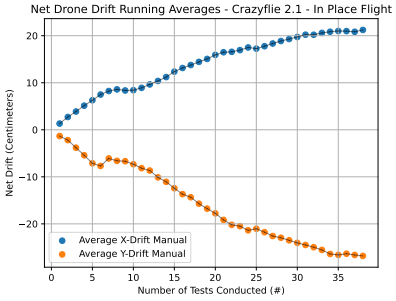
Figure 10: Crazyflie 2.1+ Plot - Leftward Path Flight
Outliers: None
Average Point for Logger Data: (-86.81 cm, 70.99 cm); **Average Point for Manual Data:** (45.43 cm, 12.33 cm)



(a) Drone Drift Plot

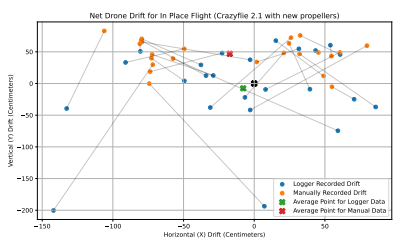


(b) Logger module: Running Avg. Plot

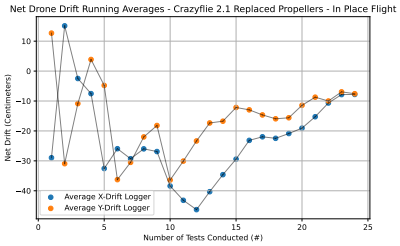


(c) Manual measurement: Running Avg. Plot

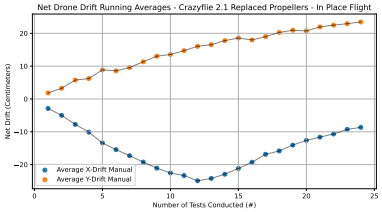
Figure 11: Crazyflie 2.1 Plot - In-Place Flight
Outliers: 26th Data Point: (0.26 cm, 0.53 cm, -3.07e-05 cm); 36th Data Point: (-15.18 cm, -33.78 cm, 0.004 cm)
Average Drift for Logger Data: (33.72 cm, -52.43 cm); **Average Point for Manual Data:** (42.46 cm, -53.62 cm)



(a) Drone Drift Plot

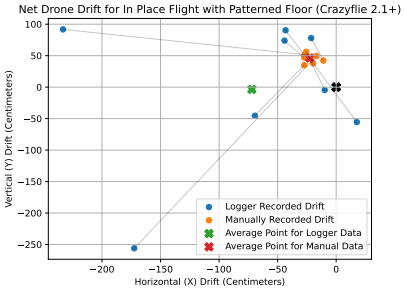


(b) Logger module: Running Avg. Plot

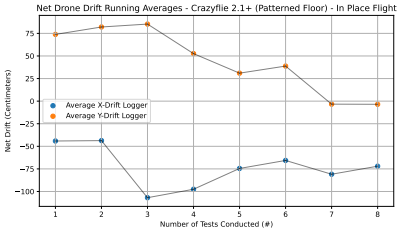


(c) Manual measurement: Running Avg. Plot

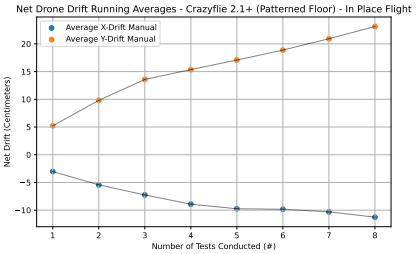
Figure 12: Crazyflie 2.1 with Replaced Propellers Plot - In-Place Flight
Outliers: 3rd Data Point: (-80.70 cm, 83.47 cm, 0.004 cm)
Average Point for Logger Data: (-7.80 cm, -7.57 cm); **Average Point for Manual Data:** (-17.30 cm, 46.93 cm)



(a) Drone Drift Plot



(b) Logger module: Running Avg. Plot



(c) Manual measurement: Running Avg. Plot

Figure 13: Crazyflie 2.1 with Patterned Floor Plot - In-Place Flight
Outliers: 3rd Data Point: (-4.86 cm, -3.92 cm, 0.01 cm); 6th Data Point: (-5.28 cm, 3.75 cm, 0.01 cm)
Average Point for Logger Data: (-55.83 cm, -104.25 cm); **Average Point for Manual Data:** (-21.10 cm, 42.976 cm)

LLM-based Code Generation for Verilog

Anshul Raghav
Redmond High School
Redmond, Washington, USA
anshul.raghav.019@gmail.com

Mark Santolucito
Barnard College, Columbia University
New York City, New York, USA
msantolu@barnard.edu

Abstract

Recent advancements in Large Language Models (LLMs) have transformed code generation across various programming languages. However, there remains a significant gap in research regarding their application to hardware description languages, particularly Verilog. This gap is especially critical as hardware development becomes increasingly complex and automated assistance could substantially reduce development time and minimize errors in system design.

To address this challenge, we conducted a comprehensive ablation study comparing GPT-4o, GPT-4o mini, and GPT-3.5 Turbo for Verilog code generation. Our methodology involved evaluating four distinct approaches: (1) one-shot generation as a baseline, (2) multi-shot generation, (3) an error correction pipeline for handling compilation errors, and (4) a full framework that addresses both compilation and functional errors. Our results demonstrate that the full framework significantly outperforms baseline approaches across all tested models.

The results show that the full framework outperformed all baseline methods, eliminating compilation errors entirely and increasing the overall pass rate by up to 17.65%. In the ablation study, the multi-shot generation and error-handling components demonstrated incremental improvements of 1.96% and 5.88%, respectively, underscoring the contribution of each component to the framework's effectiveness.

Keywords

Verilog, Code Generation, LLMs, Ablation Study

ACM Reference Format:
Anshul Raghav and Mark Santolucito. 2025. LLM-based Code Generation for Verilog. In *Proceedings of International Journal of Secondary Computing and Applications Research (IJSCAR VOL. 2, ISSUE 1).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.5281/zenodo.14988731

1 Introduction

Verilog is a popular hardware description language (HDL) widely used in electronic design automation to model and simulate digital circuits. It allows designers to describe the structure and behavior of digital systems at a high level of abstraction. However, writing functionally correct Verilog code can be challenging. Verilog requires a deep understanding of hardware-specific constructs, which are difficult to understand without a deep domain knowledge. These complexities make human coding prone to subtle errors that can lead to malfunctioning hardware or performance inefficiencies. LLMs offer promising potential in assisting with Verilog code generation by automating parts of the development process. LLMs can quickly generate code based on natural language prompts, reducing development time and aiding engineers in addressing complex design specifications.

Despite the success of LLMs in generating code for a wide range of languages [2, 3, 8, 13], generating Verilog code remains a challenge [7, 15, 16]. Single-shot generation, where the model produces code from a single input prompt, often fails to yield functionally accurate and optimized code [6]. These limitations reveal a clear need for a specialized framework that iteratively refines and validates code outputs, ensuring they meet the strict accuracy and performance standards essential for hardware design. Thus, we focus our work on the ability of LLMs to use compliler feedback and test feedback to iteratively refine code.

To address these challenges, we conducted an ablation study comparing GPT-4o, GPT-4o Mini, and GPT-3.5 Turbo for Verilog code generation. We introduce a novel framework designed to systematically address both compilation and functional errors by identifying specific issues and utilizing the LLM to generate targeted corrections. Our approach incorporates multi-shot generation and iterative refinement techniques to progressively improve code quality.

This paper aims to evaluate the effectiveness of LLMs in generating syntactically and functionally correct Verilog code and proposes a framework to enhance code generation accuracy. In summary, the key contributions of this paper are:

- We provide a comprehensive evaluation of LLMs for Verilog code generation, including an ablation study and a full framework for improved accuracy.
- We identify specific challenges in generating functionally correct Verilog code and provide insights for future research in this area.
- We make our code and datasets publicly available to facilitate further research in this domain.

2 Related Work

The advent of LLMs has markedly transformed the landscape of automated code generation, significantly altering the methodologies and practices employed by developers in the software engineering domain. Among the most compelling applications of LLMs is their capacity to facilitate code generation, which holds tremendous upside and has garnered interest from both academia and industry alike [1–3, 5, 7, 8, 10, 13, 15–17, 19]. General-purpose tools, such as ChatGPT [10], Github Copilot [5], and Claude [1], have surged in popularity in recent years, largely attributable to their user-friendly interfaces and versatility across various programming tasks. According to a survey conducted by GitHub, 92% of developers report

IJSCAR VOL. 2, ISSUE 1

using AI-based coding tools either in their professional or personal projects, with 70% believing that such tools offer them an advantage in their work [14]. These findings underscore a broader trend of widespread adoption and positive reception of LLM-based code generation. This phenomenon has prompted extensive research aimed at evaluating the performance of domain-specific LLM-based code generation and identifying approaches to enhance their effectiveness.

While general-purpose tools can perform a broad range of tasks, they often struggle with generating code that requires specialized domain knowledge, particularly for less commonly used programming languages. As a result, in parallel with the development of general-purpose applications, there has been a notable increase in the development of research designed to evaluate the quality of code generated for specific programming languages and improve their performances. Research efforts have led to the development of diverse benchmarks as standardized tests to evaluate the accuracy and efficiency of code generation by LLMs across various tasks. Among these, HumanEval has emerged as the most widely recognized benchmark in code generation research [2]. Additionally, it is a common practice to fine-tune certain LLMs for specific programming languages [13]. This fine-tuning process involves training models on language-specific datasets, allowing them to better capture the language's syntax and structure.

To further improve the syntactical and functional accuracy of code generated by LLMs, researchers have introduced comprehensive frameworks. SynCode [17] exemplifies this approach, enabling LLMs to produce syntactically correct code in various formal languages by integrating user-defined context-free grammars. Tools such as RustAssistant [3] and RTLFixer [16] also contribute to this trend, leveraging techniques such as iterative refinement, retrieval-augmented generation (RAG) [4], and ReAct prompting [19] to address and improve upon initial code outputs. By systematically refining the generation process, these frameworks enhance the reliability and effectiveness of LLM-assisted coding.

Narrowing the focus further, there exists a burgeoning interest in the application of LLMs specifically for hardware description languages (HDLs), such as Verilog. The domain-specific knowledge needed for Verilog along with its unique syntax and semantics presents a unique challenge for LLMs, necessitating the development of targeted models that can reliably generate correct and functional Verilog code. Previous studies have proposed benchmarking frameworks to evaluate LLM-generated code [7], while also fine-tuning and testing older models [15], but there is a lack of recent and comprehensive research into tracking the efficacy of LLM-based generation of Verilog code and developing a framework to improve the correctness of generated code.

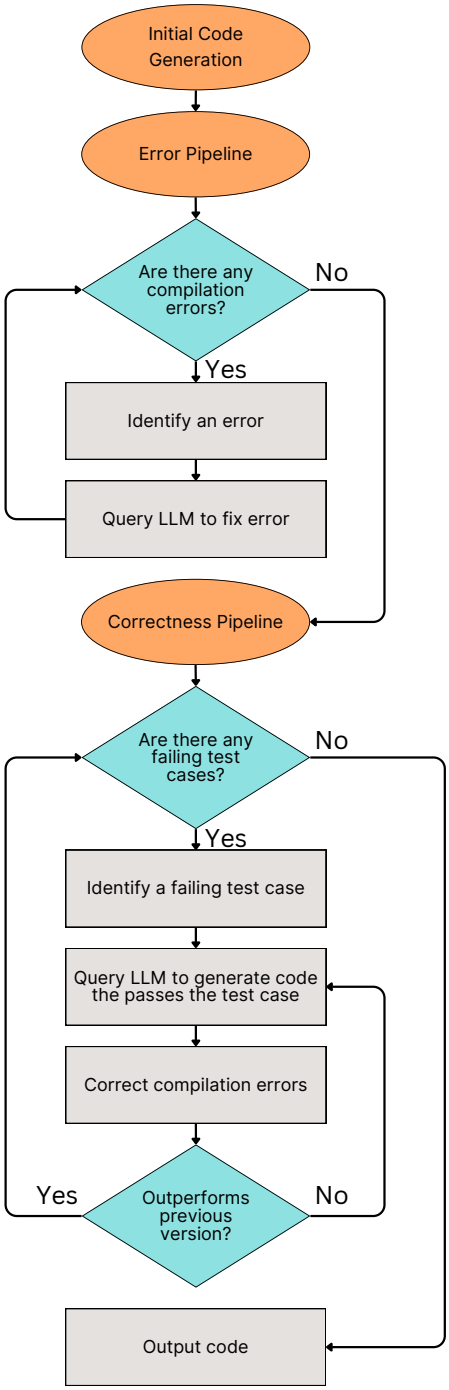


Figure 1: Code generation framework

3 System

3.1 Framework Overview

As shown in Fig 1, the framework is divided into three components that build off each other to generate functionally correct code at the highest possible rate while consuming a minimal amount of resources. At a high level, the pipeline is designed to generate initial code and continuously address specific issues, providing targeted fixes until either the code is fully functional or the maximum resource capacity has been reached. This framework leverages the OpenAI API to implement the targeted corrections, specifically utilizing the GPT-4o-2024-08-06, GPT-4o-mini-2024-07-18, and GPT-3.5-turbo-0125 models, the latest versions at the time of writing.

It is important for this tool to be able to be subjected to realistic constraints, the most important of which being a limited number of API queries. Queries incur both time and financial costs, so it is important to be able to control the maximum number of queries to the LLM while retaining optimal performance. By allowing the LLMs to build on their previous responses with feedback from the compiler or the test cases, we found that the model utilized the given resources effectively to improve upon the initially generated code.

Initial Generation: The goal of this module is to generate a starting point that we can build off of in the following components. We query the LLM with a general template to give it context into the actions it needs to take and the specified problem, and output this result directly to the error pipeline.

Error pipeline: The purpose of the error pipeline is to remove any compilation errors from the code while still maintaining the initial functionality. At a high level, this pipeline includes two main sections - error identification and error correction. The framework will initially identify any compilation errors in the code, advancing directly to the functional pipeline if there are no errors. Once a specific error is identified, it will transition to the error correction phase and query the LLM to provide a targeted fix. Then, this process will repeat from the beginning, with another issue being identified and corrected until the code compiles successfully, at which point it will advance to the correctness pipeline.

Correctness pipeline: The objective of the correctness pipeline is to produce functionally correct code, meaning for each input the code produces an output satisfying the specification. It builds on top of the error pipeline, taking in compilable code that may or may not be functionally correct. The pipeline begins by assessing whether any test cases are failing. If the code is functionally correct, it outputs the code as the final solution. In the event of a failure, the pipeline identifies the specific test case that is not passing, queries the LLM to generate a targeted adjustment aimed at resolving the identified issue, and finally subjects the code to the error pipeline to ensure that it is compilable. A prevalent issue observed was the LLM's tendency to inadvertently introduce regressions into the code while attempting to address an issue, causing the code to fail previously passed test cases and rendering it less effective than the prior iterations. In such a scenario, the code would be reverted back to the previous version and the LLM would be queried again in an attempt to resolve the issue. This iterative process of identifying failing test cases and applying corrections occurs until the code successfully passes all test cases. However, in some instances, the model exhibited repetitive behavior, consistently "hard-coding" the same response regardless of the number of queries. To mitigate this issue, we adjusted the temperature parameter of the LLM, increasing response variability and encouraging the generation of alternative solutions. Additionally, we implemented a fixed limit on the number of iterations the model could spend addressing a single issue. This constraint prevented the system from entering infinite loops, thereby optimizing computational efficiency and preventing unnecessary resource consumption.

3.2 Problem Simulation

To effectively showcase the functionality and structure of the framework, we begin by presenting a sample problem. This problem serves as a practical example through which the framework's core processes can be demonstrated. Through a systematic analysis, we will demonstrate the framework's capacity to address challenges in code generation, highlighting its ability to methodically refine the code to produce a fully functional solution.

As shown in Fig 2, the problem outlines a Verilog module for a signed adder that takes two 8-bit 2's complement numbers (a and b) as inputs, produces their sum (s), and detects if a signed overflow occurs during the addition (overflow). The key contribution of the signed adder is to account for the signs of the numbers while adding, with the most significant bit (MSB) representing the sign of the value. A signed overflow occurs when the sum of the numbers exceeds the range for the given number of bits, causing the number to wrap around and produce an incorrect result.

```
// This is a signed adder that adds two 8-bit 2's
     complement numbers. It also captures a signed
     overflow.
module signed_adder(input [7:0] a, input [7:0] b, output
     [7:0] s, output overflow );
```

Figure 2: Problem statement

Initial Generation: The code produced by the initial generation of the framework is illustrated in Fig 3. This code implements the majority of the details correctly, but contains minor compilation and functional errors. Errors during the initial generation phase are anticipated, as this stage serves as a foundational starting point upon which the error and correctness pipelines can build.

```
1  module signed_adder(input [7:0] a,
2      input [7:0] b,
3      output reg [7:0] s,
4      output wire overflow
5  );
6
7      assign s = a+b;
8      assign overflow = a[7]&&b[7]&&(~s[7]);
9
10 endmodule
```

```
.\compilation-error.v:7: error: reg s; cannot be driven
     by primitives or continuous assignment. 1 error(s)
     during elaboration.
```

Figure 3: Initial generation code

Error Pipeline: The initial step in the error pipeline involves verifying the presence of any compilation errors. As identified in Fig 3, the error arises from attempting to drive a variables with type reg (s) using a continuous assignment, which is prohibited in Verilog. To resolve this, the variable s must be declared as wire [20]. This specific error, along with the relevant code, is provided to the LLM, which is tasked with generating a targeted correction to address the issue. The results of this operation can be seen in Fig 4, along with its failing test case. The process subsequently

advances to the initial step once again, attempting to identify if there are any compilation errors. As the code is compilable, the process transitions to the correctness pipeline.

```
module signed_adder(input [7:0] a,
    input [7:0] b,
    output [7:0] s,
    output overflow
);

    assign s = a+b;
    assign overflow = a[7]&&b[7]&&(~s[7]);

endmodule
```

```
Case 3 failed
Input:
a=8'b01000000; b=8'b01000000;
#period;
Correct output: s=8'b10000000 overflow='b1
Given output: s=8'b10000000 overflow='b0
```

Figure 4: Error pipeline code

Correctness pipeline: The first steps in the correctness pipeline are to identify if there are any failing test cases, which is made clearly evident in Fig 4. The failing test case results from a functional error in the code, specifically its failure to detect overflow when both a and b are positive, but their signed sum evaluates to a negative value. This error occurs because overflow is only identified when a and b are positive and s is negative, but not under the inverse conditions. Upon identifying the failing test case, the LLM is queried to generate code that passes the specified test case, while any compilation errors are addressed using the previously covered error pipeline. The output of these operations is illustrated in Fig 5. However, the solution remains functionally incorrect due to a flaw in the overflow detection logic. Specifically, overflow is assigned a value of 1 when the sign of a differs from the sign of the sum (s). This approach is demonstrably incorrect, as evidenced by the failing test case provided. For instance, when a is -5 and b is 50, the sum is 45, and overflow is incorrectly set to 1 due to the mismatched signs of a and s. In this scenario, overflow should instead be 0, as no overflow occurs. This issue is addressed in the subsequent stage of the correctness pipeline, which involves comparing the updated solution with its previous iteration. The LLM is then queried once more to generate code that successfully passes the failing test case. The newly generated code, as shown in Fig 6, incorporates the revised logic: assigning an overflow value of 1 when both a and b are negative and the sum is positive. Upon evaluation, this version is found to outperform the previous iteration and pass all of the test cases, allowing the code to be designated as the final version.

As demonstrated, this process of iterative refinement proves to be effective. By employing a minimal number of queries, it was possible to generate fully functional code. This gradual, targeted approach ensures a more efficient use of resources and increases the likelihood of generating a correct and optimized solution.

3.3 Framework Implementation

This section offers a detailed examination of the framework's logic and structure. Designed as a command-line tool, the framework

```
module signed_adder(input [7:0] a,
    input [7:0] b,
    output [7:0] s,
    output overflow
);

    assign s = a + b;
    assign overflow = s[7] != a[7];

endmodule
```

```
Case 2 failed
Input:
a=8'b11111011; b=8'b00110010;
#period;
Correct output: s=8'b00101101 overflow='b0
Given output: s=8'b00101101 overflow='b1
```

Figure 5: Regressed solution code

```
module signed_adder(input [7:0] a,
    input [7:0] b,
    output [7:0] s,
    output overflow
);

    assign s = a+b;
    assign overflow = a[7]&&b[7]&&(~s[7]) || (~a[7])&&(~b
     [7])&&(s[7]);

endmodule
```

```
All cases passed
```

Figure 6: Full solution code

accepts input through a text file and generates output stored in a Verilog file. Several details are abstracted or slightly modified in this section to better explain the overall structure of the code, the most important being the underlying functions such as checkErrs(), checkCorrectness(), and passingCases(). Despite not being included in the pseudocode, these are covered in detail and available as separate files in the GitHub repository [12]. Furthermore, the prompts provided to the LLM to enhance its understanding of both and functional errors are embedded as variables, such as errMessage and correctnessMessage. While these variables are not explicitly visible during the model's decision-making process, they have been systematically fine-tuned on a diverse set of sample problems to maximize comprehensibility. This careful tuning ensures that the LLM receives clear and structured feedback, allowing it to develop a form of "chain of thought" reasoning [9]. Empirical observations indicate that as the clarity and specificity of these prompts increased, the model's overall performance in resolving errors exhibited a corresponding improvement.

Initial Generation: The queryLLM function utilizes the prompt and queries the chosen LLM a certain of times, returning the best result, i.e., the one with the least errors and highest number of test cases passed (line 38 of pseudocode in Fig 7). The number of LLM queries is determined by the user-defined parameter numShots, which can be adjusted based on design specifications. The input

prompt is constructed as startMessage + prompt, where startMessage provides contextual information and specifies the required output format, while prompt defines the specific problem to solve.

Error Pipeline: The purpose of the error pipeline is to remove all errors from the code while still maintaining the intended functionality (line 40). It takes in two parameters, the code and the problem statement and outputs a revised version of the code that is error free. The central concept of this pipeline is that the LLM can process feedback in the form of error messages and use it to correct the code, eliminating the identified errors. One particularly intriguing aspect of Verilog is that its compilation errors often lack clarity, presenting a significant challenge for human developers during the debugging process. This raises the question of whether LLMs would encounter similar difficulties when tasked with interpreting and resolving such errors.

The algorithm starts by checking the code for errors and storing those in a set (line 5). This approach ensures that duplicate errors are eliminated and facilitates rapid error lookups. In line 6, we establish a separate set for errors deemed irreparable. Failure to resolve an error may arise from multiple factors, such as a lack of domain specific knowledge, ambiguity in the error, or misunderstanding the intent. The consequence is that the LLM is unable to resolve the issue, resulting in a potential waste of valuable time and resources if the model is continuously queried without success. The consequence is that the LLM is unable to resolve the issue, resulting in a potential waste of time and resources if the model is continuously queried without success. Consequently, errors that the LLM cannot address are categorized into failedSet on line 17. Notably, these unresolved errors are often rectified by the subsequent resolution of other underlying errors that initially contributed to their emergence.

Lines 7-20 are where the body of the error-fixing process occurs. We continuously iterate through the loop as long as there are remaining errors in the code, selecting a specific error for resolution with each iteration. Given that modifications to the code can lead to the emergence of new errors during the correction of existing ones, we implement an abstraction termed an error group to effectively manage this complexity (line 8). As we proceed with the resolution process, any newly encountered errors that arise will be incorporated into the error group (line 15). This iterative process continues until the error group is completely emptied. As previously noted, there are instances in which the LLM is unable to rectify certain errors. We classify the error as unfixable if it persists despite revisions aimed at resolving the issue (line 16-19). Consequently, should we encounter this unresolvable error again during our examination of the error group, we will simply disregard it and move onto the next error (lines 12-13).

We implement several heuristics to ensure that the code remains within a specified query limit. For clarity, these heuristics are not detailed in the pseudocode. We impose a maximum limit for the number of times the outer loop can run (line 7) and allow the user to define a fixed number of iterations for the inner loop (line 10). Additionally, when the LLM is queried, it executes a user-defined number of queries, represented by numShots, and selects the optimal response from among them (line 14).

Correctness Pipeline: This pipeline accepts error-free code along with the problem statement and produces functionally correct code that successfully passes all test cases (line 42). Prior to initiating the pipeline's core processes, three key concepts are incorporated. First, the pipeline maintains a record of all previously passed test cases, enabling verification that the current code performs at least as well as prior iterations (line 24). Since the LLM occasionally generates revisions that inadvertently reduce code performance, causing earlier test cases to fail, this measure serves as a safeguard against regression. Second, we incorporate a version control mechanism, adapted from the error pipeline (line 26). This allows us to revert the code to its prior state if earlier test cases begin to fail. Third, we track the current failing test case, which is later provided to the LLM as feedback, guiding it in refining the code to pass additional test cases (line 25).

The main processing occurs within the while loop, where we iterate through the code body until all test cases are successfully passed, as indicated by the checkCorrectness() function returning 'all cases passed' (line 27). The information from the failing test cases is used to guide the LLM in generating an improved version of the code that addresses the specific failure. We provide the LLM with a structured prompt that includes contextual information on deriving and formatting the solution, as well as the current code, the failing test case, and the original problem statement (line 28). Including the problem statement is crucial, as it helps the LLM remain focused on the intended solution; without it, we found that the LLM tends to 'hardcode' responses, inadvertently causing other test cases to fail. Next, we apply the error pipeline to the newly generated code to ensure it is free from errors (line 29). Following this, we assess whether the code has regressed: if it fails a test case it previously passed, we employ version control to revert to a more functional version (lines 30-32). If no regression is detected, we can confirm that the current code represents the best iteration to date. Consequently, we save this version and update the set of solved test cases (lines 33-35). Finally we return the completed code at the conclusion of the process (line 36).

We employ heuristics similar to those in the error pipeline to constrain the number of queries sent by the code. The primary heuristic is a fixed limit on the number of while loop iterations (line 27), which is set to a constant value configurable by the user.

4 Evaluation

4.1 Results

4.1.1 Experimental Design. To evaluate the effectiveness of our framework for generating Verilog code, we selected a series of problem sets and benchmarks from [15]. These benchmarks offer a thorough evaluation of code generation accuracy, and the problem set is well-suited for examining the performance of LLMs in hardware design tasks.

The problem set in VGen is systematically categorized into three levels of difficulty: basic, intermediate, and advanced, ensuring a progressive evaluation of the LLM's Verilog code generation capabilities. The dataset consists of 17 distinct problems, each designed to cover a broad range of digital design concepts, including combinational and sequential logic, arithmetic operations, and finite state machines (FSMs). This diverse selection ensures a comprehensive

```
Input: Problem description
Output: Code that has no compilation errors and behaves according to the specifications of the problem description

def errPipeline(code, problem):
    errs = checkErrs(code)
    failedErrs = {}
    while errs != {}:
        errGroup = {errs.pop()}
        snap = code
        while errGroup != {}:
            currentErr = errGroup.pop()
            if currentErr not in checkErrs(code):
                continue
            code = queryLLM(errMessage+currentErr+code+problem, numShots)
            errGroup = checkErrs(code)-errSet
            if currentErr in errGroup:
                failedErrs.append(currentError)
                errGroup.remove(currentErr)
                code = snap
        errSet = checkErrs(code)-failedErrs
    return code

def correctnessPipeline(code, problem):
    solvedCases = {}
    failingCase = checkCorrectness(code)
    snap = code
    while failingCase != "all cases passed":
        code = queryLLM(correctnessMessage+code+failingCase+problem, numShots)
        code = errPipeline(code)
        failingCase = checkCorrectness(code)
        if failingCase in solvedCases:
            code = snap
        else:
            snap = code
            solvedCases = passingCases(code)
    return code

code = queryLLM(startMessage+prompt, numShots)

code = errPipeline(code, problem)

code = correctnessPipeline(code)
```

Figure 7: Pipeline pseudocode

assessment of the LLM's ability to generate Verilog across various complexity levels.

Each problem is further subdivided into three distinct prompts, all addressing the same core problem but differing in the level of explicit guidance provided. The prompts range from highly specific instructions, which closely outline the implementation strategy, to open-ended problem descriptions that require the model to independently determine the appropriate approach. This structured variation enables an analysis of how prompt specificity impacts the LLM's performance and reasoning ability in Verilog code generation.

To ensure objective evaluation, each problem is accompanied by a dedicated testbench that verifies the functional correctness of the generated Verilog code. These testbenches consist of multiple test cases, systematically designed to evaluate different edge cases and functional requirements within each problem. The testbenches serve as a benchmark for correctness, allowing for a clear and standardized assessment of the generated code's validity. However, in our methodology, the testbenches are deliberately excluded from assisting the LLM in generating functionally correct code.

A contribution of this work is the introduction of individual test cases for each problem, aimed at enhancing the rate of functionally correct code generation. In real-world applications, users interacting with an interface often lack access to predefined testbenches to verify functional correctness. To simulate this scenario, the LLM was not granted access to the testbenches during the generation process. However, our findings revealed that without test cases available for the LLM to be able to identify issues and provide targeted fixes, the LLM produced functionally incorrect code at an alarmingly high rate. To address this challenge, we developed a program that enables users to create their own test cases efficiently. The program allows users to specify parameters such as the module name, input and output names and values, the number of clock periods, and the period length. Based on these inputs, the program generates a fully functional testbench. This user-generated testbench can then be utilized by the LLM to verify the correctness of its code and iteratively refine it if errors are detected. By streamlining the creation of test cases, this approach addresses the critical need for functional verification while ensuring ease of use for users.

The framework was evaluated on the problem set using three distinct OpenAI models: GPT-4o, GPT-4o mini, and GPT-3.5 Turbo. GPT-4o is designed for complex, multi-step tasks, and is the most resource-intensive out of the three models. GPT-4o mini is a more lightweight and affordable version of GPT-4o, and is better for fast, lightweight tasks. Finally, GPT-3.5 Turbo is a fast, inexpensive model for simple tasks.

4.1.2 Ablation Study. The purpose of this ablation study is to isolate and assess the impact of each component in the code generation framework on the quality and reliability of LLM-generated Verilog code. This approach allows us to understand the role of each component in improving code quality and to identify which elements are most essential for generating functionally correct Verilog code.

To evaluate the impact of each component in our code generation framework, we define a baseline approach and consistent evaluation metrics. The baseline for this ablation study is the pass@1 metric, which we will refer to in this paper as the one-shot generation method. In this, the models respond to each problem with a single prompt, without any error handling or correctness validation. This baseline serves as a benchmark, allowing us to clearly see the relative impact of the other components.

For evaluating each component's contribution to performance, we rely on two key metrics: compilation success rate and functional correctness rate. Compilation success rate measures whether the generated Verilog code is syntactically correct and free of compilation errors. Functional correctness rate, on the other hand, assesses whether the generated code performs as expected in terms of functionality by passing the predefined testbenches. Each subsequent component - multi-shot prompting, error correction, and functional correction—is evaluated relative to this baseline, allowing us to observe how each addition improves or impacts the baseline performance.

The framework comprises of three main components: initial generation, the error correction pipeline, and the correctness pipeline. Each component builds upon the previous one to iteratively enhance the model's output. In the following, we describe each component and its role in generating high-quality Verilog code. Additionally, single-shot generation is included as a baseline for comparison. The initial generation is referred to as multi-shot generation to more precisely capture its function.

- **One-Shot Generation (Baseline):** This approach serves as our baseline and involves presenting each problem to the model once, generating a single response without any follow-up. With no additional context, examples, or error handling, this method captures the model's performance in generating initial Verilog code.
- **Multi-Shot Generation:** This component improves upon the baseline by querying the LLM multiple times and selecting the best result—defined as the output with the fewest compilation errors and the highest number of passed cases. From the perspective of the code generation diagram presented in the System section 1, this corresponds solely to the initial generation component.
- **Error Pipeline:** This pipeline extends the previous approach by iteratively refining its output to eliminate errors, as detailed in the system section. Notably, it does not address functional errors. This encompasses both the initial generation and error pipeline components found in the code generation diagram.
- **Full Framework:** Building upon the error pipeline, the full framework integrates all components discussed in the system section. Its primary advancement lies in its capability to identify and resolve functional errors effectively. This pipeline encompasses all components depicted in the code generation diagram.

With these components defined, we now turn to the results of our ablation study. The following figures provide a comprehensive overview of the framework's overall performance, illustrating how each component influences the models' ability to generate functionally correct Verilog code, as well as comparing the relative success rates of each model.

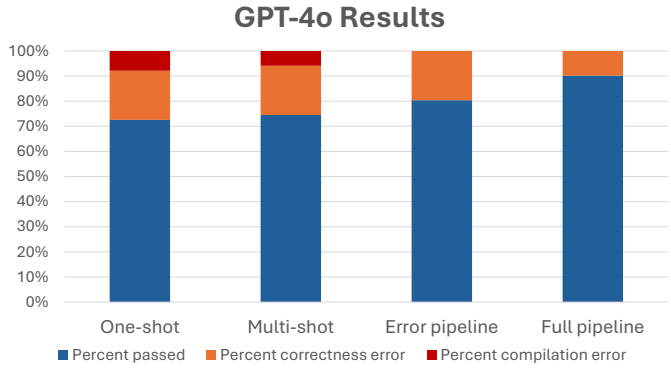


Figure 8: GPT4o Success Rates Graph

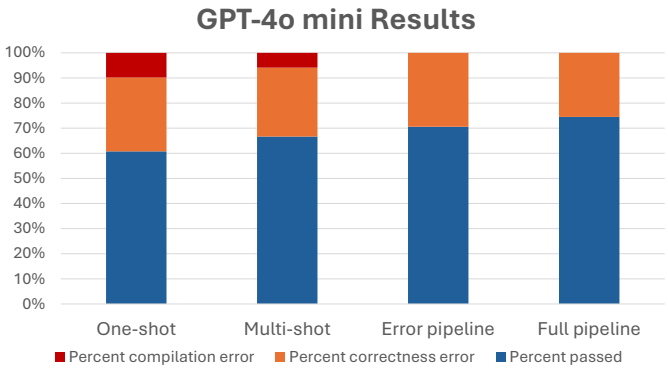


Figure 9: GPT-4o mini Success Rates Graph

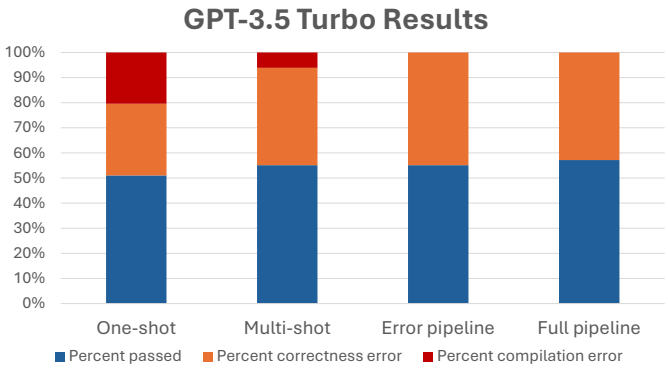


Figure 10: GPT-3.5 Turbo Success Rates Graph

4.2 Discussion

To fully assess the impact and effectiveness of the framework developed in this study, it is crucial to analyze the key findings and their broader implications. The discussion will highlight key takeaways and insights derived from the results, offering a more thorough understanding of the effectiveness of the framework, its components, and the various LLMs utilized.

To assess whether the framework provided a meaningful improvement over single-shot generation, it is necessary to evaluate the difference in the rate of fully functional code generated between the two approaches. At a high level, the improvements observed between the initial code generation and the final code output varied considerably across the different LLM models, yet remained consistently substantial. GPT-4o exhibited the most pronounced improvement, with a 17.65% increase, underscoring its advanced reasoning capabilities and its ability to make precise adjustments to the generated code. The framework also showed a significant improvement when paired with GPT-4o mini, which achieved a 13.73% improvement in success rates from the initial to final outputs. In contrast, GPT-3.5 Turbo exhibited a comparatively poor performance of 6.12%, reflecting its more limited capacity for reasoning and error correction. Overall, the framework proved highly effective, as demonstrated by the substantial improvements across

	One-shot	Multi-shot	Error pipeline	Full pipeline
Percent passed	72.55%	74.51%	80.39%	90.20%
Percent correctness error	19.61%	19.61%	19.61%	9.80%
Percent compilation error	7.84%	5.88%	0%	0%

Table 1: GPT-4o Success Rates Table

	One-shot	Multi-shot	Error pipeline	Full pipeline
Percent passed	60.78%	66.67%	70.59%	74.51%
Percent correctness error	29.41%	27.45%	29.41%	25.49%
Percent compilation error	9.80%	5.88%	0%	0%

Table 2: GPT-4o mini Success Rates Table

	One-shot	Multi-shot	Error pipeline	Full pipeline
Percent passed	51.02%	55.10%	55.10%	57.14%
Percent correctness error	28.57%	38.78%	44.90%	42.86%
Percent compilation error	20.41%	6.12%	0%	0%

Table 3: GPT-3.5 Turbo Success Rates Table

all models. Notably, when paired with more advanced LLMs such as GPT-4o, the framework achieved particularly significant gains.

It is also important to consider the role each component of the framework played in the final improvement. Initially, one-shot generation yielded mixed results, with a significant proportion of cases encountering either compilation or functional errors. This pattern was consistent across all models, although the frequency of these errors diminished as the models' complexity increased, again underscoring the significant influence that model sophistication has on error reduction.

Multi-shot generation produced results similar to those of single-shot generation, but with a notable decrease in both compilation and functional errors. This would have ensured a more structured foundation for the framework, guiding it toward more accurate and reliable solutions and minimizing the risk of diverging into ineffective or irrelevant paths. An interesting observation was the behavior of GPT-3.5 Turbo, which exhibited a unique pattern where the number of compilation errors decreased significantly, only to be replaced by functional errors. This shift suggests that the inherent limitations of GPT-3.5 Turbo in reasoning and functional comprehension may hinder its ability to effectively address functional errors, which demand more advanced reasoning capabilities.

Next, the error pipeline proved highly effective, successfully eliminating all compilation errors across all three models, highlighting its crucial role within the framework. The reduction in compilation errors amounted to approximately 5-6%, with the majority of these errors subsequently transitioning into functional errors. However, in some instances, the generated code passed all test cases and the functional errors were entirely resolved, achieving full functionality. This further underscores the importance of the error pipeline in improving the overall performance and reliability of the generated code.

The correctness pipeline also played a significant role in the overall final improvement. However, the improvement was more varied, at about 2-10% depending on the complexity of the model. As noted earlier, more advanced models, such as GPT-4o, exhibited greater accuracy improvements in comparison to simpler models such as GPT-3.5 Turbo. Our findings indicate that these issues were inherent with GPT-3.5 Turbo, as regardless of the number of queries, GPT-3.5 Turbo consistently struggled to provide effective fixes for the more complex problems within the dataset. However, the correctness pipeline remained a crucial component of the overall framework, making a significant contribution to the observed improvements.

A critical analysis of the performance of various LLM models is essential for understanding the nuances of their effectiveness within the framework. By systematically comparing the results from GPT-3.5 Turbo, GPT-4o, and GPT-4o mini, it is possible to assess the relative strengths and limitations of each model in the context of Verilog code generation. All three models were successful in eliminating compilation errors, demonstrating their ability to generate syntactically correct Verilog code. However, there was significant variation in their performance with respect to functional errors and their capacity for improvement over time. GPT-4o emerged as the most effective model, demonstrating the highest success rate in generating functionally correct code, as well as the greatest improvement in performance. This highlights its superior ability to not only produce accurate code but also to leverage reasoning to refine and enhance the code. In contrast, GPT-3.5 Turbo demonstrated the poorest performance, struggling to achieve consistently accurate results or improve upon the initial code. GPT-4o mini, while less powerful than GPT-4o, struck a balance by offering a competitive performance with a significantly lower computational cost. This makes GPT-4o mini an appealing option for scenarios where resource efficiency is a priority without severely compromising on accuracy.

Success rate was used as the evaluation metric rather than pass@k because the framework allowed for automatic requerying, making

it difficult to identify the top k results. Unlike traditional retrieval tasks, where a fixed set of top results can be ranked and evaluated, this framework operates iteratively, with each requery refining the solution to a compilation or functional error. As a result, there is no clear "top k" set of results; instead, the model continually evolves its responses, providing a series of progressively improved fixes until the issue is resolved. The pass@k metric, which relies on ranking a set of results at a given moment, becomes unsuitable for this dynamic process where the optimal solution is reached through successive refinements. Therefore, success rate provides a more accurate measure of the system's performance, focusing on whether the framework ultimately succeeds in fixing the identified errors rather than how well it ranks intermediate fixes.

While the results of this study demonstrate promising improvements in LLM-assisted Verilog generation, it is important to acknowledge the limitations of the research. Understanding these constraints provides valuable context for interpreting the findings and identifying areas for future development. One key limitation of this study is the relatively narrow range of LLMs used, which may have constrained the understanding of model performance. Expanding the selection to include additional models, such as OpenAI's o1, could offer a more comprehensive view, especially given its demonstrated success on human exams and ML benchmarks [11]. Additionally, incorporating state-of-the-art models such as Claude 3 Opus, GitHub Copilot, and Grok would enable a more comprehensive comparison, providing deeper insights into their reasoning capabilities and effectiveness in Verilog code generation. This study could also have been extended to include a broader range of programming languages, particularly HDLs closely related to Verilog, such as SystemVerilog and VHDL. Given SystemVerilog's similarities to Verilog, it would likely exhibit comparable behavior. However, its stricter type enforcement and explicit variable declarations could enhance an LLM's ability to debug code by providing more detailed and structured error messages. In contrast, VHDL, which originates from Ada—a less commonly utilized programming language—may pose greater challenges for an LLM due to its limited representation in training data. However, its strong typing system could enhance the debugging process by imposing strict constraints, thereby minimizing ambiguities in error detection and correction. LLMs equipped with advanced chain-of-thought reasoning capabilities are likely to achieve greater accuracy in solving problems in VHDL, as they can leverage knowledge from analogous languages to address novel challenges effectively. Furthermore, incorporating techniques such as ReAct prompting [19], Retrieval-Augmented Generation (RAG) [4], or skeletonization [18] could have enhanced the framework's ability to generate functionally correct Verilog code. These factors suggest that while the framework showed promise, there are several opportunities for further refinement and enhancement.

5 Conclusions

This paper presented an innovative approach to automated hardware design verification through the development of a framework that leverages LLMs to detect and correct both compilation and logical errors in Verilog code. By addressing the critical challenge of hardware design validation, our work demonstrates significant potential for reducing development time and improving code reliability in hardware description languages.

Our primary contribution is the development of an end-to-end framework that successfully identifies and resolves Verilog errors, achieving promising results across a diverse set of test cases. The system not only corrects syntactical errors but also demonstrates capability in addressing functional errors, a traditionally challenging aspect of hardware design verification.

Looking ahead, this research opens several promising avenues for future development. The testing dataset could be expanded to encompass more complex Verilog constructs and edge cases, thereby improving the robustness of the error correction system. In particular, we could explore other benchmark sets of Verilog code to test the system's capabilities, such as VerilogHumanEval [7]. Additionally, exploring the integration of specialized LLMs or fine-tuning existing models could enhance performance specifically for hardware description languages. Another potential enhancement would be incorporating additional debugging tools beyond compiler messages, such as ModelSim, Verdi, and GTKWave. This approach would be particularly novel, as HDLs are unique in that their outputs can be represented visually through waveforms and timing diagrams. Exploring the ability LLMs to analyze and debug HDL designs based on visual representations could provide valuable insights and open new avenues for AI-assisted hardware verification.

The successful implementation of this automated error correction system represents a significant step forward in hardware design verification, potentially transforming how developers approach HDL debugging and validation. As hardware designs become increasingly complex, such specialized tools will be essential for ensuring development efficiency and maintaining high standards of code quality.

References

[1] Anthropic. 2024. The Claude 3 Model Family: Opus, Sonnet, Haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] https://arxiv.org/abs/2107.03374

[3] Pantazis Deligiannis, Akash Lal, Nikita Mehrotra, and Aseem Rastogi. 2023. Fixing Rust Compilation Errors using LLMs. arXiv:2308.05177 [cs.SE] https://arxiv.org/abs/2308.05177

[4] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[5] Github. 2024. The World's most widely adopted AI developer tool. https://github.com/features/copilot

[6] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2024. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Trans. Inf. Syst.* (Nov. 2024). https://doi.org/10.1145/3703155 Just Accepted.

[7] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. 2023. VerilogEval: Evaluating Large Language Models for Verilog Code Generation. arXiv:2309.07544 [cs.LG] https://arxiv.org/abs/2309.07544
[8] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation. arXiv:2402.19173 [cs.SE] https://arxiv.org/abs/2402.19173
[9] OpenAI. [n. d.]. Prompt engineering. https://platform.openai.com/docs/guides/prompt-engineering
[10] "OpenAI". 2022. Introducing ChatGPT. https://openai.com/index/chatgpt
[11] OpenAI. 2024. Survey reveals AI's impact on the developer experience. https://openai.com/index/learning-to-reason-with-llms/
[12] Anshul Raghav. 2024. LLM-based Code Generation for Verilog. https://github.com/anraghav/Personal-Project
[13] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL] https://arxiv.org/abs/2308.12950
[14] Inbal Shani and GitHub Staff. 2023. Survey reveals AI's impact on the developer experience. https://github.blog/news-insights/research/survey-reveals-ais-impact-on-the-developer-experience/
[15] Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. 2022. Benchmarking Large Language Models for Automated Verilog RTL Code Generation. arXiv:2212.11140 [cs.PL] https://arxiv.org/abs/2212.11140
[16] Yun-Da Tsai, Mingjie Liu, and Haoxing Ren. 2024. RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models. arXiv:2311.16543 [cs.AR] https://arxiv.org/abs/2311.16543
[17] Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. 2024. SynCode: LLM Generation with Grammar Augmentation. arXiv:2403.01632 [cs.LG] https://arxiv.org/abs/2403.01632
[18] Xingbo Wu, Nathanaël Cheriere, Cheng Zhang, and Dushyanth Narayanan. 2023. RustGen: An Augmentation Approach for Generating Compilable Rust Code with Large Language Models. https://openreview.net/pdf?id=y9A0vJ5vuM
[19] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629
[20] Jason Yu. 2017. Verilog reg, Verilog wire, SystemVerilog logic. What's the difference? https://www.verilogpro.com/verilog-reg-verilog-wire-systemverilog-logic/#:~:text=Verilog%20rule%20of%20thumb%201,to%20represent%20a%20physical%20connection.

Received 13 January 2025; accepted 18 March 2025

Using AI Modeling to Predict the Impact of Cloud Seeding on Amount of Rainfall and How that Affects the Temperature in Different Regions

Rex Carvalho
rexcarvalho7@gmail.com
Gems Wellington International School
Dubai, United Arab Emirates

Ihita Mandal*
Carnegie Mellon University
Pittsburgh, USA
imandal@andrew.cmu.edu

Abstract

Water is one of the most essential resources on Earth, yet many regions face severe water scarcity due to growing populations, increased land use, and climate change. To address this challenge, we employed cloud seeding as a promising technique for augmenting water supplies. While prior research has primarily focused on increasing precipitation, this study introduces a novel angle by examining how cloud seeding-induced rainfall may also influence regional temperature regulation over time. This dual-purpose investigation is particularly relevant in arid regions like the Middle East, where water scarcity and extreme heat coexist.

Using historical datasets and AI-based modeling, including linear regression and neural networks, we analyzed the relationship between rainfall and temperature across different regions. Results revealed a consistent negative correlation between rainfall and temperature, suggesting that cloud seeding may have cooling effects in addition to boosting precipitation. This study presents early evidence that weather modification could play a role in mitigating long-term climate stress, offering a new perspective on the benefits of cloud seeding.

Keywords

Index Terms—Cloud seeding, Linear regression, Neural networks, Rainfall

ACM Reference Format:
Rex Carvalho and Ihita Mandal. 2025. Using AI Modeling to Predict the Impact of Cloud Seeding on Amount of Rainfall and How that Affects the Temperature in Different Regions . In *Proceedings of International Journal of Secondary Computing and Applications Research (IJSCAR VOL. 2, ISSUE 1).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.5281/zenodo.15149793

1 Introduction

Water scarcity, intensified by growing populations, increased land use, and climate change, has placed significant strain on traditional water sources such as groundwater, rivers, and reservoirs. This has prompted a search for innovative solutions, one of which is cloud seeding, a weather modification technique designed to enhance precipitation. First developed in the mid-20th century, cloud seeding involves introducing substances such as silver iodide or sodium chloride into the atmosphere to stimulate the formation of rain or snow.

Countries worldwide, including those in the Middle East, have adopted cloud seeding to tackle critical water challenges. In addition to increasing rainfall, some suggest that cloud seeding may have potential secondary effects, such as influencing local temperatures. This possible relationship is particularly intriguing for regions like the Middle East, where extreme heat during the summer poses public health and infrastructure challenges. Despite these possibilities, the intersection between rainfall enhancement through cloud seeding and its potential impact on temperature regulation remains poorly understood. Understanding this relationship could reveal how consistent cloud seeding efforts might not only alleviate water shortages but also contribute to long-term temperature moderation—a critical insight for regions facing both climate and resource stress.

2 Related Work

Several studies have examined the effectiveness of cloud seeding in enhancing precipitation. A review paper titled "A Review of Cloud Seeding Experiments to Enhance Precipitation and Some New Prospects" provides a comprehensive overview of the current status of glaciogenic and hygroscopic seeding experiments (Roelof T. Bruintjes) [3–5]. It concludes that the potential of precipitation enhancement through cloud seeding is intricately tied to water resource management. This study underscores the importance of linking weather modification techniques to practical applications, particularly in regions experiencing water shortages.

Another significant contribution is the study "Cloud Seeding Experiments in Australia," which discusses experiments where silver iodide smoke was released from the ground (E. J. Smith) [1, 6]. These experiments showed no immediate results, as the ice-nucleating properties of silver iodide were rapidly diminished under daylight exposure. Despite these challenges, the researchers concluded that cloud seeding with silver iodide has substantial potential for modifying rainfall. However, they also noted the complexities involved, including persistent effects of seeding after operations ceased, which yielded both positive and negative results.

A third paper, "A Cloud-Seeding Experiment in Tasmania," documents an experiment conducted using a target area and three control areas (E. J. Smith, et al.) [2, 7]. This study, which utilized silver iodide smoke released from aircraft, aimed to measure increases

*Thank you for the guidance of Ihita Mandal, mentor from Carnegie Mellon University in the development of this research paper.

IJSCAR VOL. 2, ISSUE 1

in precipitation in a catchment area. The findings indicated that seeding led to increased rainfall in the eastern half of the target area during the autumn season. Notably, this study uses the same dataset that we incorporated, making it a foundational reference for our analysis.

In addition to the studies already reviewed, several foundational papers have contributed to the understanding of cloud seeding dynamics and challenges. For example, Bowen (1966) discussed the persistent effects of seeding on cloud systems, while Grant (1963) addressed the residual atmospheric impacts of silver iodide particles. These works highlight the complexity of atmospheric intervention and provide important context for interpreting results in modern seeding experiments.

While these papers provide valuable insights into cloud seeding's effects on precipitation, they focus almost exclusively on measuring rainfall levels or evaluating the techniques themselves. None of the reviewed studies explicitly investigates the potential correlation between cloud seeding-induced rainfall and temperature changes. This gap is particularly significant for regions such as the Middle East, where extreme heat poses substantial challenges alongside water scarcity.

Our study aims to bridge this gap by examining the relationship between cloud seeding, precipitation levels, and temperature changes. By using historical data and employing AI modeling, we seek to explore the potential for cloud seeding to serve not only as a tool for water augmentation but also as a means of mitigating extreme temperatures. This dual-purpose approach could provide critical insights for regions facing simultaneous water and heat stress. We focused on the research question: How does cloud seeding influence temperature in different regions? We hypothesize that increased rainfall due to cloud seeding leads to observable temperature reductions. Over time, this effect may contribute to regional cooling, particularly in areas where cloud seeding is used frequently and consistently. The primary aim of this research is to develop an AI-based model capable of predicting temperature changes based on cloud seeding-induced rainfall.

3 System

To begin our project, we researched datasets related to cloud seeding in different regions worldwide and selected one most suitable for our goals. We chose a dataset from the study "A Cloud-Seeding Experiment in Tasmania." The paper details seeded and unseeded rainfall in various Tasmanian regions such as NC, SC, NWM, and TE. The dataset includes the period of each recording, the season, whether the rainfall was seeded or unseeded, and the amount of rainfall in millimeters. However, this dataset lacked corresponding temperature data for the same regions during each recording period. To address this limitation, we obtained a dataset containing temperature records from Tasmania during the same years. Using the seasonal data and the number of recordings in each season, we calculated average temperatures for periods with rainfall and no rainfall. We sorted daily temperature data into rainfall and no-rainfall categories and then averaged the values. We repeated this process for all available recordings, producing a new dataset containing the period, season, location, rainfall amount, and average temperatures for seeded and unseeded events. This enabled us to compare temperature differences during rainfall events and non-events. We employed a linear regression model in Python to estimate relationships between rainfall and temperature. We explored combinations such as average temperatures during rainfall versus no rainfall (Y-axis) and rainfall amounts (X-axis). The resulting lines of best fit revealed a consistent negative correlation.

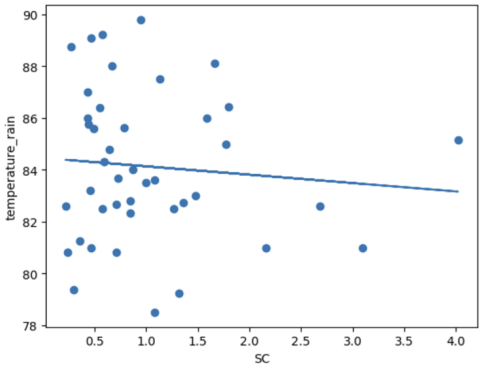


Figure 1: Temperature against amount of rainfall in the South region

On the X-axis of Figure 1, the rainfall in the South region is shown, while the Y-axis represents temperature. As illustrated, this also shows a similar negative correlation.

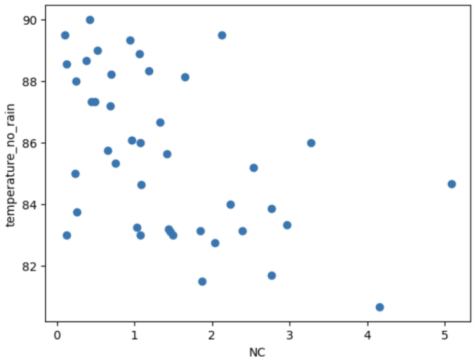


Figure 2: Temperature against amount of rainfall in the North region

Figure 2 presents the correlation between rainfall and temperature during non-rainfall periods. While slightly weaker, the trend remains generally negative.

To further validate the findings, we implemented neural network models. We experimented with various configurations of Dense, Dropout, and Batch Normalization layers using a feedforward neural network architecture. The model was trained using the Adam optimizer with a learning rate of 0.001 and the mean squared error (MSE) loss function. We employed the ReLU activation function for all hidden layers and used early stopping to prevent overfitting. The dataset was split into 80 percent training and 20 percent testing. Despite extensive tuning, the best-performing configuration

included 12 dense layers, one dropout layer (rate = 0.3), and one batch normalization layer at the input. This configuration achieved an MSE of 564.3296 after 15 epochs.

Our models primarily used rainfall amount and regional identifiers as input features. While we considered including additional meteorological variables such as humidity, wind patterns, and atmospheric pressure, we excluded them due to the unavailability of synchronized historical data for those variables in the studied regions. We acknowledge that incorporating a richer feature set may enhance predictive accuracy and recommend this for future research.

4 Evaluation

The results from our analysis using linear regression models consistently demonstrated that increased rainfall correlates with decreased temperatures. We reached this conclusion by analyzing the lines of best fit across different models, where we compared average temperatures during rainfall or no rainfall (Y-axis) with rainfall amounts in millimeters across regions (X-axis). Most models displayed a strong negative correlation, indicating that as rainfall amounts increased, average temperatures decreased. To validate and enhance these findings, we used neural network models, which achieved a mean squared error (MSE) of 564.3296. This MSE value represents the average squared difference between the predicted and actual temperature values, providing a quantitative measure of the model's accuracy. A lower MSE indicates a smaller average error and greater predictive reliability. In this study, the MSE of 564.3296 suggests that the model was effective in capturing the relationship between rainfall amounts and temperature changes, albeit with some level of variance due to the inherent complexity of the data. This level of accuracy reinforces the hypothesis of a negative correlation between rainfall and temperature while also highlighting the utility of neural networks for modeling complex environmental phenomena.

Our experimentation revealed that simpler models or more complex configurations performed less effectively. Minor variability in MSE (between 560 and 570) was observed across multiple runs due to the stochastic nature of training, but the overall trend remained stable. Overall, our results provide strong evidence that cloud seeding-induced rainfall has the potential to reduce temperatures. These findings underscore the dual benefits of cloud seeding in addressing water scarcity and mitigating extreme heat in targeted regions.

The findings of this study reveal a consistent negative correlation between increased rainfall and reduced temperatures, highlighting the dual benefits of cloud seeding in addressing water scarcity and mitigating extreme heat. This aligns with the hypothesis that enhanced rainfall from cloud seeding contributes to cooling effects, making it a potentially transformative tool for regions struggling with both water shortages and high temperatures.

A significant limitation of this study lies in the dataset selection. Our analysis is based solely on data from Tasmania, which raises questions about the generalizability of our findings to other climates—particularly arid regions like the Middle East, which are frequently referenced in this paper. While Tasmania provided a reliable and accessible dataset for initial modeling, it differs substantially in geography, humidity, and seasonal patterns from Middle Eastern regions. A comparative discussion of Tasmania's climate with that of the Middle East could help clarify the potential for generalizing these findings. Future studies should integrate datasets from multiple geographical locations to validate the universality of these results across diverse climatic zones. Compared to previous studies that primarily evaluated the efficiency of cloud seeding in increasing precipitation, this research extends the scope by exploring its secondary effects on temperature—although we acknowledge that this analysis is correlational in nature. This novel perspective provides a more comprehensive understanding of the broader implications of cloud seeding and underscores its potential to address interrelated environmental challenges.

It is important to note that while our study identifies a negative correlation between rainfall and temperature, this does not establish a causal relationship. The analysis does not account for potential confounding variables such as seasonal variation, natural climate patterns, or urban heat island effects, all of which could independently influence temperature trends. Although our study focused on historical trends rather than controlled experimental conditions, future research should incorporate causal inference methods or real-time controlled experiments to isolate the effects of cloud seeding. In the current work, such variables were excluded primarily due to data constraints and the exploratory nature of the study, but we recognize the need for greater control and rigor in subsequent analyses.

Another limitation lies in the use of seasonal average temperature values rather than real-time data. Averaging across long periods may obscure short-term fluctuations and delay effects of rainfall events, potentially diminishing the model's sensitivity to direct seeding outcomes. While this approach was necessary due to data availability, it may limit the precision of temperature predictions. Future studies should incorporate high-resolution temporal data to better capture localized weather effects.

5 Conclusions

This study highlights the dual benefits of cloud seeding, demonstrating its potential to address water scarcity and mitigate extreme temperatures. By analyzing datasets from Tasmania and employing both linear regression and neural network models, we identified a consistent negative correlation between rainfall and temperature. The neural network models, which achieved a mean squared error of 564.3296, further support the reliability of these findings despite certain data limitations.

Our research contributes to a broader understanding of cloud seeding's secondary effects, emphasizing its value not only as a tool for enhancing precipitation but also as a means of addressing heat stress in vulnerable regions. Practically, this research could inform policies and strategies for climate adaptation, particularly in arid and semi-arid environments. For example, countries facing extreme heat and water scarcity might prioritize cloud seeding as part of integrated water resource management programs.

Further research should address the limitations of this study by analyzing datasets from diverse geographic regions with varying climates to evaluate the universality of these findings. Additionally,

advancements in modeling techniques—such as the integration of real-time temperature and rainfall data—could refine predictions and provide deeper insight into the complex dynamics of weather modification.

By demonstrating a link between cloud seeding and temperature change, this research opens a path toward exploring whether sustained use of weather modification could help cool climate-stressed regions over time. This long-term perspective adds a new dimension to existing weather modification strategies, merging water resource management with potential climate mitigation efforts.

References

[1] FD Bethwaite, EJ Smith, JA Warburton, and KJ Heffernan. 1966. Effects of seeding isolated cumulus clouds with silver iodide. *Journal of Applied Meteorology and Climatology* 5, 4 (1966), 513–520.

[2] EG Bowen. 1966. The effect of persistence in cloud seeding experiments. *Journal of Applied Meteorology and Climatology* 5, 2 (1966), 156–159.

[3] Roelof T. Bruintjes. 1999. A Review of Cloud Seeding Experiments to Enhance Precipitation and Some New Prospects. (1999), 805–820. https://journals.ametsoc.org/view/journals/bams/80/5/1520-0477_1999_080_0805_arocse_2_0_co_2.xml

[4] Lewis O Grant et al. 1963. Indications of residual effects from silver iodide released into the atmosphere. (1963).

[5] EJ Smith. 1949. Experiments in seeding cumuliform cloud layers with dry ice. *Australian Journal of Chemistry* 2, 1 (1949), 78–91.

[6] EJ Smith. 1967. Cloud seeding experiments in Australia. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 5: Weather Modification*, Vol. 5. University of California Press, 161–177.

[7] EJ Smith, LG Veitch, DE Shaw, and AJ Miller. 1979. A cloud-seeding experiment in Tasmania. *Journal of Applied Meteorology (1962-1982)* (1979), 804–815. https://journals.ametsoc.org/view/journals/apme/18/6/1520-0450_1979_018_0804_acseit_2_0_co_2.xml

Received 19 March 2025; accepted 06 April 2025