A close-up of a document

AI-generated content may be incorrect.

In my current role as a Computer Scientist in the Reliability and Maintainability division at NAWCWD, I support U.S. Navy systems through a combination of statistical reliability analysis and model-based systems engineering. I’ve conducted Weibull modeling, fault tree analysis (FTA), and failure modes and effects analysis (FMEA) to assess and improve system reliability and readiness. To streamline analysis and support predictive maintenance, I’ve developed Python and MATLAB scripts that automate data processing, test planning, and modeling workflows. These tools have been instrumental in identifying failure trends and optimizing maintenance strategies. Additionally, I’ve worked with SysML and other MBSE tools to document simulation workflows and ensure traceability between system architecture, reliability assessments, and operational requirements—bridging the gap between analytical models and system-level design.

In support of Navy missile systems, I developed Python and MATLAB scripts to analyze flight test and diagnostic data for reliability and predictive maintenance. Using libraries like NumPy, Pandas, and MATLAB’s Predictive Maintenance Toolbox, I implemented Weibull modeling, survival analysis, and RUL estimation to assess component health and forecast failures. These tools enabled condition-based monitoring of critical missile subsystems, helping to reduce unplanned downtime and improve test planning and system readiness.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

A close-up of a text

AI-generated content may be incorrect.

Because early detection is so important for lung cancer patient outcomes, a lot of work has been done in developing models that can detect lung cancer early. Some are clinical models, that is statistical models that consider risk factors to calculate the likelihood of a patient developing cancer. Others are imaging models, that is models that use medical scans like ct scans to detect diseases. Although both can be effective, by focusing on one type of data, valuable information can be overlooked. For this reason, in this project, I explored ways to combine the two. To do so, I looked at two existing models, the PLCO (Prostate, Lung, Colorectal, Ovarian) cancer screening trial and sybil, a deep-learning based imaging model developed by MIT and explored ways combine both imaging and clinical data using these two models. More specifically, I modified sybil to act as a feature extracter. I removed the final prediction layer to so I can use it to extract meaningful imaging features from CT scans and combined these extracted features with the clinical features used in the PLCO model. I took two main approaches, early fusion and late fusion. In early fusion, I combined the imaging features and clinical features and trained models on this combined data to predict the likely hood of a patient developing lung cancer. In late fusion, I trained models separately on the two types of data and combined the outputs to make the final prediction. Using AUROC cure and AUPRC as metrics, both approaches saw an improvement in performance, though the early fusion method saw a larger increase in performance.

Because early detection is critical to improving lung cancer outcomes, many models have been developed to detect it at earlier stages. Some are **clinical models**, which use statistical analysis of patient risk factors, while others are **imaging models**, which use CT scans to identify cancerous patterns. Each approach is useful but limited when used in isolation, potentially missing key information.

In this project, I explored how combining the two could improve predictive performance. I focused on two existing models: the **PLCO clinical model**, based on the Prostate, Lung, Colorectal, and Ovarian cancer screening trial, and **Sybil**, a deep learning-based imaging model developed by MIT. I modified Sybil by removing its final prediction layer, allowing it to serve as a **feature extractor** for meaningful imaging features from CT scans. I then combined those features with the clinical features used in the PLCO model.

I tested two fusion strategies: **early fusion**, where imaging and clinical features were combined before training, and **late fusion**, where separate models were trained for each modality and their outputs were merged. Using AUROC and AUPRC as performance metrics, both fusion approaches outperformed the individual models, with early fusion showing the greatest improvement. This work demonstrates the potential of **multimodal learning** to enhance early lung cancer risk prediction.

* Late fusion:
  + weighted average
  + average the two outputs
  + stacking approach: train model to combine the two outputs
* G
* Fgsd

**Questions:**

How did you handle class imbalance?

To handle class imbalance in my lung cancer prediction project, I explored both data-level and algorithm-level strategies. On the data side, I applied SMOTE to oversample the minority class. On the algorithmic side, I adjusted class weights in my logistic regression and neural models to give more importance to positive cases. I also used AUROC and AUPRC to evaluate model performance more fairly and performed threshold tuning to improve sensitivity for rare outcomes. These steps helped ensure that the model didn't simply default to predicting the majority class

What is SMOTE:

SMOTE stands for Synthetic Minority Over-sampling Technique. It’s a method used to address class imbalance by creating new synthetic examples of the minority class instead of simply duplicating them. It works by finding a real example from the minority class and selecting one of its nearest neighbors — then generating a new data point somewhere along the line connecting them. This helps increase the diversity of minority class samples, allowing the model to learn better decision boundaries. I used SMOTE in my project to balance lung cancer cases in the dataset, which improved recall and made the model more sensitive to rare but critical outcomes

What are some ways to handle class imbalance? What are strengths and weaknesses of each technique?

There are several ways to handle class imbalance, and the best choice often depends on the dataset and model.

One approach is **resampling**. You can **oversample the minority class** using techniques like SMOTE, which generates synthetic samples based on existing ones. This can improve model sensitivity to rare classes but may lead to overfitting if synthetic samples aren’t diverse enough. Alternatively, **undersampling the majority class** reduces imbalance by removing some majority samples, which is faster but risks losing valuable data.

Another strategy is to **adjust class weights** during training. Many models like logistic regression, random forests, and neural networks allow you to increase the penalty for misclassifying the minority class. This is efficient and avoids changing the data, but may require careful tuning to avoid harming precision.

A third option is **threshold tuning** — instead of using a default 0.5 threshold for classification, you can shift the threshold to favor recall or precision, depending on your goal.

In my experience, combining class weighting with SMOTE and evaluating performance using metrics like AUROC and AUPRC gives the best balance, especially in high-stakes domains like healthcare

How to combat overfitting?

“Overfitting happens when a model learns patterns that are too specific to the training data, including noise, and doesn’t generalize well to new data. I use several techniques to combat this:

* **Cross-validation** is my first line of defense. I typically use k-fold cross-validation to ensure the model performs consistently across different subsets of the data.
* **Regularization**, such as L1 (Lasso) or L2 (Ridge), helps penalize overly complex models by shrinking weights. This is especially useful in linear models and neural networks.
* I also **simplify the model** architecture when needed — using fewer parameters or limiting tree depth in decision trees, for instance.
* **Early stopping** is helpful in training neural networks; it monitors validation loss and halts training once performance starts to degrade.
* **Dropout layers** in neural networks randomly deactivate neurons during training, which helps prevent the model from relying too heavily on any one part of the network.
* Finally, I ensure I have enough training data, and if not, I might use **data augmentation** in image tasks or **feature engineering** to enrich the input.

For example, in my lung cancer project, I used regularization, early stopping, and cross-validation to ensure that the combined imaging-clinical model generalized well, especially given the class imbalance.

How do you know the features extracted from Sybil are meaningful or optimal?

 Acknowledge that this is a limitation due to the black-box nature of deep CNNs.

 Mention that you evaluated performance improvements empirically and saw that adding Sybil features increased AUC and other metrics.

 Suggest future work using **feature attribution tools** like **Grad-CAM** to visualize what the model is focusing on.

**Grad-CAM: a technique used to visualize and understand the predictions of deep learning models, particularly Convolutional Neural Networks (CNNs). It generates heatmaps that highlight the regions in an input image that are most important for the model's decision.**

Did you consider fine-tuning the Sybil model instead of using it as-is?

 Explain that fine-tuning a model as large as Sybil requires **significant GPU resources and labeled data**.

 Your dataset (PLCO) had limited size and labeling consistency.

 Using Sybil as a frozen encoder was a **resource-aware tradeoff** that still allowed you to leverage its learned representations.

How did you ensure the imaging and clinical data were synchronized properly?

 Note the use of **patient IDs, timestamps**, and **cohort filtering** to ensure matched pairs.

 Mention any preprocessing steps to remove mismatches or incomplete cases.

 Acknowledge this as a potential **source of bias** and describe how you mitigated it.

Did you assess the interpretability of the final multimodal model?

 Say you prioritized **performance benchmarking first**, but are aware of the need for explainability.

 Suggest future use of **SHAP, LIME, or Grad-CAM** to inspect feature importance for both image and clinical features.

 If applicable, mention that **tree-based models** (e.g., Random Forest, XGBoost) provide inherent feature importance.

Were there any limitations in the PLCO dataset that affected your findings?

 Note that PLCO is older and U.S.-based, which may limit demographic generalizability.

 You can mention that part of your motivation for using multimodal data was to **boost robustness** in real-world settings.

 Suggest validating the approach on newer or external datasets (e.g., NLST, LIDC-IDRI).

Why not use a more advanced fusion strategy (e.g., attention, transformers)?

 Acknowledge that **early fusion and shallow models** were chosen for simplicity, interpretability, and stability in training.

 Point out that the goal was to **establish a baseline and prove the value** of using Sybil features before escalating complexity.

 Mention this as a direction for future work.

### **Intermediate Fusion (Cross-modal attention or gating)**

* Use **attention layers** to weigh one modality based on the importance of the other.
* Example: Use an attention mechanism to weigh clinical inputs based on spatial cues from image features.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

A close up of a document

AI-generated content may be incorrect.

CalEnviorScreen is a data collection effort that publishes reports detailing census tract level socieoeconomic and environmental data on the area of Los Angeles. We worked to to identify the most vulnerable communities in Los Angeles based on pollution exposure, socioeconomic stressors, and health outcomes, and to analyze how these vulnerabilities relate to COVID-19 case and death rates, asthma, low birth weight, and cardiovascular disease.

In this project, I collaborated with a team to identify and analyze health vulnerabilities across Los Angeles using environmental pollution data and socioeconomic indicators. Using CalEnviroScreen data and census tract-level COVID-19 case and death rates, we applied statistical analyses and machine learning models like Lasso, Random Forest, and XGBoost to uncover strong correlations between pollution exposure, low education, poverty, and adverse health outcomes such as asthma, heart disease, and low birth weight. Our findings highlighted that communities near the 105 and 110 freeways faced the greatest risks, and that factors like linguistic isolation and education level were among the strongest predictors of COVID-19 impact. This work underscored the value of data-driven approaches to inform equitable public health policy and targeted interventions.

**Questions:**

What was the main objective of your project?

The goal was to identify the most vulnerable communities in Los Angeles by analyzing how pollution, socioeconomic stressors, and preexisting health conditions correlated with adverse outcomes like COVID-19 cases and deaths, asthma, and cardiovascular disease. We wanted to use data-driven methods to uncover systemic health disparities and inform targeted interventions.

What types of data did you use, and how did you process it?

We used CalEnviroScreen data, which includes 21 indicators of environmental and population vulnerability, along with COVID-19 case and death rates aggregated at the census tract level. We cleaned and merged these datasets using geographic identifiers and standardized the features by converting them into raw values, percentiles, and z-scores for use across different modeling techniques.

What machine learning models did you use and why?

We used a mix of regularized linear models like Lasso and Ridge for interpretability, as well as ensemble models like Random Forest and XGBoost for their ability to handle non-linear relationships and feature interactions. We compared their performance using cross-validation to identify which models were most effective for predicting different health outcomes.

What did your results reveal?

Our analysis showed that areas with high pollution burden, low education levels, and high linguistic isolation experienced significantly worse health outcomes. For instance, COVID-19 case and death rates were strongly associated with poverty, asthma prevalence, and education. These findings suggest that addressing environmental and social inequities could improve public health resilience.

What challenges did you face and how did you overcome them?

A major challenge was aligning different datasets by geographic unit—some were in census tracts, others by ZIP code or county. We used spatial joins and shapefiles to standardize everything to the census tract level. Another challenge was dealing with missing or unreliable data, which we addressed through imputation and filtering.

How would you improve this project in the future?

I’d expand it to include longitudinal data to monitor trends over time, incorporate more granular health outcome data, and explore causal inference methods. I’d also consider integrating satellite data and more advanced geospatial techniques to refine the environmental exposure models.

**What statistical techniques did you use before modeling?**  
We performed Spearman and Kendall correlation analyses to evaluate associations between vulnerability indicators and health outcomes. We also used bootstrapped confidence intervals to assess the robustness of our estimates.

Did you do any hyperparameter tuning?

Yes, I used grid search with k-fold cross-validation to tune parameters like alpha for regularized models, and max depth, learning rate, and tree count for tree-based models.

How did you evaluate model performance?

We used R², mean absolute error, and root mean squared error as primary metrics, and compared them across models and data variants—raw values, percentiles, and z-scores—to determine the most effective modeling approach for each health outcome.

What were the most surprising insights from your analysis?

The strong correlation between linguistic isolation and COVID-19 death rates was unexpected and underscored the role of language barriers in healthcare access. It showed how non-medical factors can greatly influence health outcomes.

How did you ensure the findings were understandable by non-technical audiences?

We created geographic heatmaps, bar plots, and interpreted statistical findings in plain language. We also contextualized results with real-world implications for policy, focusing on accessibility and clarity.

“The primary purpose of the predictive models was to estimate and identify which communities in Los Angeles were at greatest risk for adverse health outcomes—such as COVID-19 cases and deaths, asthma, cardiovascular disease, and low birth weight—based on environmental and socioeconomic stressors. By leveraging machine learning algorithms like Lasso, Random Forest, and XGBoost, we were able to assess which combinations of factors—like air pollution, poverty, and linguistic isolation—were most predictive of those outcomes. This allowed us not only to uncover underlying relationships but also to build models that could be used by policymakers or public health officials to forecast health burdens in other neighborhoods, prioritize interventions, and guide resource allocation more effectively.”

**Spearman’s Rank Correlation Coefficient (ρ or “rho”)**

**What it does**:  
Spearman’s correlation assesses how well the relationship between two variables can be described by a monotonic function. It is based on the **ranked values** rather than the raw data.

**How it works**:

* Each variable is ranked independently.
* The Pearson correlation formula is then applied to the ranks.
* Perfect correlation = +1, perfect inverse correlation = –1, no correlation = 0.

**Best for**:

* When the relationship is not linear but still monotonic.
* When the data contains outliers or isn’t normally distributed.

**🔁 Kendall’s Tau (τ)**

**What it does**:  
Kendall’s Tau measures the strength of association between two variables by comparing **the number of concordant and discordant pairs** in the ranked data.

**How it works**:

* A pair is **concordant** if the ranks for both elements agree (both go up or down together).
* A pair is **discordant** if one goes up and the other goes down.
* Tau is calculated from the difference between concordant and discordant pairs.

**Best for**:

* Smaller datasets or when tied ranks are common.
* Giving a more conservative and reliable estimate of correlation than Spearman's in some contexts.

What is *Non-Parametric*?

A **non-parametric method** is a type of statistical technique that does **not assume a specific distribution** (like normal/Gaussian) for the data. In contrast, **parametric methods** rely on assumptions about the underlying distribution (e.g., the data must be normally distributed or follow a certain form).

As part of a collaboration LA’s Data Team, I helped analyze crime records from 2010 to mid-2022 to uncover trends in serious (Part 1) crimes across Los Angeles neighborhoods. The dataset was geocoded and aligned with public health neighborhood designations, enabling spatial analysis at the community level. We observed fluctuations in crime trends over time, with notable drops during the COVID-19 pandemic and spikes in 2016, 2018, and 2021. Demographic analysis showed differences in crime victimization by age, gender, and ethnicity, while regression-based trend analysis highlighted neighborhoods with the most significant increases and decreases in crime over the 12-year period.

To enhance the insights, a logical extension of our analysis would be the application of clustering algorithms—such as K-Means or DBSCAN—to group neighborhoods with similar crime profiles. This would help city planners and law enforcement identify regions with shared characteristics for targeted interventions. Additionally, I would apply ARIMA models to perform time series forecasting of future crime rates in each neighborhood. This would allow proactive resource allocation by forecasting crime surges based on historical patterns. Integrating these techniques could transform static historical analysis into a forward-looking, data-driven crime prevention strategy.

Demographic analysis included bar charts and regression plots showing trends by age, gender, and ethnicity. We found males were 1.5 times more likely to be victims overall, with victimization peaking in the 23–25 age group. Homicides were explored separately, with visualizations revealing a significant spike in 2020–2021 and disproportionate impacts on Hispanic and Black communities.

**Questions:**

What was the goal of your crime trends project?

The goal was to analyze long-term crime patterns in Los Angeles from 2010 to 2021 using open data provided by the Mayor’s Office. We aimed to identify trends by neighborhood, track demographic changes among victims, and examine shifts in the frequency of serious (Part 1) crimes over time. Our findings were used to support citywide decision-making and public safety policy development.

What tools or libraries did you use for your analysis?

We primarily used Python, leveraging libraries like Pandas for data manipulation, Matplotlib and Seaborn for visualization, and Statsmodels and Scikit-learn for regression and statistical analysis. For mapping and spatial analysis, we used geospatial tools such as ArcGIS and geopandas to visualize neighborhood-level crime trends.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

A close-up of a black text

AI-generated content may be incorrect.

This project aimed to accurately classify hand gestures representing letters in American Sign Language (ASL) using both traditional machine learning models and deep learning approaches. The team began by reviewing a Kaggle dataset of 87,000 images representing 29 ASL signs. To simplify the task and address computational limits, they focused only on the 26 alphabet letters and resized each image from 200x200 pixels to 50x50, sampling 1,000 images per letter to create a balanced dataset of 26,000 samples.

For traditional models—including SVM, Decision Tree, Random Forest, KNN, XGBoost, and AdaBoost—the team used Principal Component Analysis (PCA) to reduce dimensionality from 7,500 to 80 features, preserving 95% of the data variance. These models underwent hyperparameter tuning via GridSearch. Separately, a Convolutional Neural Network (CNN) was trained directly on the resized images, with architecture and parameters iteratively optimized.

The best-performing model was the deep CNN, achieving 96.38% accuracy, 96.65% precision, 96.38% recall, and an AUC of 99.96. Among traditional models, the SVM performed the best with 93.23% accuracy. AdaBoost significantly underperformed, reaching only 17.72% accuracy even after tuning. Evaluation metrics—including precision, recall, and AUC—corroborated that CNN was the most robust model overall.

In conclusion, the project demonstrated that deep learning, particularly CNNs, is highly effective for image-based ASL classification. The authors suggested that future work could benefit from using the full dataset, retaining more variance during PCA, and running more comprehensive hyperparameter searches for further performance gains.

**Questions:**

What was the goal of your ASL classification project?

The goal was to build a machine learning system that could accurately classify static hand signs from American Sign Language into their corresponding alphabet letters. We worked with a large dataset of labeled images and implemented both traditional machine learning and deep learning models to compare performance and accuracy. Ultimately, we wanted to understand how well different approaches could handle image-based gesture recognition.

How did you prepare and preprocess the data?

We started with an 87,000-image dataset from Kaggle, which included images for 29 ASL signs. We narrowed it down to just the 26 letters of the alphabet and sampled 1,000 images per class to ensure balance. Each image was resized from 200x200 to 50x50 pixels to reduce memory usage. For traditional ML models, we flattened the image matrices and applied PCA to reduce dimensionality from 7,500 to 80 features while preserving 95% of the variance.

Why did you use PCA?

PCA helped us address the curse of dimensionality by compressing image data while retaining most of its variance. With 7,500 input features per image, training models would have been slow and prone to overfitting. Reducing this to 80 principal components allowed for much faster and more accurate model training, particularly with algorithms like SVM and KNN.

What is the curse of dimensionality?

The curse of dimensionality refers to the various challenges and inefficiencies that arise when analyzing and organizing data in high-dimensional spaces. As the number of dimensions (features) in a dataset increases, the volume of the space grows exponentially, leading to data becoming increasingly sparse. This sparsity makes it difficult for machine learning algorithms to generalize accurately, as there's not enough data to fill the space effectively.

Here's a breakdown of the key issues:

[Data Sparsity:](https://www.google.com/search?sca_esv=2dc8bb8b89129ca2&q=Data+Sparsity&sa=X&ved=2ahUKEwjwjJaWn9GOAxUmk68BHWKKLicQxccNegQIOhAD&mstk=AUtExfCBekObscqATFILxS9Ho6oeqQ971lbKqTFbTRD_Z_DfUNW2EDJAHXPJJOz0GdhG6wba3Bx5nYS15AUbvaPA7HgfJXf1iHdgw8MHTyLyWAztEtjFdwWWXBh79kL5RPFVpYI&csui=3" \t "_blank)

[.](https://www.google.com/search?sca_esv=2dc8bb8b89129ca2&q=Data+Sparsity&sa=X&ved=2ahUKEwjwjJaWn9GOAxUmk68BHWKKLicQxccNegQIOhAD&mstk=AUtExfCBekObscqATFILxS9Ho6oeqQ971lbKqTFbTRD_Z_DfUNW2EDJAHXPJJOz0GdhG6wba3Bx5nYS15AUbvaPA7HgfJXf1iHdgw8MHTyLyWAztEtjFdwWWXBh79kL5RPFVpYI&csui=3" \t "_blank)

In high-dimensional spaces, data points become increasingly isolated from each other, making it harder to find meaningful patterns or relationships.

[Increased Computational Complexity:](https://www.google.com/search?sca_esv=2dc8bb8b89129ca2&q=Increased+Computational+Complexity&sa=X&ved=2ahUKEwjwjJaWn9GOAxUmk68BHWKKLicQxccNegUIiAEQAw&mstk=AUtExfCBekObscqATFILxS9Ho6oeqQ971lbKqTFbTRD_Z_DfUNW2EDJAHXPJJOz0GdhG6wba3Bx5nYS15AUbvaPA7HgfJXf1iHdgw8MHTyLyWAztEtjFdwWWXBh79kL5RPFVpYI&csui=3" \t "_blank)

[.](https://www.google.com/search?sca_esv=2dc8bb8b89129ca2&q=Increased+Computational+Complexity&sa=X&ved=2ahUKEwjwjJaWn9GOAxUmk68BHWKKLicQxccNegUIiAEQAw&mstk=AUtExfCBekObscqATFILxS9Ho6oeqQ971lbKqTFbTRD_Z_DfUNW2EDJAHXPJJOz0GdhG6wba3Bx5nYS15AUbvaPA7HgfJXf1iHdgw8MHTyLyWAztEtjFdwWWXBh79kL5RPFVpYI&csui=3" \t "_blank)

More dimensions mean more computations are required to analyze the data, leading to slower processing times and potentially requiring more powerful hardware.

[Overfitting:](https://www.google.com/search?sca_esv=2dc8bb8b89129ca2&q=Overfitting&sa=X&ved=2ahUKEwjwjJaWn9GOAxUmk68BHWKKLicQxccNegUIggEQAw&mstk=AUtExfCBekObscqATFILxS9Ho6oeqQ971lbKqTFbTRD_Z_DfUNW2EDJAHXPJJOz0GdhG6wba3Bx5nYS15AUbvaPA7HgfJXf1iHdgw8MHTyLyWAztEtjFdwWWXBh79kL5RPFVpYI&csui=3" \t "_blank)

With high dimensionality, models can become overly sensitive to noise in the data, leading to overfitting where the model performs well on training data but poorly on unseen data.

[Difficulty in Visualization:](https://www.google.com/search?sca_esv=2dc8bb8b89129ca2&q=Difficulty+in+Visualization&sa=X&ved=2ahUKEwjwjJaWn9GOAxUmk68BHWKKLicQxccNegUIhwEQAw&mstk=AUtExfCBekObscqATFILxS9Ho6oeqQ971lbKqTFbTRD_Z_DfUNW2EDJAHXPJJOz0GdhG6wba3Bx5nYS15AUbvaPA7HgfJXf1iHdgw8MHTyLyWAztEtjFdwWWXBh79kL5RPFVpYI&csui=3" \t "_blank)

[.](https://www.google.com/search?sca_esv=2dc8bb8b89129ca2&q=Difficulty+in+Visualization&sa=X&ved=2ahUKEwjwjJaWn9GOAxUmk68BHWKKLicQxccNegUIhwEQAw&mstk=AUtExfCBekObscqATFILxS9Ho6oeqQ971lbKqTFbTRD_Z_DfUNW2EDJAHXPJJOz0GdhG6wba3Bx5nYS15AUbvaPA7HgfJXf1iHdgw8MHTyLyWAztEtjFdwWWXBh79kL5RPFVpYI&csui=3" \t "_blank)

Visualizing and understanding data in high dimensions becomes challenging, making it harder to gain insights.

Essentially, the curse of dimensionality means that while more features might seem helpful initially, they can quickly overwhelm algorithms and lead to poor performance if not handled carefully. This is why techniques like [dimensionality reduction](https://www.google.com/search?sca_esv=2dc8bb8b89129ca2&q=dimensionality+reduction&sa=X&ved=2ahUKEwjwjJaWn9GOAxUmk68BHWKKLicQxccNegUIgwEQAQ&mstk=AUtExfCBekObscqATFILxS9Ho6oeqQ971lbKqTFbTRD_Z_DfUNW2EDJAHXPJJOz0GdhG6wba3Bx5nYS15AUbvaPA7HgfJXf1iHdgw8MHTyLyWAztEtjFdwWWXBh79kL5RPFVpYI&csui=3) are crucial in machine learning when dealing with high-dimensional data.

What models did you use, and which performed best?

We used several traditional classifiers including SVM, Decision Tree, Random Forest, KNN, XGBoost, and AdaBoost. We also built a CNN for deep learning. The CNN significantly outperformed the traditional models, achieving 96.38% accuracy, while SVM was the best among the classical methods with 93.23% accuracy. AdaBoost performed poorly even after tuning, only reaching 17.72% accuracy.

What was the architecture of your CNN?

We used a sequential CNN with convolutional layers followed by ReLU activations, max-pooling, and dropout layers to reduce overfitting. After flattening, we passed the output through fully connected layers and a softmax classifier for the final prediction. We used Adam optimizer and categorical cross-entropy loss, and we trained for multiple epochs with tuned batch sizes.

What challenges did you face and how did you address them?

One of the main challenges was working within memory and computational limits when handling image data. To overcome this, we resized images and used PCA for traditional models. Another challenge was overfitting in the CNN, which we mitigated by adding dropout layers, tuning batch size and learning rates, and using early stopping during training.

**How did you evaluate model performance?**

We used accuracy, precision, recall, and AUC as our evaluation metrics. CNN achieved the highest performance across all metrics, with an AUC of 99.96. We also used confusion matrices to examine which letters were commonly misclassified and considered using data augmentation to address misclassification in future work.

**What would you improve or do differently next time?**

We would use the full dataset instead of sampling, which would likely improve performance, especially for underrepresented classes. Additionally, we’d explore data augmentation techniques, more sophisticated CNN architectures like ResNet, and even apply transfer learning. We would also conduct a more exhaustive hyperparameter search using Bayesian optimization or Random Search.

What was your biggest takeaway from this project?

My biggest takeaway was how critical model selection and data preprocessing are for image classification tasks. While traditional ML models offer value, deep learning methods like CNNs can capture spatial hierarchies that flat vector-based models cannot. It also underscored the importance of balancing performance with interpretability and computational cost.

What is Bayesian optimization and Random Search?

Both Bayesian Optimization and Random Search are techniques used for hyperparameter tuning — the process of finding the best settings (like learning rate, number of layers, batch size, etc.) to improve a model’s performance.

**Random Search (Simple & Surprisingly Effective)**

**What it is:**

* Randomly samples combinations of hyperparameters from defined ranges.
* Runs the model with each combination and evaluates performance.
* Chooses the combination that works best.

## Bayesian Optimization (Smarter, More Efficient)

### What it is:

* Uses a **probabilistic model** (usually Gaussian Processes) to **learn from past tests** and **predict which hyperparameter combos will work best next**.
* Chooses new points to test based on this prediction — it **balances exploring new areas and exploiting promising ones**.

### Workflow:

1. Try a few initial combinations.
2. Build a model that predicts performance for other combinations.
3. Pick the next best combo to try.
4. Repeat until the budget (e.g., number of trials) is used.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

A black text on a white background

AI-generated content may be incorrect.

In this project, the team explored advanced generative models to synthesize images in the style of **Claude Monet**, using a dataset of Monet paintings. The primary objective was to compare different generative approaches — namely **Deep Convolutional GANs (DCGAN)**, **Wasserstein GANs with Gradient Penalty (WGAN-GP)**, and **Diffusion Models** — to evaluate their effectiveness in producing high-quality, stylized images.

The initial part of the project involved training **GAN-based models**. DCGAN served as a foundational architecture, while WGAN-GP was employed to address typical GAN issues such as **training instability** and **mode collapse**. The models were implemented in **PyTorch** with **CUDA acceleration** to enable efficient GPU-based training.

The second phase of the project transitioned to **diffusion models**, a newer class of generative models that progressively **denoise random noise** to generate realistic images. The team implemented a basic diffusion model from scratch, allowing them to experiment with **forward and reverse diffusion processes**, and assess the model’s ability to synthesize Monet-style artwork. These models generally produced smoother and more globally coherent images than GANs, although training was more computationally intensive.

To evaluate model performance, the team used:

* **Loss curves** to track training stability,
* **Generated image samples** over training epochs,
* And **side-by-side visual comparisons** to assess texture, brushstroke quality, and stylistic coherence.

Ultimately, the project concluded that while GANs (especially WGAN-GP) were faster to train and capable of producing sharp images, **diffusion models** offered superior stylistic consistency and fewer artifacts, making them especially promising for high-fidelity artistic generation.

Why did you use multiple generative models instead of focusing on just one?

Each model type has different strengths. DCGANs are easy to implement and fast to train, WGAN-GP offers better training stability and convergence, and diffusion models, though slower, tend to produce more globally coherent and artifact-free images. Comparing them helped us understand the trade-offs and choose the most appropriate tool depending on the use case.

What challenges did you face with GANs, and how did you address them?

GANs often suffer from instability and mode collapse. To address this, we used WGAN-GP, which replaces the standard loss with a Wasserstein loss and enforces a Lipschitz constraint via gradient penalty. This improved training stability and diversity of generated samples.

How did you evaluate the quality of generated images?

We used both qualitative and quantitative evaluations. Visually, we assessed the texture, color composition, and brushstroke realism. Quantitatively, we tracked generator and discriminator losses, and monitored training curves for divergence or oscillation. We also did side-by-side comparisons of samples across epochs.

**Q6: What is mode collapse and how did you observe it in your project?**

**A:** Mode collapse is a common issue in GANs where the generator produces limited variations of images, ignoring parts of the data distribution. We observed this in the DCGAN model where different latent vectors often resulted in nearly identical Monet-style images. To mitigate this, we used WGAN-GP, which helped improve diversity by stabilizing training with the Wasserstein loss and gradient penalty.

**Q7: Can you explain how gradient penalty works and why it's important?**

**A:** Gradient penalty is used in WGAN-GP to enforce the Lipschitz constraint, which is essential for the Wasserstein loss to work properly. It penalizes the model when the gradient norm deviates from 1, helping to prevent training instability. In our case, it made the training process smoother and helped avoid exploding or vanishing gradients.

**Q8: Why did you choose Monet paintings, and what challenges did this dataset pose?**

**A:** We chose Monet because his artwork features distinct color palettes and textures, which make for a visually interesting and challenging style to replicate. However, the dataset was relatively small and stylistically diverse within the Monet genre, which increased the risk of overfitting. We addressed this by using data augmentation techniques such as flipping, cropping, and scaling.

**Q9: How did CUDA accelerate your workflow?**

**A:** We used CUDA to enable GPU acceleration through PyTorch, significantly speeding up the training of deep neural networks. For example, training epochs that would take hours on CPU could be completed in minutes on GPU, allowing us to iterate on model architecture and hyperparameters more efficiently.

**Q10: What are the main differences in training GANs vs. diffusion models?**

**A:** GANs involve a min-max game between a generator and a discriminator, which can lead to instability. Diffusion models, on the other hand, use a denoising objective based on a fixed noise schedule, resulting in more stable and predictable training. Although diffusion models require more compute and training time, they yielded higher-quality results in our Monet project.

**Q11: How would you scale this project for a larger or more diverse dataset?**

**A:** I would use larger, curated datasets of impressionist paintings across multiple artists, and potentially apply transfer learning from pre-trained generative models. Additionally, I’d explore distributed training to handle the computational load and implement early stopping and learning rate scheduling to optimize performance.

**Q12: What were some key takeaways from comparing different generative models?**

**A:** One key takeaway is that no single model is universally superior—DCGANs are fast and simple but unstable; WGAN-GP offers balance but still requires tuning; diffusion models are robust and high-quality but computationally intensive. Understanding these trade-offs is essential for choosing the right model depending on the problem and resources available.