# Portfolio


## Srecharan Selvam

sselvam@andrew.cmu.edu linkedin.com/srecharan srecharan.com GitHub

# Research Experience

# Autonomous Leaf Grasping: A Hybrid CV-ML Approach

**Project Overview:**

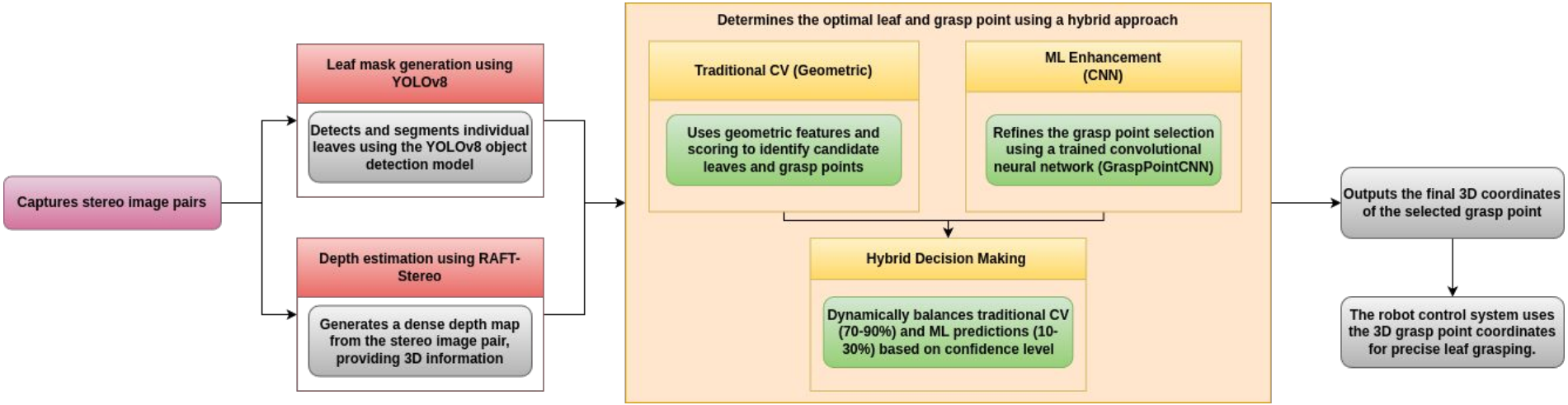
A real-time vision system for autonomous robotic leaf manipulation combining geometric computer vision techniques with deep learning. This hybrid system integrates YOLOv8 for leaf segmentation, RAFT-Stereo for depth estimation, and a custom CNN (GraspPointCNN) for grasp point optimization. The architecture features self-supervised learning that eliminates manual annotation, and a confidence-weighted decision framework that dynamically balances traditional CV algorithms with CNN predictions to achieve superior grasping performance.

**GitHub Repository:** LeafGrasp-Vision-ML

**Key Technologies and Skills Used:**

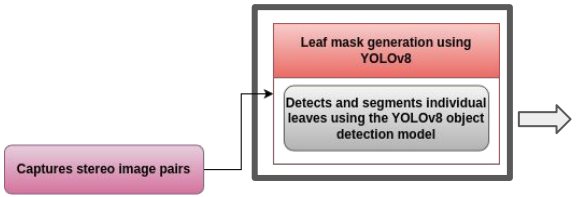
**Languages**: Python, C++
**Frameworks**: PyTorch, CUDA, OpenCV, Scikit-learn, Numpy, Pandas, Matplotlib, ROS2, MLflow
**Computer Vision**: Instance Segmentation, Depth Estimation, Point Cloud Processing, SDF, 3D Perception
**Deep Learning**: CNN Architecture Design, Self-Supervised Learning, Model Training & Optimization, Attention Mechanisms
**Cloud Computing:** AWS EC2

# Pipeline:

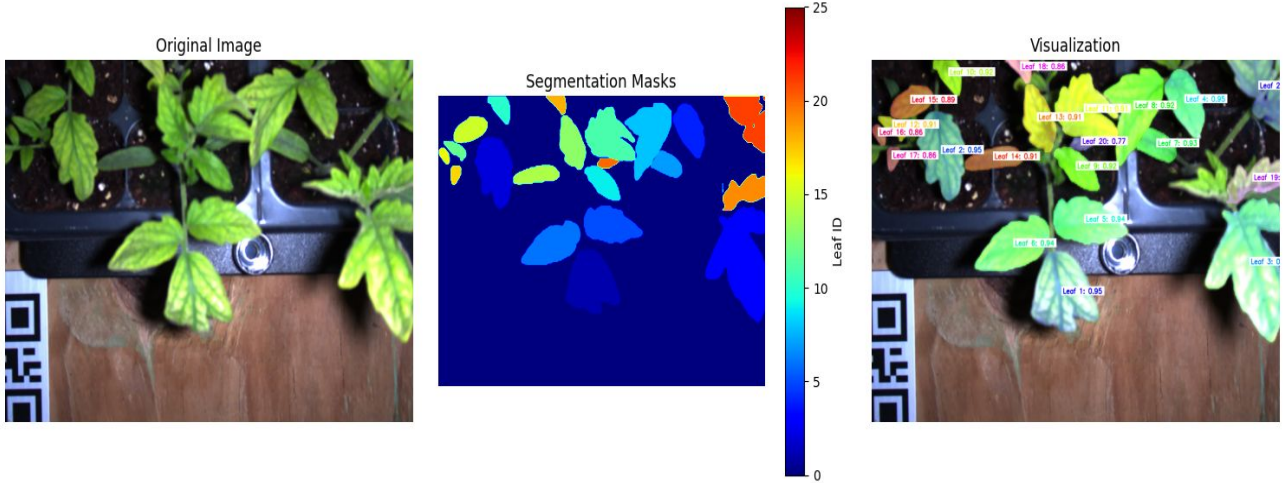



*High-Level System view*

# YOLOv8 Leaf Instance Segmentation




*High-Level System view*

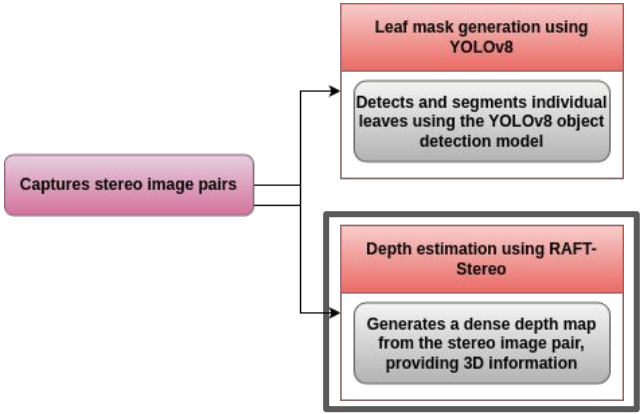



*Output from YOLOv8 pipeline: Left: Original RGB image showing tomato plant; Center: Generated segmentation masks; Right: visualization with leaf IDs and confidence scores*

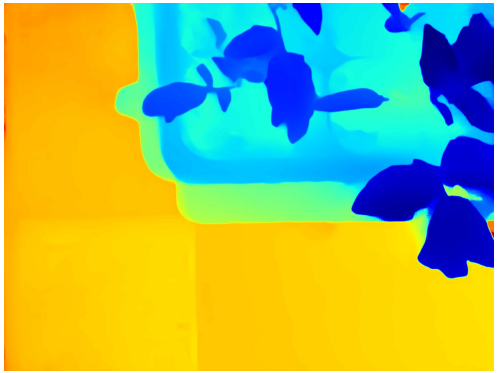
**Key Details:**

- Custom dataset: ~900 images of soybean and tomato plants
- 68% mAP@[0.5:0.95] for leaf mask generation

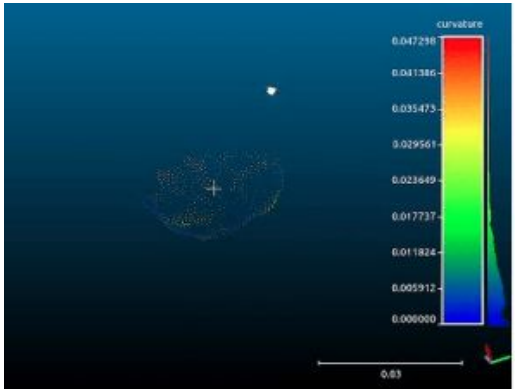
# High-Precision Depth Estimation with RAFT-Stereo




*High-Level System view*



*Depth Map*



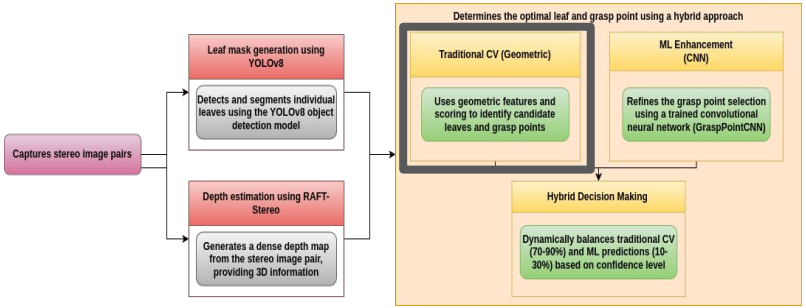

*Geometric capture of leaf surface*

**Key Technical Details:**

- Recurrent GRU refinement (the disparity map)
- 4D correlation volume computation
- Sub-pixel disparity accuracy (<0.5px) on 1080p stereo pairs
- Processing speed: ~150ms total pipeline latency on RTX 3080


*Reconstructed 3D point cloud enabling precise spatial understanding*

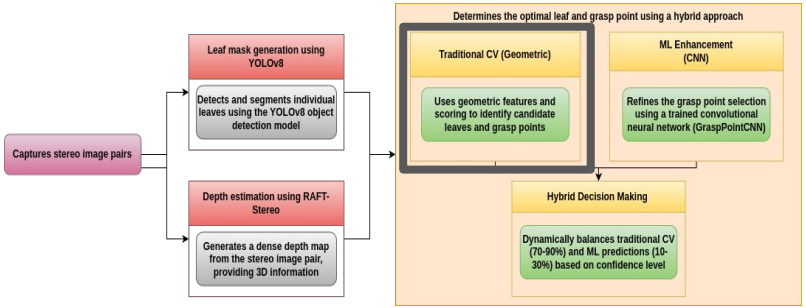
# Traditional CV Pipeline: Geometric Grasp Point Selection




*High-Level System view*


*Original camera view with leaf midrib*

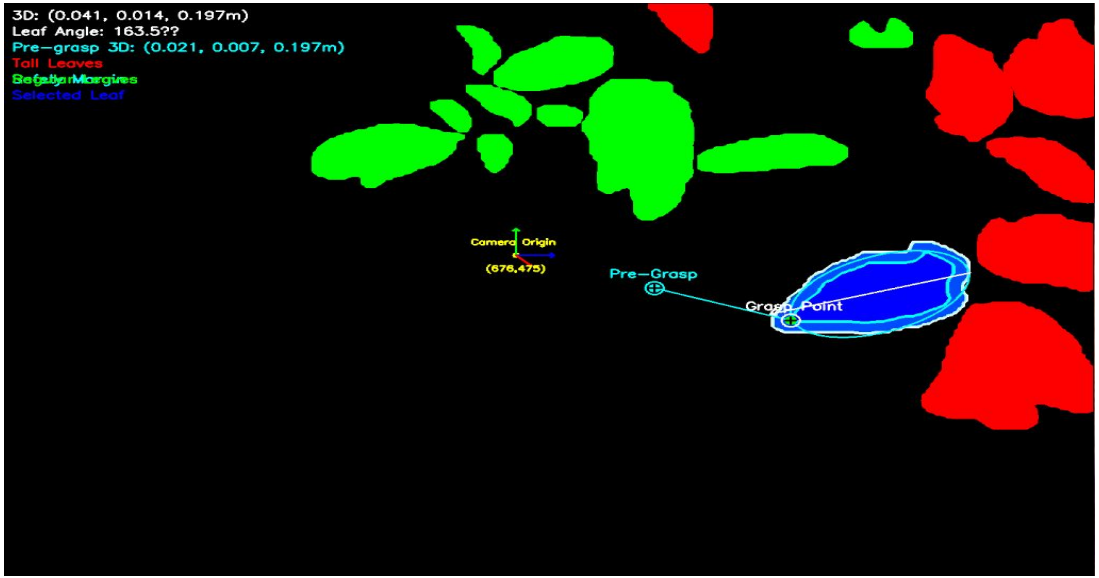
# Traditional CV Pipeline: Geometric Grasp Point Selection




*High-Level System view*

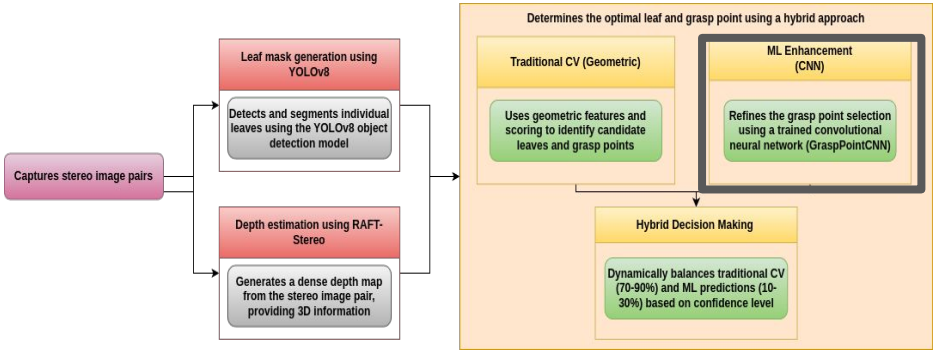
**Optimal Leaf Selection (Pareto Optimization):**
- Clutter Score (35%): Leaves that are isolated
- Distance Score (35%): Leaves that are closer to camera
- Visibility Score (30%): Leaves that are fully visible

**Grasp Point Selection:**
- Flatness Score (25%): Flatter regions of the leaf surface
- Approach Vector Score (40%): Approach vector (from the camera to the point) is closer to the camera's Z-axis
- Accessibility (15%): Distance of the grasp point from the camera origin and position relative to the center of the image
- Edge Awareness (20%): Grasp points within leaf boundary




*Segmented leaves with optimal leaf & grasp point visualization*

# GraspPointCNN: ML-Based Grasp Refinement




*High Level System view*

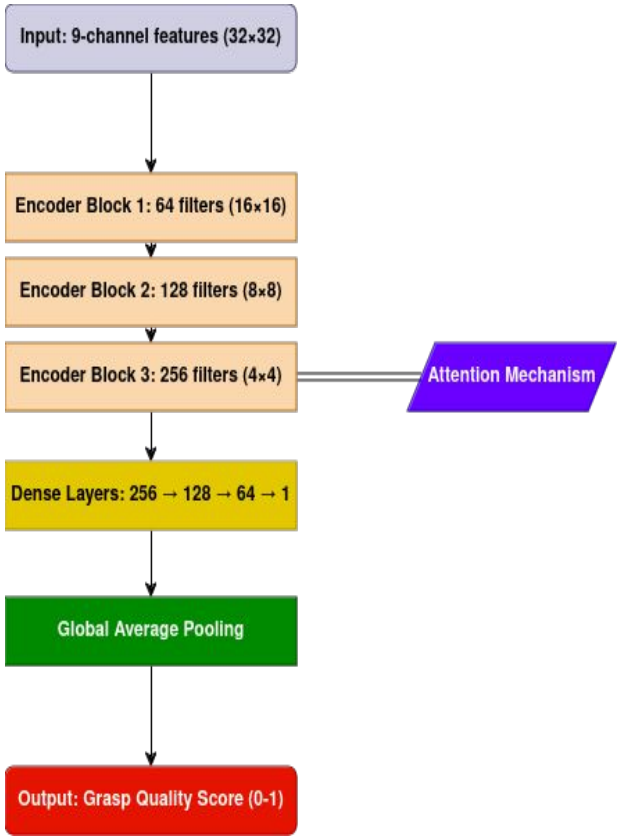
**Self-Supervised Learning Pipeline:**

**- Input: 9-channel features (32×32)**

- Depth map (1)
- Binary mask (1)
- Score maps (7): SDF, approach vector, flatness, isolation, distance, accessibility, stem penalty

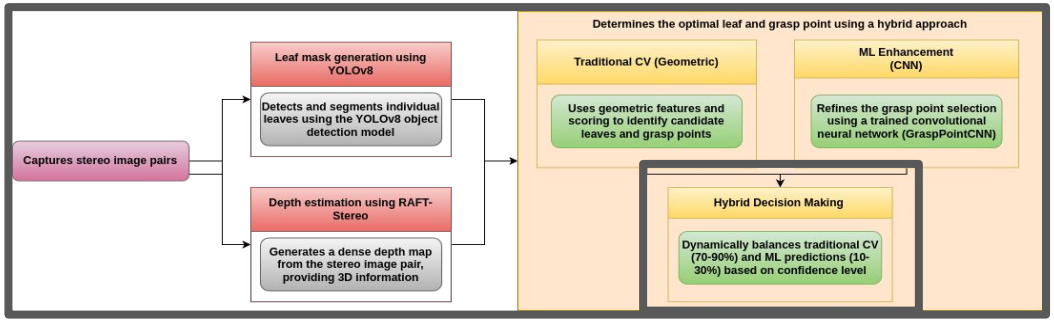
**- Training Configuration:**

- BCEWithLogitsLoss (pos_weight=2.0)
- Adam (lr=0.0005, weight_decay=0.01)
- ReduceLROnPlateau scheduler
- Early stopping (patience=15)
- Data augmentation: 90°,180°,270° rotations

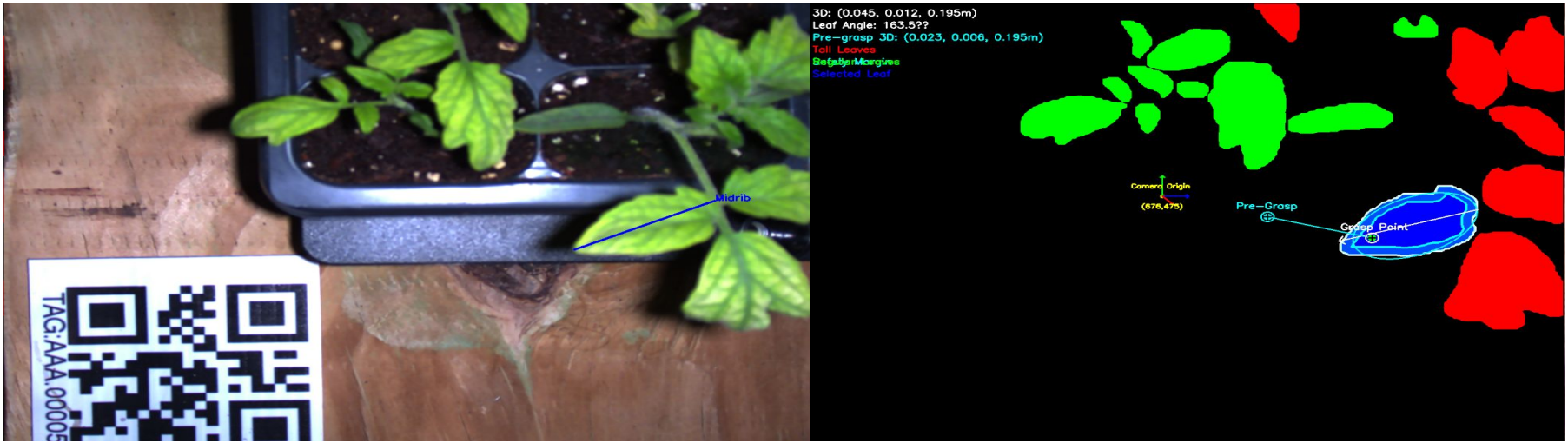



*GraspPointCNN Architecture*

# Hybrid Decision Integration: Combining CV & ML

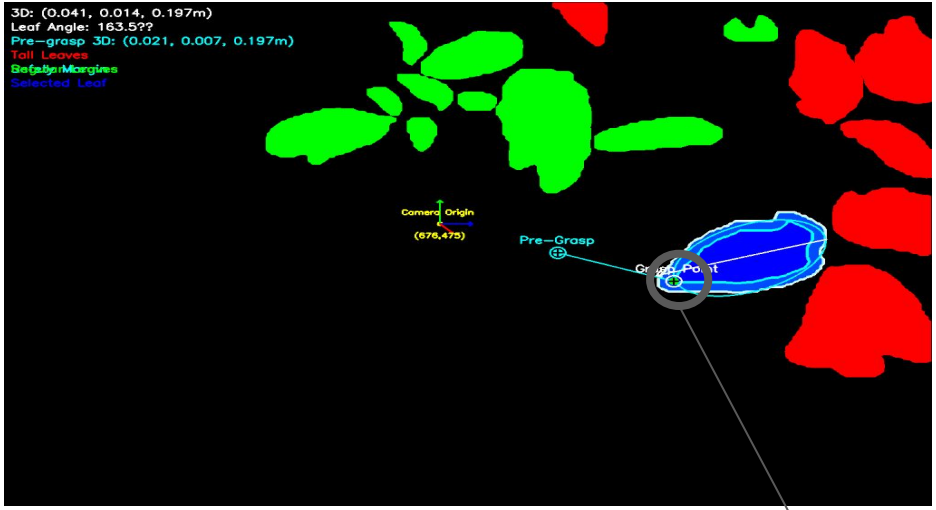



*High-Level System view*

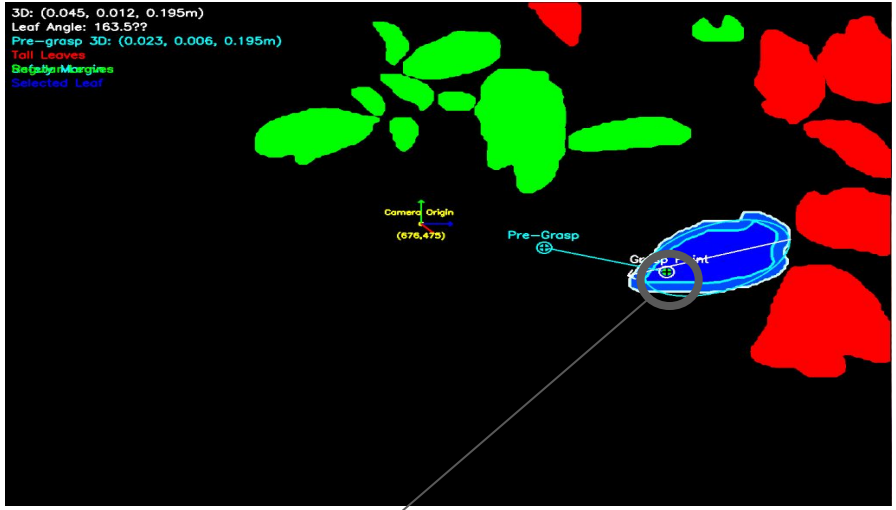



*Hybrid CV-ML grasp point selection: Left - Original camera view with leaf midrib ; Right - Segmented leaves with grasp point visualization*

# Traditional CV vs Hybrid Decision System



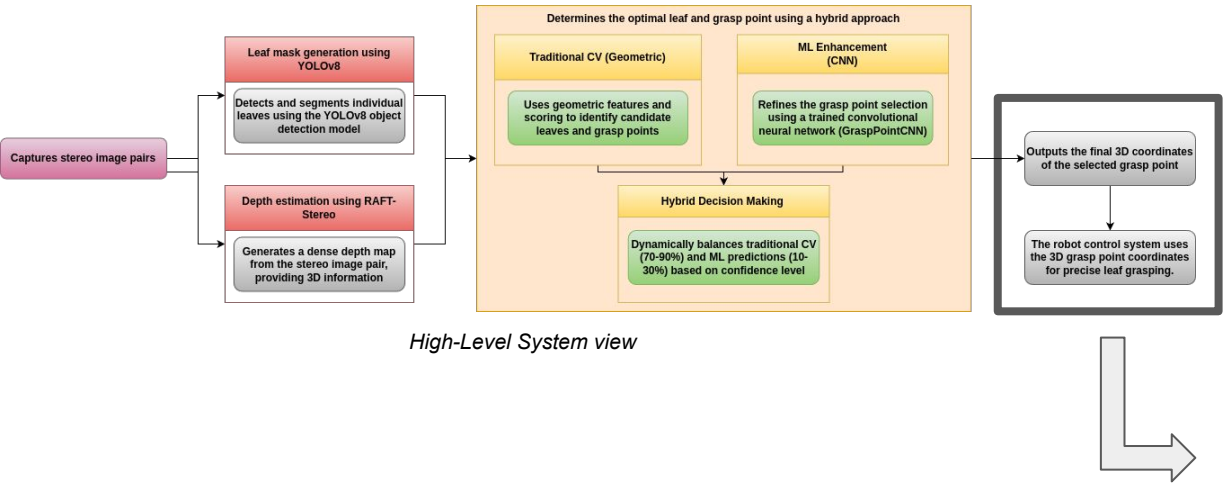

Traditional CV grasp point selection




Hybrid CV-ML grasp point selection

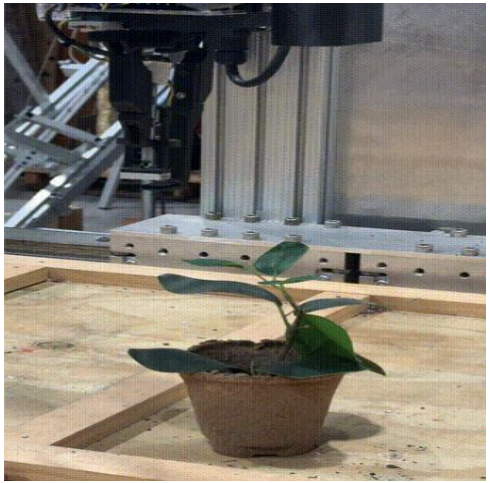
Grasping at the leaf tip often fails as the REX robot struggles to secure it, leading to missed grasps or leaf displacement. The hybrid grasp point selection method outperforms traditional CV, achieving a 4.66% improvement over 150 test cases.

# REX Robot Integration for Leaf Grasping

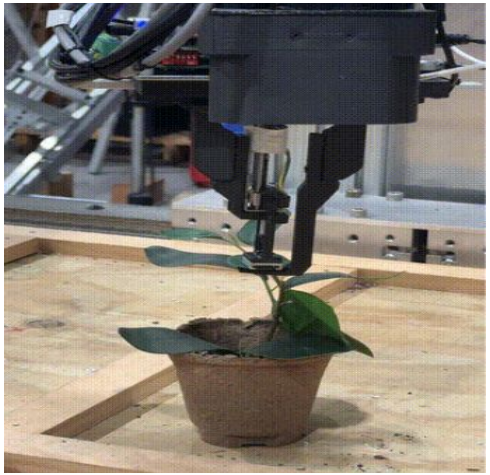



*High-Level System view*

(1)



(2)



*(1) Robot capturing image (2) Robot grasping the leaf*

# Professional Experience

# Real-time Hand Gesture Recognition for AR Interaction

**Project Overview:**

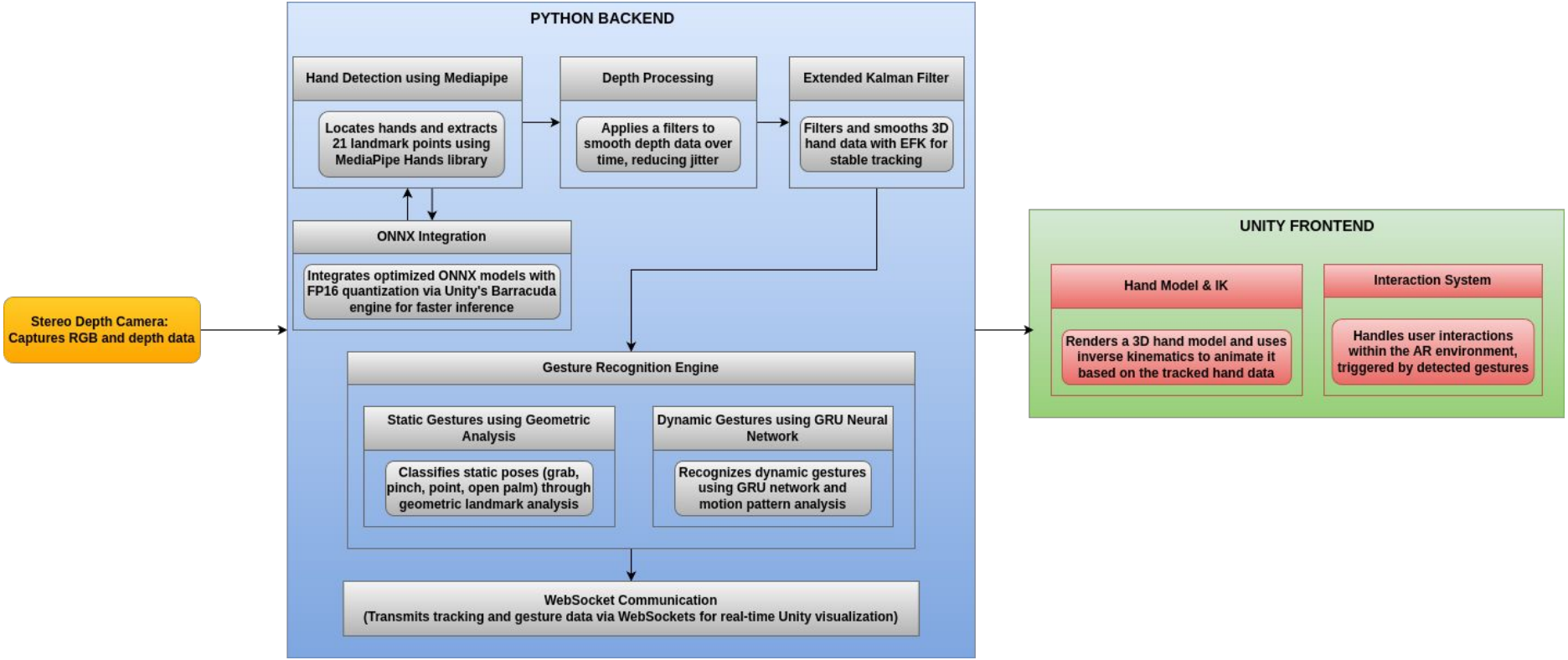
A comprehensive hand tracking and gesture recognition system built for augmented reality applications in automotive training. This system combines advanced computer vision techniques with deep learning to enable intuitive interaction with virtual HVAC components. The architecture integrates Extended Kalman Filtering for precise 3D hand tracking (<7.5mm accuracy), geometric analysis for static gesture recognition (97% accuracy), and a custom GRU neural network for dynamic gesture detection (<30ms latency). Features a Python backend for processing and a Unity frontend for visualization, connected via WebSockets for real-time, low-latency performance at 30+ FPS with optimized ONNX models.

**GitHub Repository:** VirtuHand

**Key Technologies and Skills Used:**

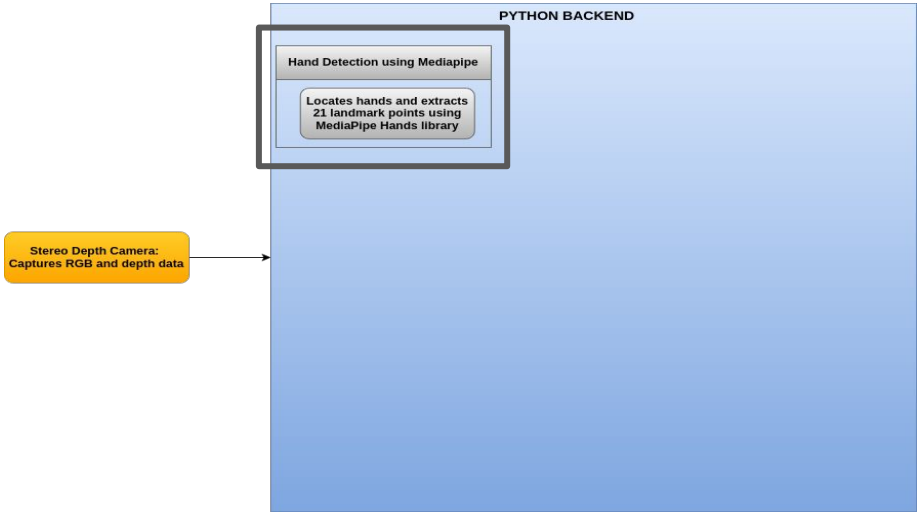
**Languages & Frameworks**: Python, PyTorch, ONNX, scikit-learn, MediaPipe, Websockets, OpenCV, NumPy
**Machine Learning:** 3D Tracking, Landmark Detection, Depth Sensing, Geometric Analysis, Kalman Filtering
**Deep Learning**: Recurrent Neural Networks (GRU), Model Optimization (ONNX), Inference (Unity Barracuda)
**Augmented Reality (AR):** Unity Development, 3D Interaction Design

# Pipeline:

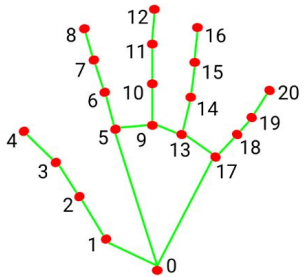



*System Architecture and Data Flow*

# Hand Detection and Landmark Extraction with MediaPipe

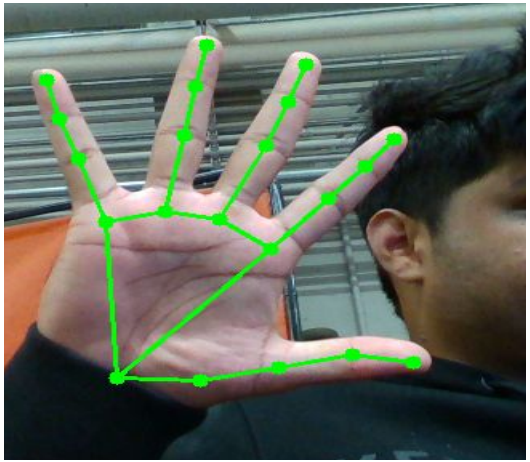


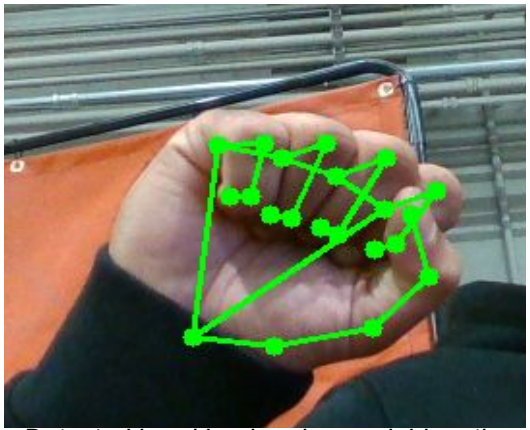

*System Architecture and Data Flow*




0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

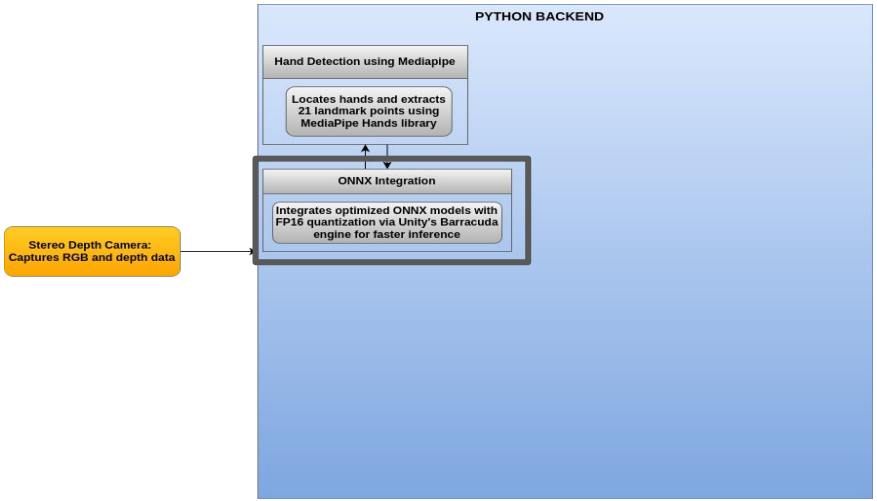
*Uses Google's MediaPipe Hands library to detect hand presence and location in the RGB image*



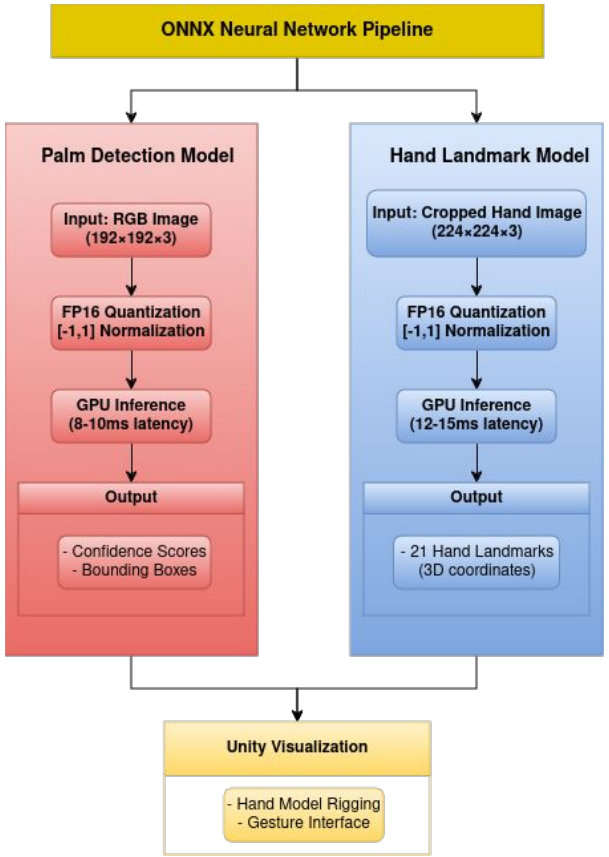


*Detected hand landmarks overlaid on the original image in real-time*

# Neural Network Optimization with ONNX




*System Architecture and Data Flow*

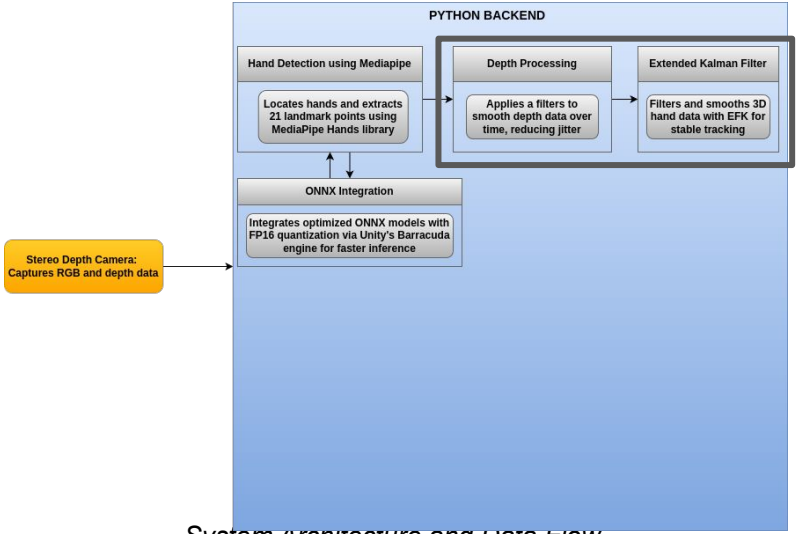



*Two-stage ONNX pipeline*

To improve performance and reduce reliance on external libraries, an experimental pipeline was developed using ONNX (Open Neural Network Exchange) models for hand detection and landmark extraction. This approach leverages the Unity Barracuda engine for GPU-accelerated inference.

1. **Performance Optimization**
   - FP16 quantization reducing model size by 50%
   - 33% faster inference compared to MediaPipe

# 3D Hand Tracking with Extended Kalman Filter

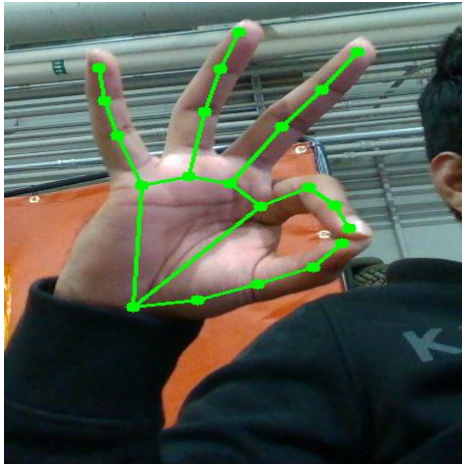



*System Architecture and Data Flow*

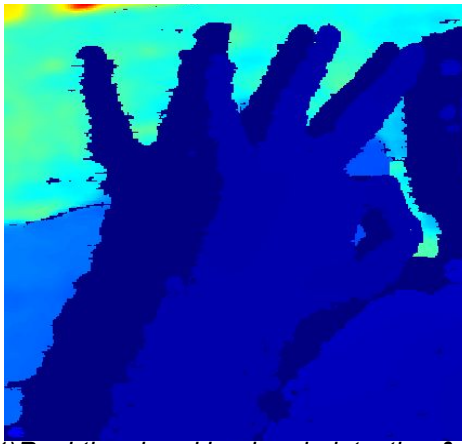
- Combines MediaPipe landmarks with RealSense depth data and applies Extended Kalman Filtering to achieve smooth 3D tracking at 30Hz, with robustness to occlusions and jitter reduction.
- Enables precise velocity estimation and maintains tracking during fast hand movements


*Multi-stage depth filtering for 3D hand tracking*

(1)

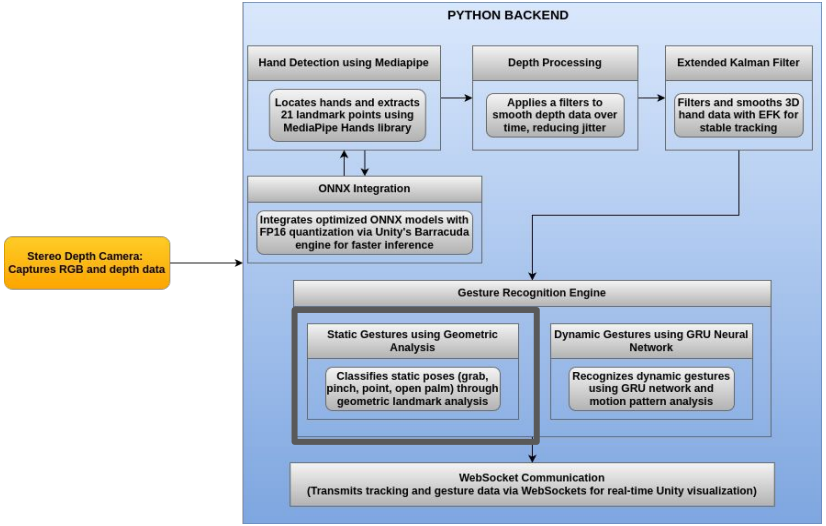


(2)

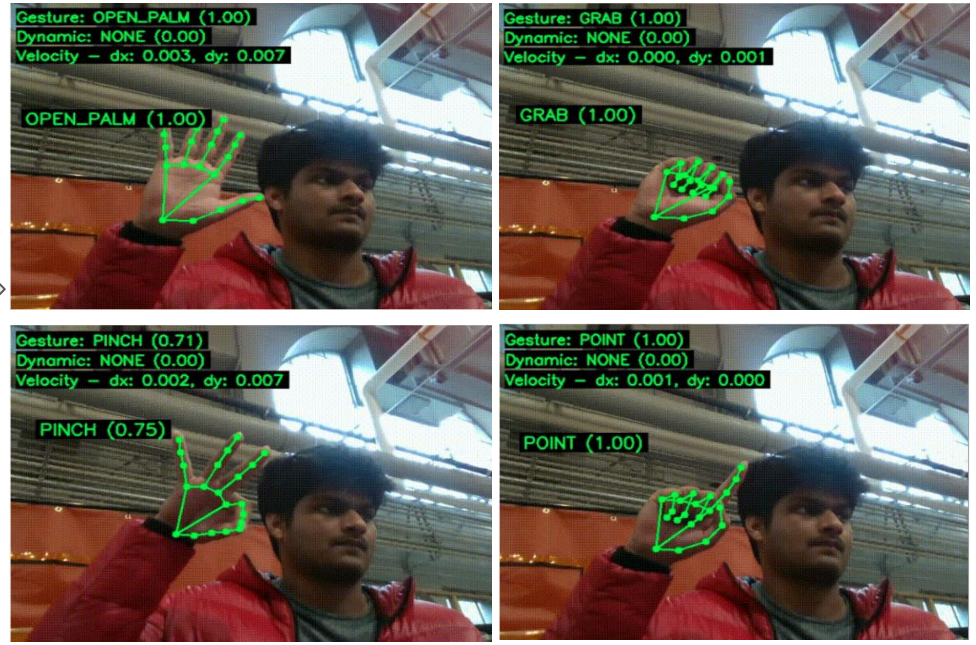


*(1)Real-time hand landmark detection &*
*(2) Visualization of depth map using Filter*

# Static Gesture Recognition



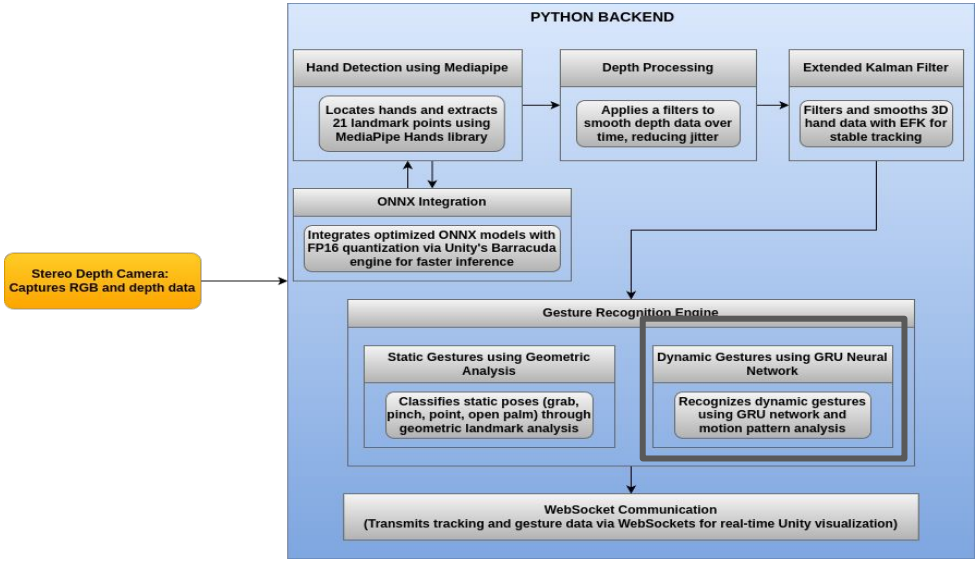

*System Architecture and Data Flow*




*Real-time detection of OPEN_PALM, GRAB, PINCH, and POINT gestures*

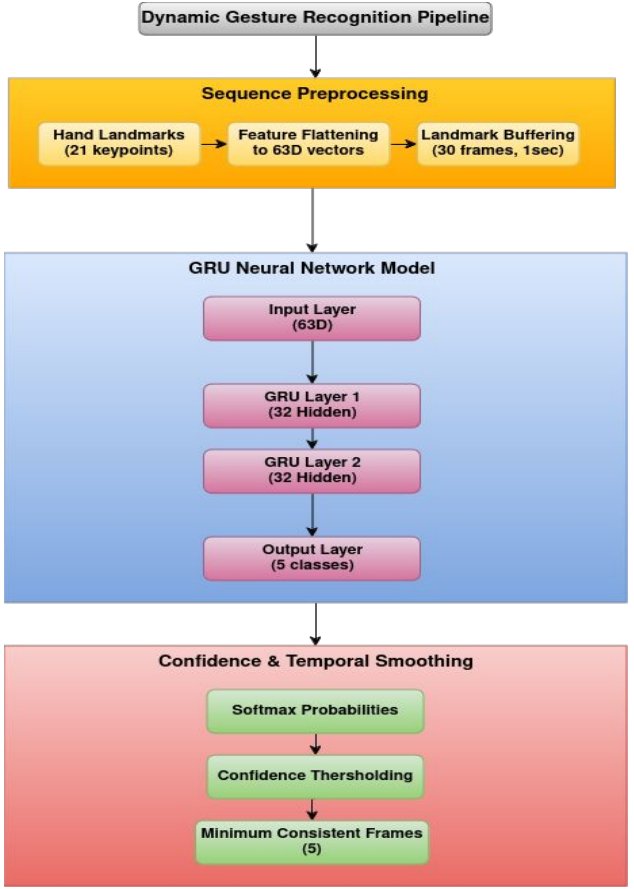
| Gesture | Description | Detection Logic |
|---|---|---|
| Open Palm | All fingers extended, hand open. | All fingertips are further from wrist than their corresponding base joints. Thumb is extended. |
| Pinch | Thumb and index finger close together. | Distance between thumb tip and index fingertip below threshold with one finger near thumb |
| Grab | Fingers curled inwards towards the palm. | All fingertips are closer to wrist than their corresponding base joints. Thumb is also curled. |
| Point | Index finger extended, others closed. | Index fingertip is extended (from wrist than its base). All other fingertips are curled |

# Dynamic Gesture Recognition




*System Architecture and Data Flow*

- A GRU network is a type of recurrent neural network (RNN) that is well-suited for processing sequential data, such as the time series of hand landmark positions.
- The GRU model was trained on a custom dataset of dynamic gestures.
- Input: The model takes a sequence of 30 frames of hand landmark data (21 landmarks x 3 coordinates = 63 input features).
- Output: The model predicts the probability of each supported dynamic gesture.
- Architecture : Input size is 63, hidden layer is 32 and 2 layers.

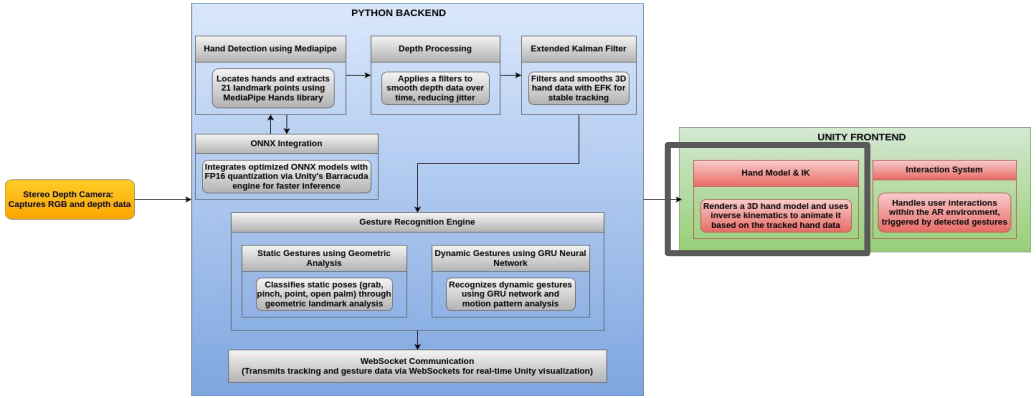



*GRU Architecture Diagram*

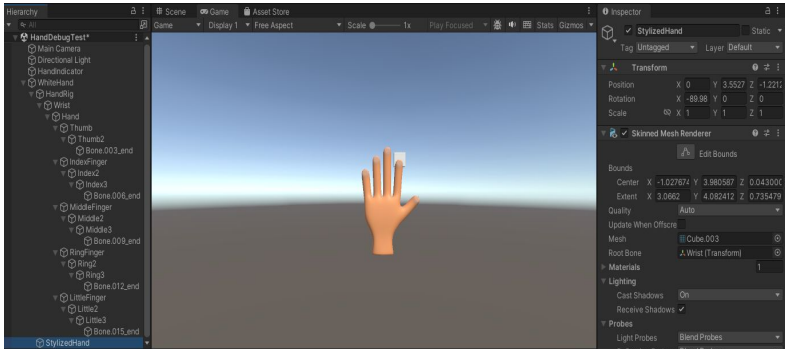
# Dynamic Gesture Recognition


*Real-time detection of dynamic gestures (SWIPE_RIGHT, SWIPE_LEFT, CIRCLE) using the GRU model and motion pattern analysis*

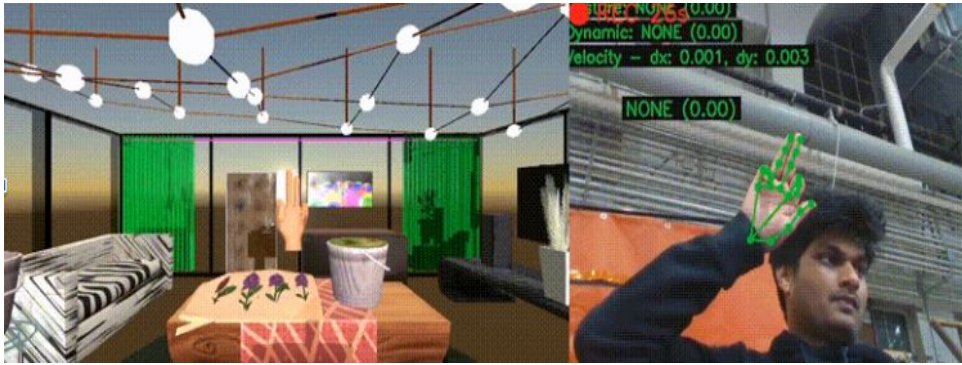
# Unity Frontend: Hand Rigging and Interaction




*System Architecture and Data Flow*

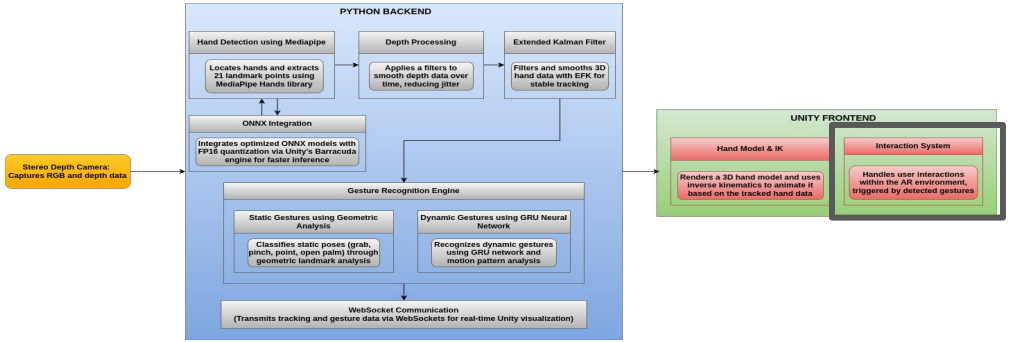


*A rigged 3D hand model within the Unity editor*

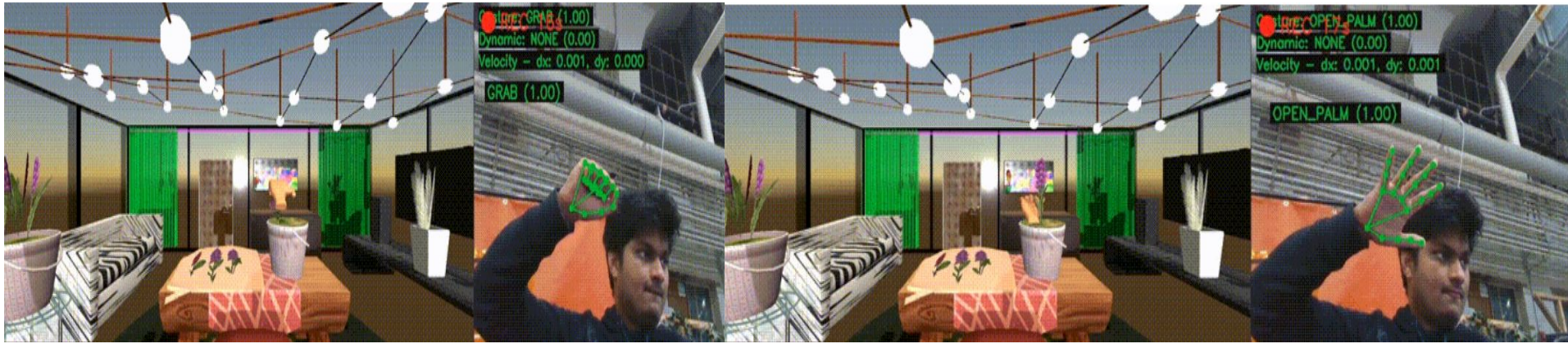




*Real-time hand rigging in Unity. The 3D hand model accurately mirrors the user's hand movements and gestures.*

# Unity Frontend: Real-time AR Implementation & Testing




*System Architecture and Data Flow*




*Sequence demonstrating the virtual flower arrangement demo. The user can grab, move, and place flowers using hand gestures*

# Multi-Camera Vision System for Automated Material Detection and Sorting

**Project Overview:**

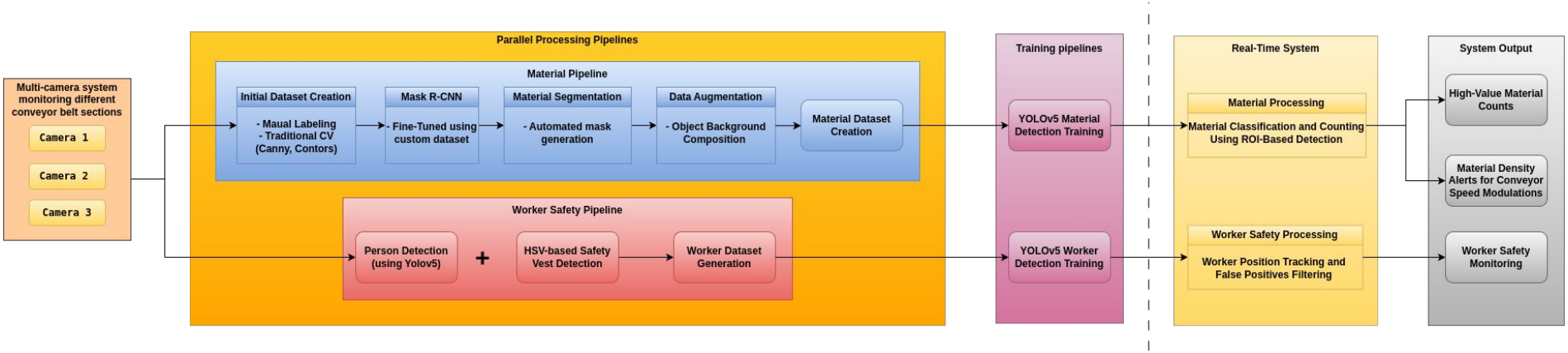
A real-time computer vision system for enhancing high-value material recovery and worker safety monitoring on industrial conveyor belts. This system combines YOLOv5 for precise material detection and segmentation with intelligent background subtraction for motion analysis. The architecture features camera-specific region-of-interest (ROI) processing, worker-interaction filtering to minimize false positives, and a robust counting mechanism. The system achieved 96.8% mAP@[0.5:0.95] for material detection and 74.5% mAP@[0.5:0.95] for worker detection, with <15ms inference latency, enabling real-time monitoring across multiple camera feeds.

**GitHub Repository:** CVAnnotate

**Key Technologies and Skills Used:**

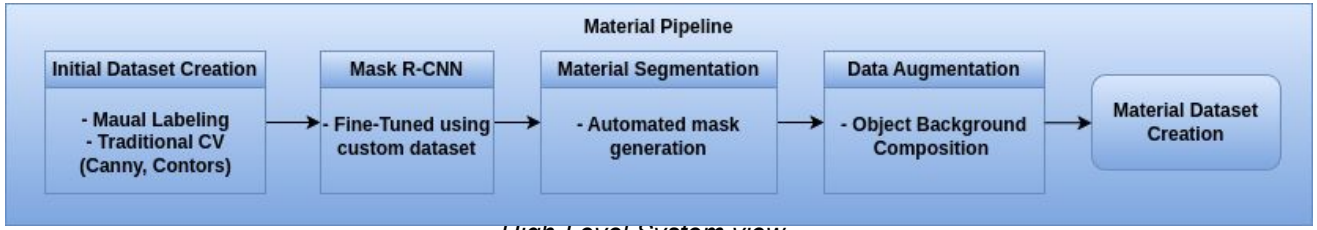
**Languages:** Python, C++
**Frameworks**: PyTorch, OpenCV, scikit-learn, NumPy, Pandas
**Machine Learning:** Model Training, Fine-tuning, Data Augmentation, Model Evaluation (mAP, Precision, Recall)
**Computer Vision:** Object Detection (YOLOv5), Instance Segmentation (Mask R-CNN), Multi-Camera Systems, Region of Interest (ROI) Processing, Background Subtraction (MOG2), Image Processing

# Pipeline



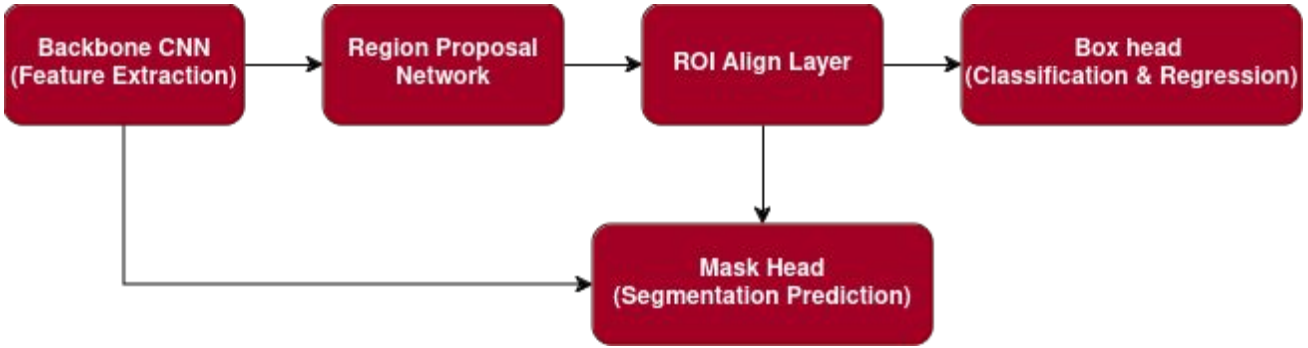

*End-to-end pipeline for automated material sorting and worker safety, with parallel processing for offline training and real-time operation*

# Semi-Automated Data Annotation and Augmentation




*High-Level System view*

*Initial Dataset Creation using labelme*

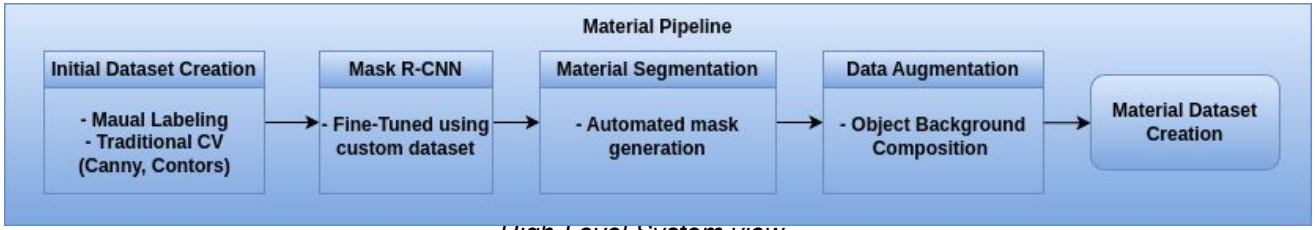



*Simple Mask R-CNN Architecture Diagram*

1. **Initial Dataset:** A small initial dataset of ~800 images (200 per material class) was created using a combination of manual annotation (with LabelMe) and traditional computer vision techniques (Canny edge detection, adaptive thresholding) to generate initial segmentation masks.

2. **Mask R-CNN Fine-tuning:** A pre-trained Mask R-CNN model was fine-tuned on this initial dataset, significantly improving segmentation accuracy.

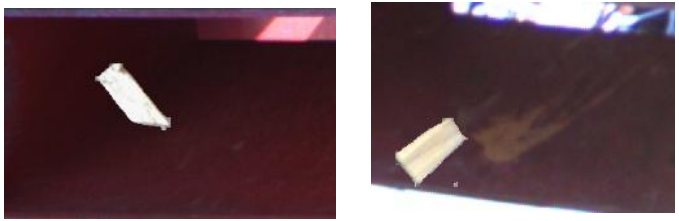
# Data Augmentation and Dataset Creation




*High-Level System view*

*Initial Dataset Creation using labelme*

*Automated Masked & Segmented images*

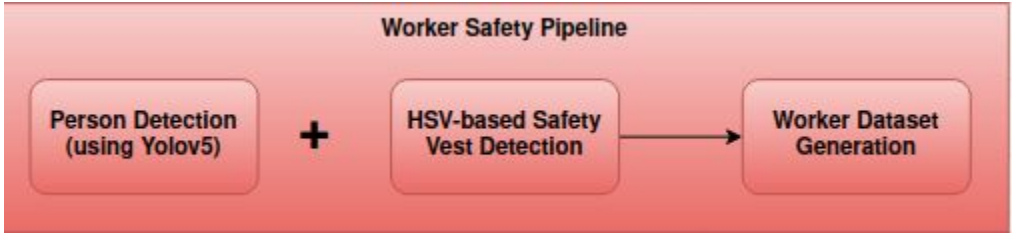


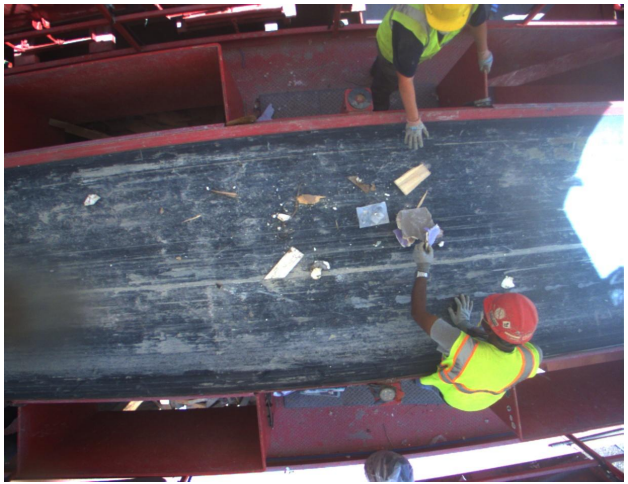
*Data Augmentation (Object-Background Compositing)*

**Automated Segmentation:** The fine-tuned Mask R-CNN was then used to automatically generate segmentation masks for a much larger dataset of 43,000+ images captured from the live camera feeds. This drastically reduced manual annotation effort.

**Data Augmentation**: To create a robust and diverse training dataset, a sophisticated data augmentation strategy was employed. Segmented material images (generated by the Mask R-CNN model) were overlaid onto images of the empty conveyor belt bins

# Worker Detection Dataset Creation



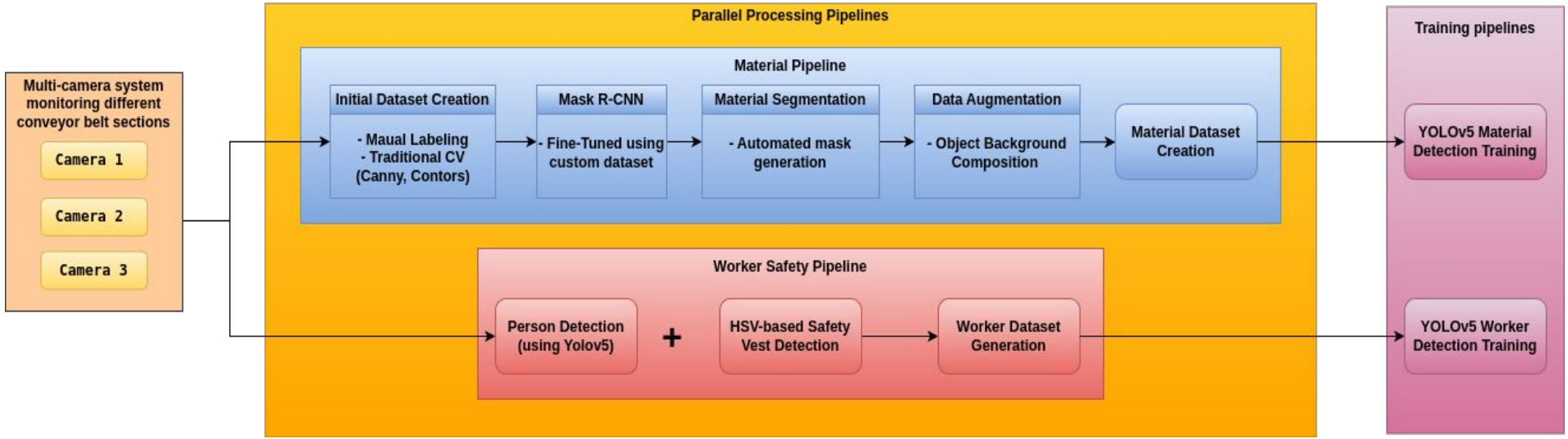



*Raw image from the camera*

*Worker detection using a combination of HSV-based color thresholding (for initial detection) and a trained YOLOv5 model (for refined bounding box prediction)*

A separate dataset was created for training the worker detection model. To quickly gather initial bounding box annotations, a color-based detection method was used, targeting the bright yellow safety vests commonly worn by workers. This initial detection was then used to train a YOLOv5 model for more robust person detection.

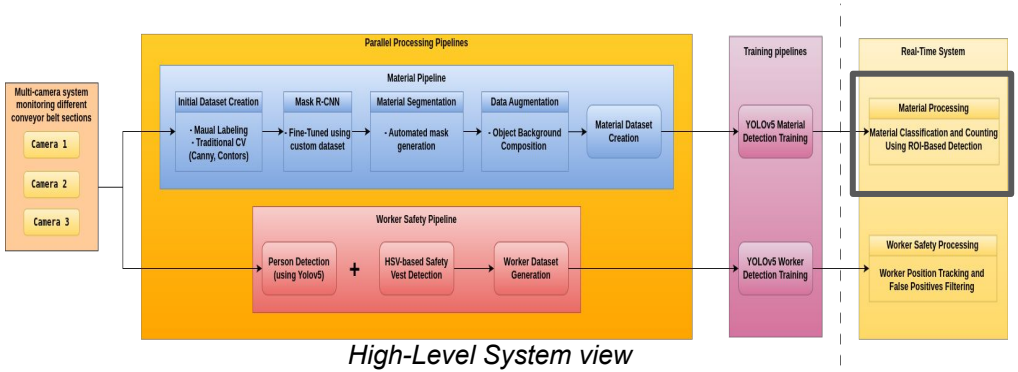
# Detection Model Training and Performance




**Performance:**

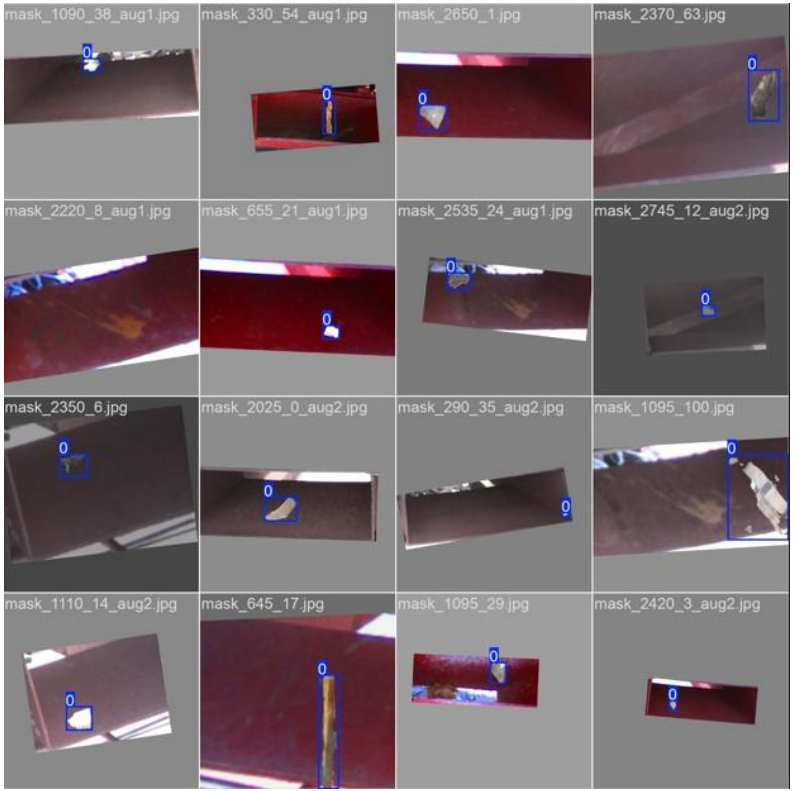
- Material Detection Model: 96.8% mAP@[0.5:0.95]
- Worker Detection Model: 74.5% mAP@[0.5:0.95]

# Real-Time Processing System



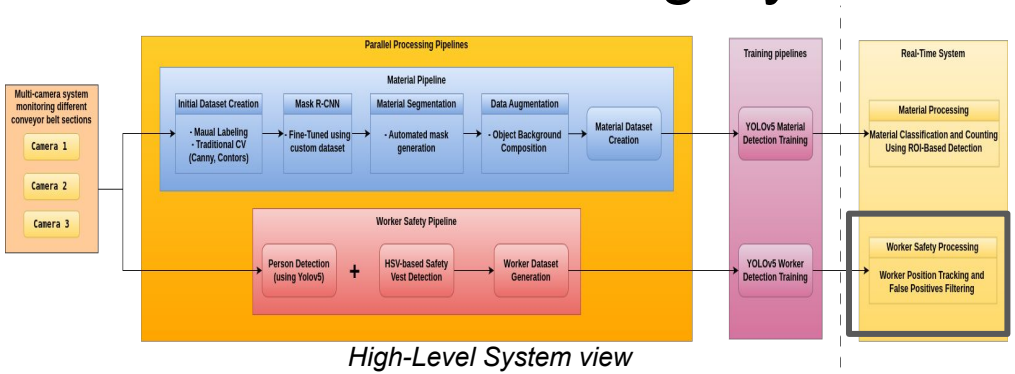

*High-Level System view*

*Real-time material detection and counting. Green boxes indicate detected and counted materials within the defined ROI*




*Real-time detection and counting results from the trained model*

# Real-Time Processing System




*High-Level System view*


*Real Time worker position tracking*

**False Positive Filtering:**

- **Worker Overlap:** Detected materials overlapping with detected workers are *not* counted. This prevents miscounting worker interactions as materials.
- **Cooldown:** After counting an object in an ROI, a short cooldown prevents immediately recounting the *same* object.

*Examples of False Positives*

# Projects

# DeepTrade AI: Multi-Model Stock Prediction with NLP & Automated Trading

**Project Overview:**

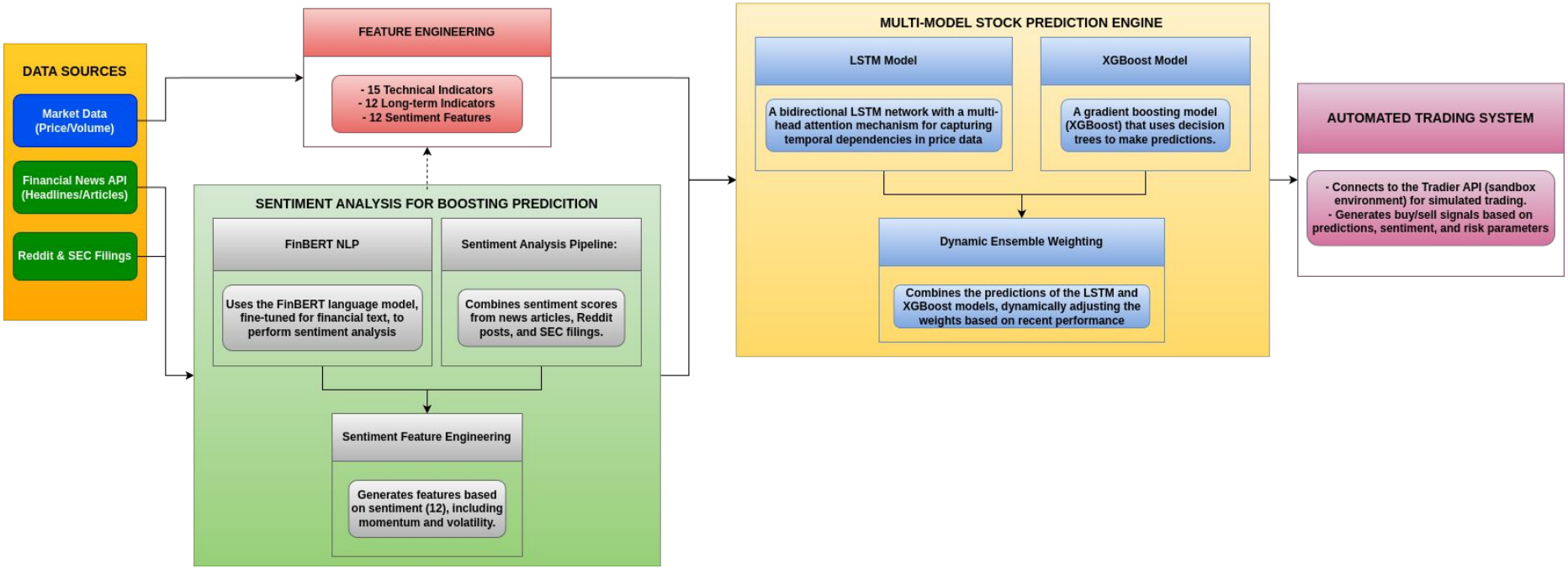
An end-to-end automated stock trading system that combines machine learning price prediction with NLP-based sentiment analysis. The system features a bidirectional LSTM with attention mechanism and XGBoost ensemble for multi-timeframe price forecasting, and integrates FinBERT for real-time sentiment analysis of financial news, Reddit posts, and SEC filings. The architecture employs dynamic model weighting, comprehensive risk management controls, and simulated execution through the Tradier API, achieving 55-65% directional accuracy and a 58.5% win rate in paper trading.

**GitHub Repository:** DeepTrade-AI

**Key Technologies and Skills Used:**

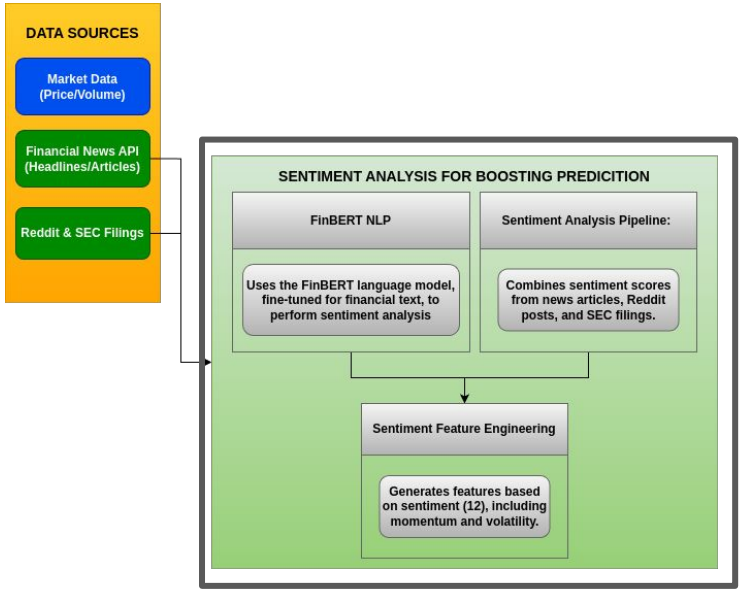
**Languages & Frameworks**: Python, PyTorch, TensorFlow, CUDA, Scikit-learn, Pandas, NumPy, Hugging Face Transformers
**Machine Learning:** Gradient Boosting (XGBoost), Feature Engineering, Regression, Time Series Forecasting
**Deep Learning**: LSTM Networks (Bidirectional, Attention), Model Ensembling, Model Training & Hyperparameter Optimization
**Cloud Computing:** AWS SageMaker (for distributed model training and hyperparameter optimization)
**Natural Language Processing (NLP):** FinBERT, Sentiment Analysis, Text Processing, Financial Text Mining

# Pipeline:

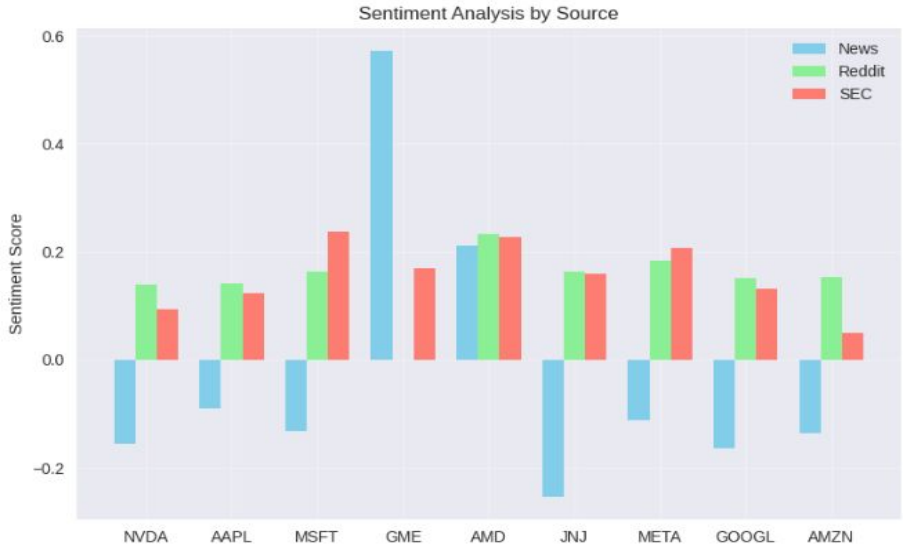



*System Architecture and Data Flow*

# Multi-Source Sentiment Analysis:




*System Architecture and Data Flow*

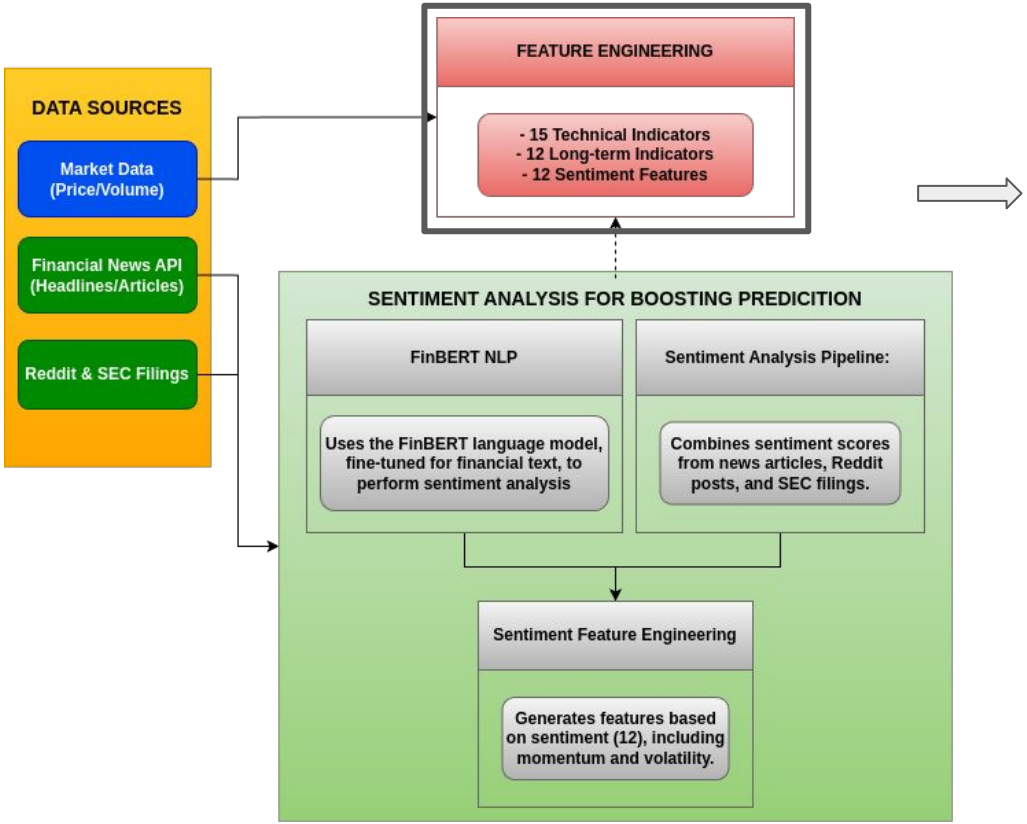



*Sentiment scores for selected stocks; Positive values indicate positive sentiment, negative values indicate negative sentiment, and values near zero indicate neutral sentiment*

**Key Details:**

- Tokenization: "Converts text data into numerical representations for the FinBERT model."
- 3-class Classification: "Classifies sentiment as positive, negative, or neutral."
- Multi-Source Weighted Integration: Financial News (40%)/Reddit(30%)/SEC(30%)

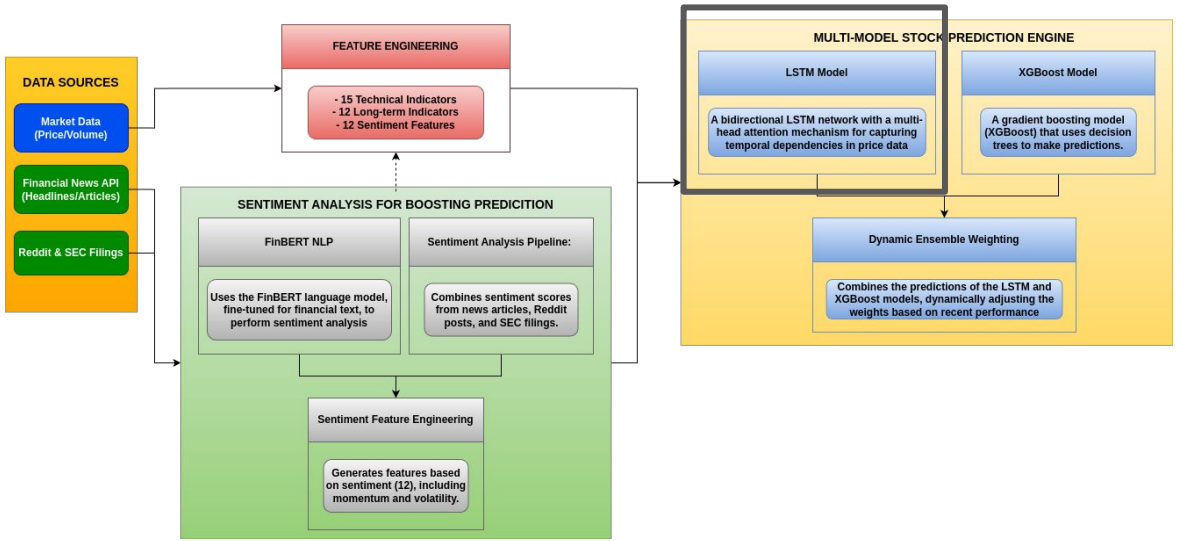
# Feature Engineering:




*System Architecture and Data Flow*

The prediction models utilize a comprehensive set of 39 features, encompassing technical indicators, long-term trend indicators, and sentiment-derived features. This rich feature set provides a holistic view of market dynamics, enabling the models to capture complex relationships and patterns

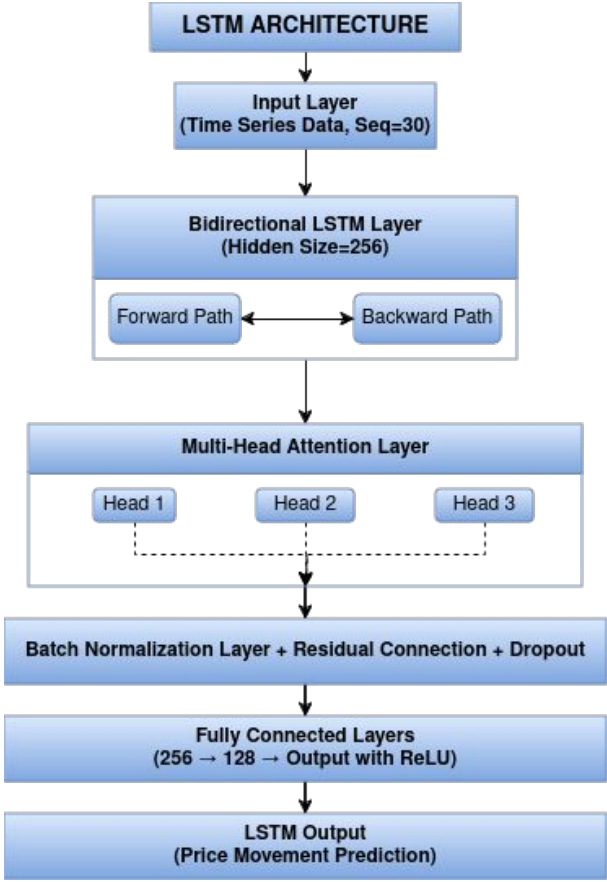
# LSTM for Price Prediction




*System Architecture and Data Flow*

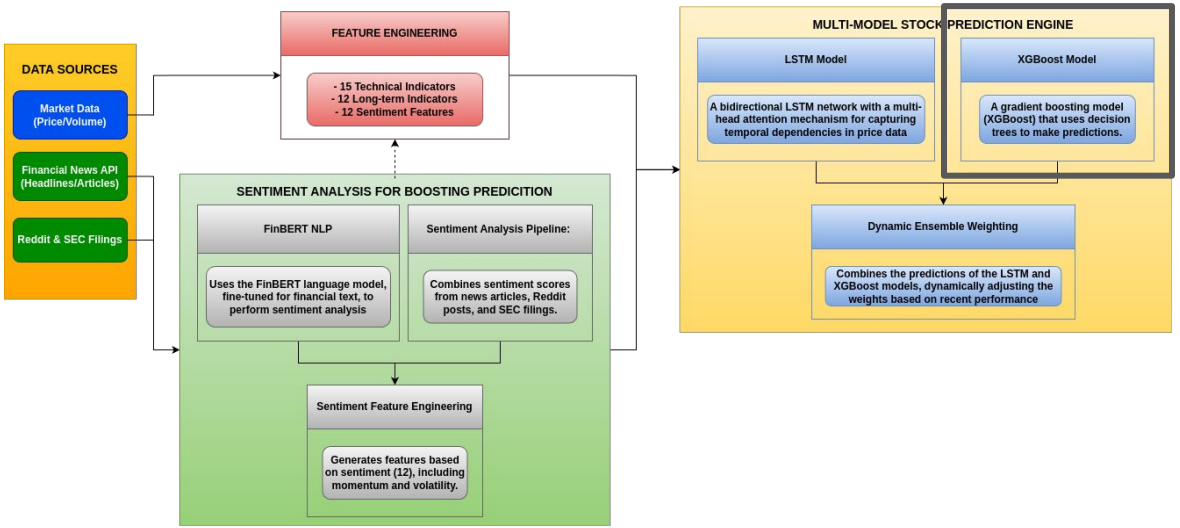
**LSTM Neural Network:**

- Bidirectional LSTM: The network processes the input sequence in both forward and backward directions, allowing it to learn from past and future context.
- Multi-Head Attention: This mechanism allows the model to focus on different parts of the input sequence that are most relevant for prediction. The model uses 3 attention heads.
- Batch Normalization: Batch normalization layers are used after each LSTM layer to improve training stability and speed.




*LSTM Model Architecture*

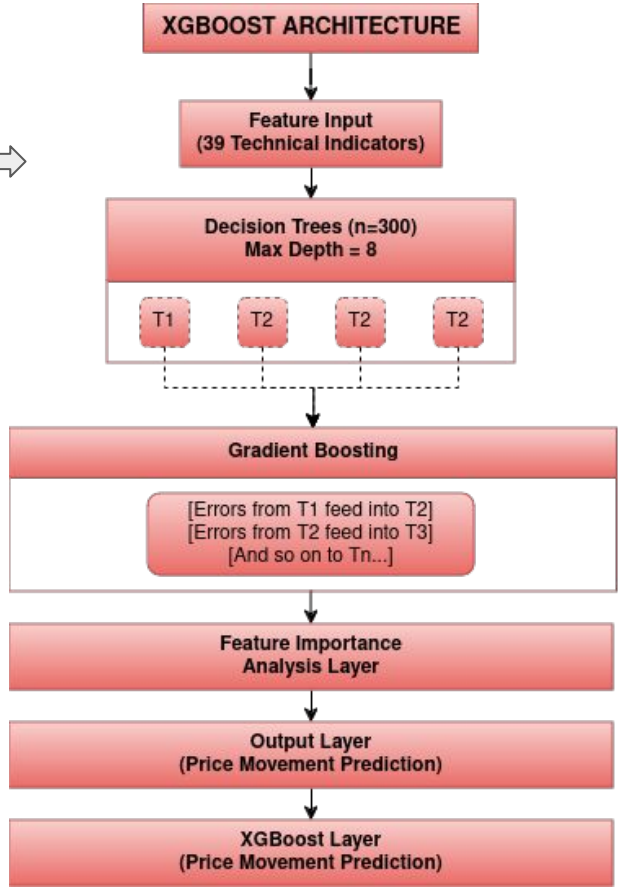
# XGBoost for Price Prediction




*System Architecture and Data Flow*

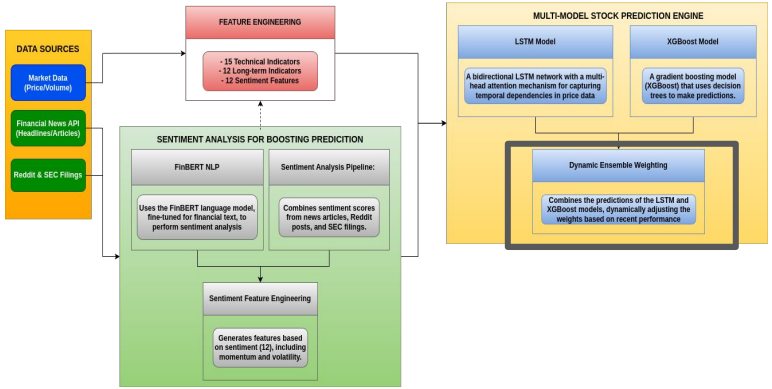
**XGBoost Ensemble:**

- Gradient boosting with 300 decision trees
- Feature engineering including 39 market indicators
- Min child weight:3, Subsample:0.8, Max depth:8



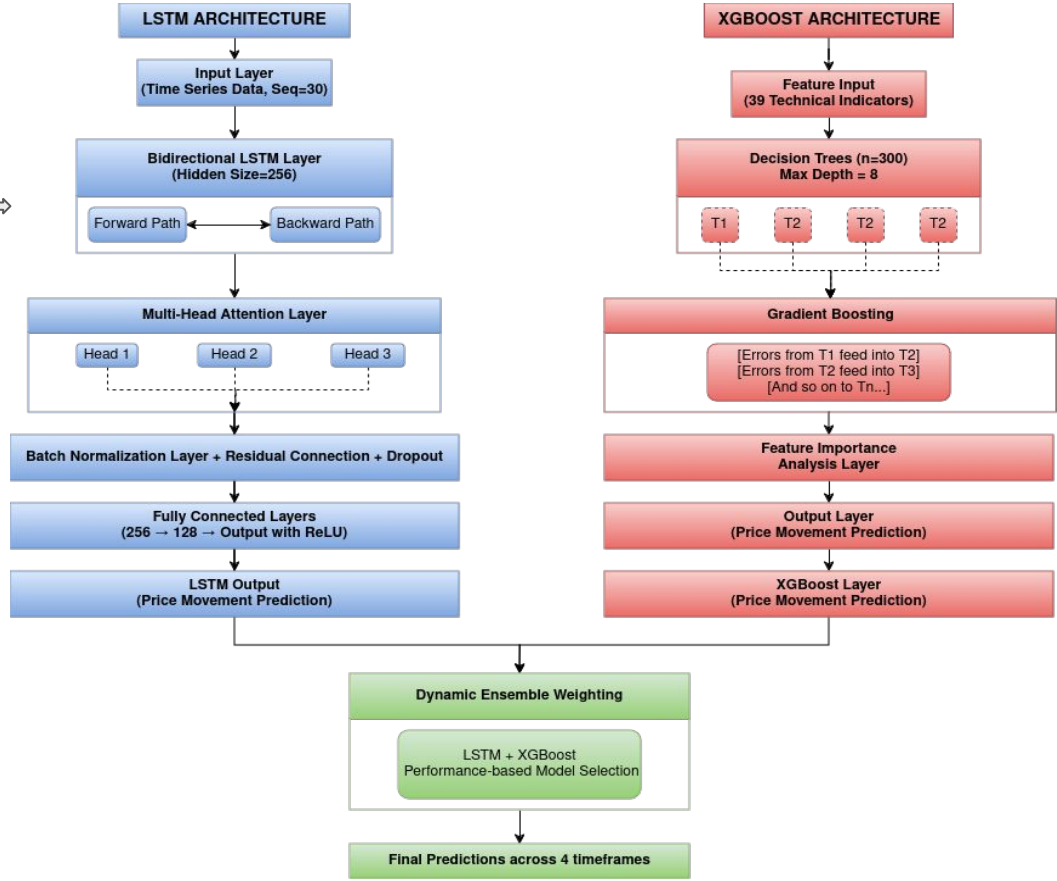

*XGBoost Model Architecture*

# LSTM-XGBoost Ensemble for Price Prediction



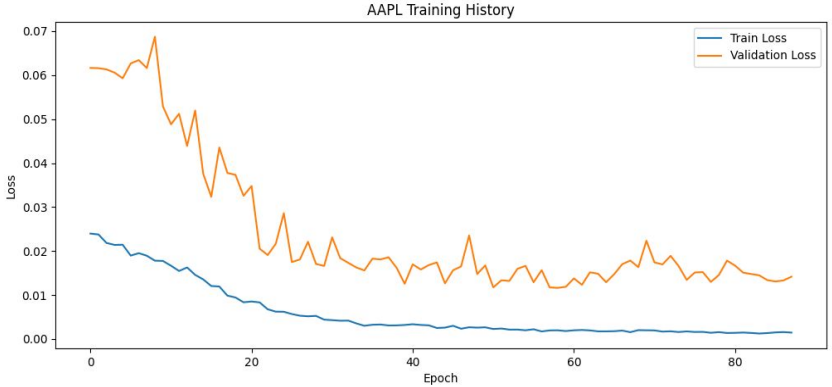

*System Architecture and Data Flow*

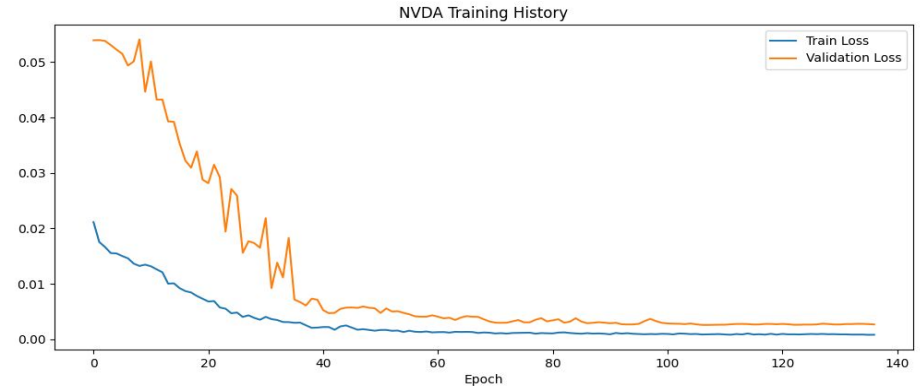
The trained ensemble model achieves a directional accuracy of 55-65% across multiple timeframes (5min, 15min, 30min, 1h) and a mean absolute error of 0.3-0.4% on normalized returns for multiple stocks.



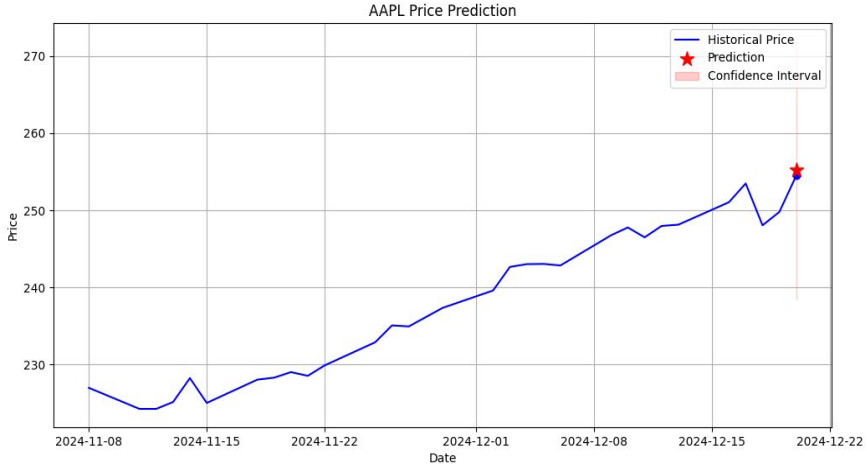

*LSTM-XGBoost Ensemble Architecture*

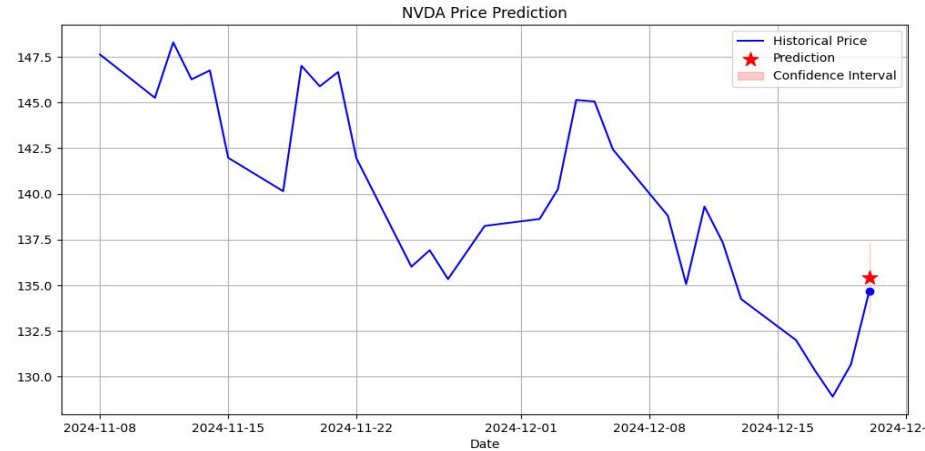
# LSTM-XGBoost Ensemble Model Prediction Results with Examples



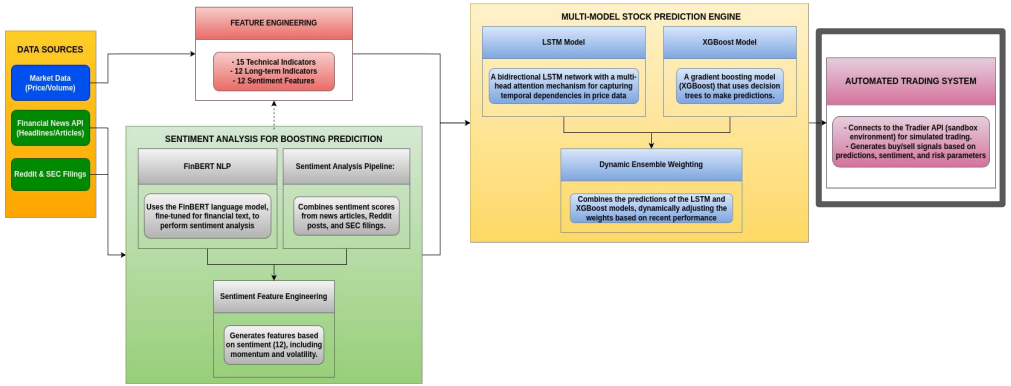

*Training Convergences*




*Price Predictions*

# Automated Trading System: Real-World Implementation




*System Architecture and Data Flow*

**Trading Strategy Integration:**

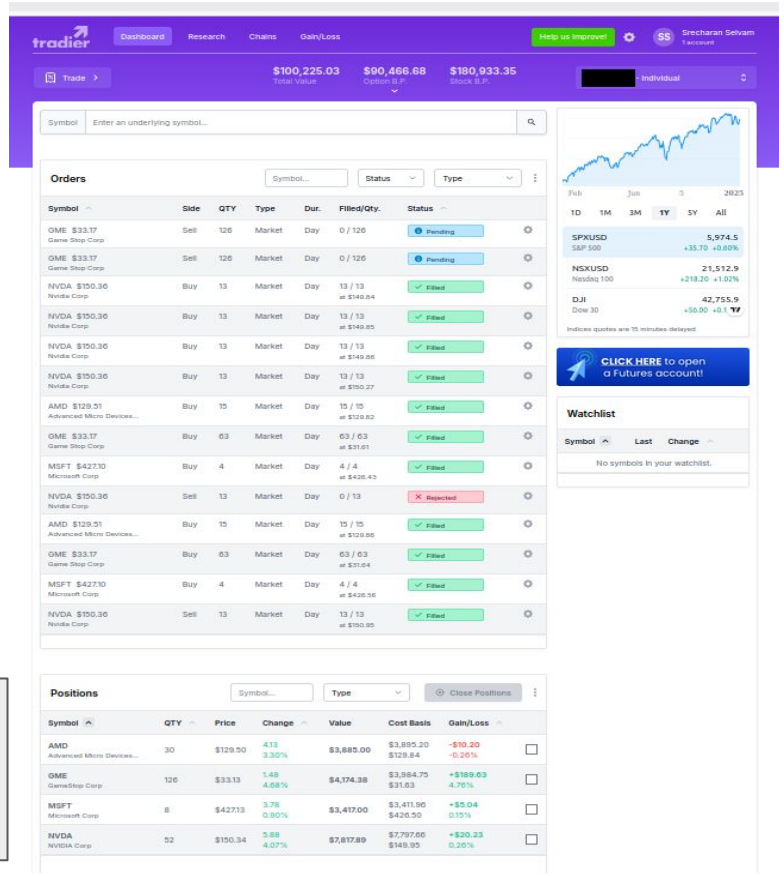
- Entry conditions: Positive directional accuracy in 5/15/30min timeframes.
- Exit conditions: Negative directional forecast

**Risk Management Framework:**

- Risk per trade: 1% of capital
- Stop-loss: 1.5%, Take-profit: 3%

**System Validation:**

- Win rate: 58.5% across 9 major stocks
- +0.32% net return




*Screenshot of the Tradier paper trading interface, showing successful execution of trades (01/06/2025)*

# Deep Image Synthesis with GANs, VAEs, and Diffusion Models

**Project Overview:**

A comprehensive exploration of generative image synthesis comparing three leading architectures on the challenging CUB-200-2011 bird dataset. This project implements multiple GAN variants (Vanilla, LSGAN, WGAN-GP) from scratch with custom ResBlock architectures, explores VAEs with β-annealing optimization, and investigates diffusion models through DDPM and DDIM sampling strategies. The implementation includes custom loss functions, stability improvements like gradient penalty and β-annealing, and rigorous evaluation through FID scoring. WGAN-GP emerged as the strongest performer with a 33.07 FID score, outperforming diffusion models (34.73) and demonstrating the effectiveness of adversarial training with proper regularization.

**GitHub Repository:** GenVision

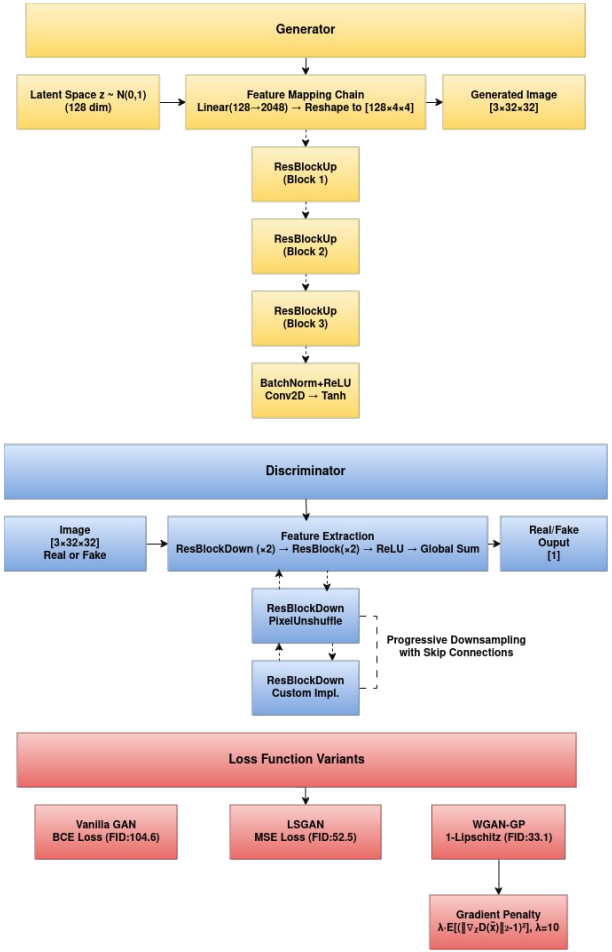
**Key Technologies and Skills Used:**

**Languages & Frameworks:** Python, PyTorch, TensorFlow, NumPy, OpenCV, scikit-learn, clean-fids, MLflow, Weights & Biases
**Deep Learning:** Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), Diffusion Models (DDPM, DDIM), Model Training, Hyperparameter Tuning, Loss Functions (Adversarial Loss, Reconstruction Loss, KL Divergence, Gradient Penalty)
**Computer Vision:** Image Synthesis, Image Generation, Feature Visualization

# Generative Adversarial Networks (GANs) Architecture

**Generator**:

- Input: 128-dimensional noise vector (z ~ N(0,1))
- Architecture: Linear projection → Reshape to [128×4×4] → Series of ResBlockUp modules with BatchNorm and ReLU → Final Conv2D with Tanh
- ResBlockUp: Improves gradient flow and increases spatial resolution progressively
- Output: Generated RGB image [3×32×32]

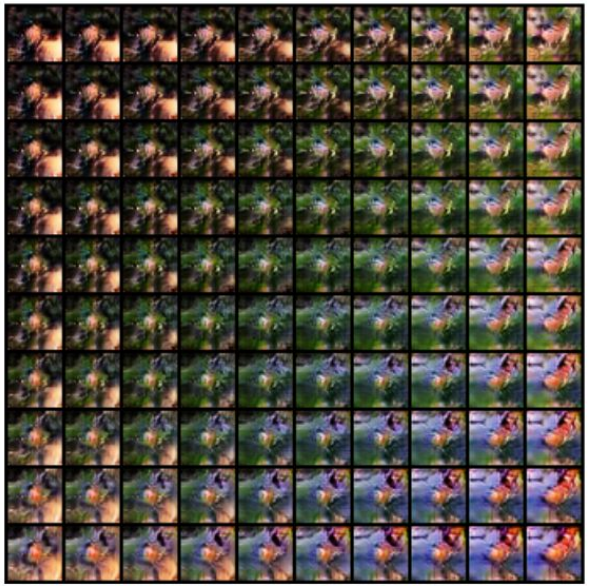



**Discriminator**:

- Input: RGB image [3×32×32]
- Architecture:ResBlockDown modules→ Standard ResBlocks→ReLU→Global Sum Pooling
- ResBlockDown: Feature extraction with downsampling (using DownSampleConv2D)
- Output: Scalar value indicating image authenticity (probability or Wasserstein score)

# Types of GANs - Vanilla GAN

*Vanilla GAN Samples*

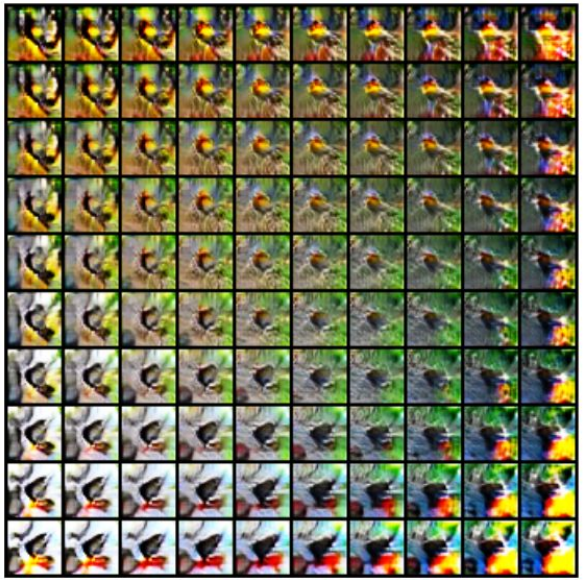


*Vanilla GAN Latent Space Interpolations*

| GAN Variant | Loss Function | Key Features | FID Score |
|---|---|---|---|
| Vanilla GAN | Binary Cross-Entropy (BCE) | Original GAN formulation. Prone to training instability (mode collapse). | 104.62 |

# Types of GANs - LS-GAN

*LS-GAN Samples*

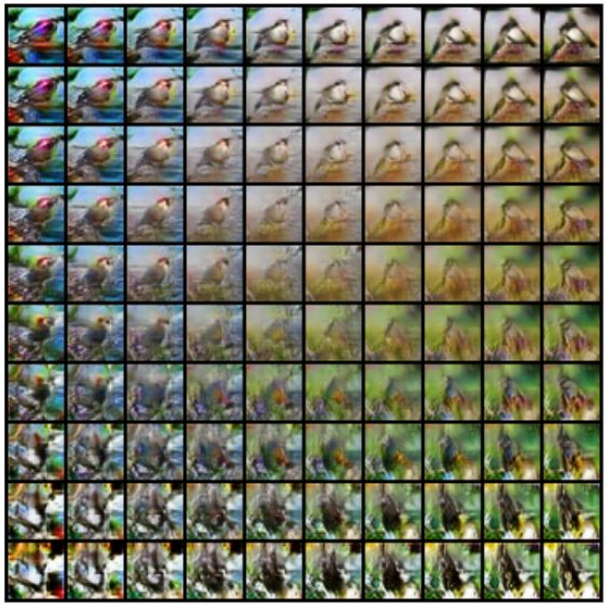


*LS-GAN Latent Space Interpolations*

| GAN Variant | Loss Function | Key Features | FID Score |
|---|---|---|---|
| LSGAN | Least Squares Loss (MSE) | Uses Mean Squared Error instead of BCE. More stable training than Vanilla GAN. | 52.48 |

# Types of GANs - WGAN-GP

*WGAN-GP Samples*



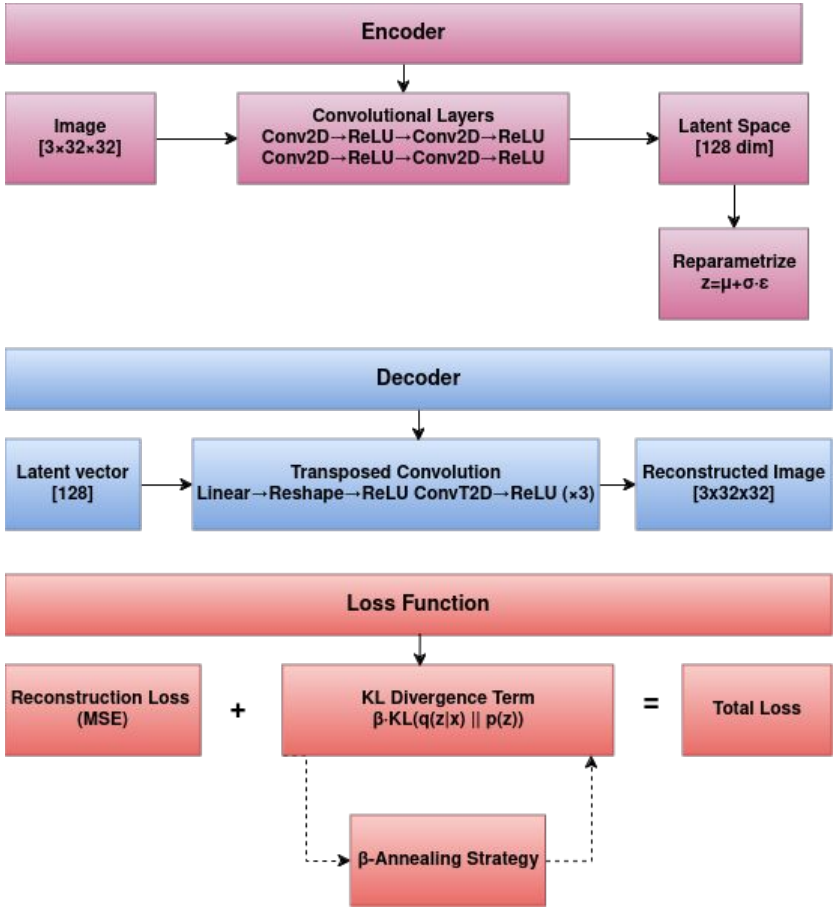
*WGAN-GP Latent Space Interpolations*

| GAN Variant | Loss Function | Key Features | FID Score |
|---|---|---|---|
| WGAN-GP | Wasserstein Distance + Gradient Penalty (GP) | Uses Wasserstein distance for a more stable measure of the difference between distributions. Gradient Penalty enforces the 1-Lipschitz constraint. | 33.07 |

# Variational Autoencoder (VAEs) Architecture

**Encoder**:

- Input: RGB image [3×32×32]
- Architecture: Series of Conv2D→ReLU layers with progressive spatial dimension reduction
- Output: Mean vector (μ) and log standard deviation vector (log σ) of dimension 128
- Reparameterization trick: z = μ + σ * ε where ε ~ N(0,1)



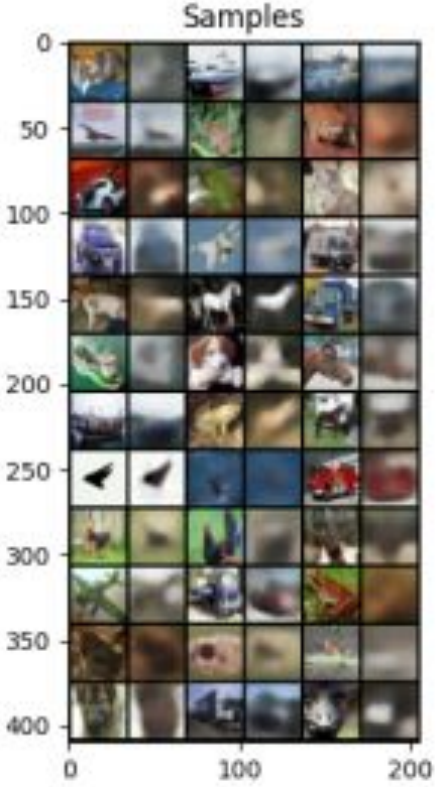

**Decoder**:

- Input: 128-dimensional latent vector (z)
- Architecture: Linear projection → Reshape → Series of transposed convolutions with ReLU
- Output: Reconstructed RGB image [3×32×32]

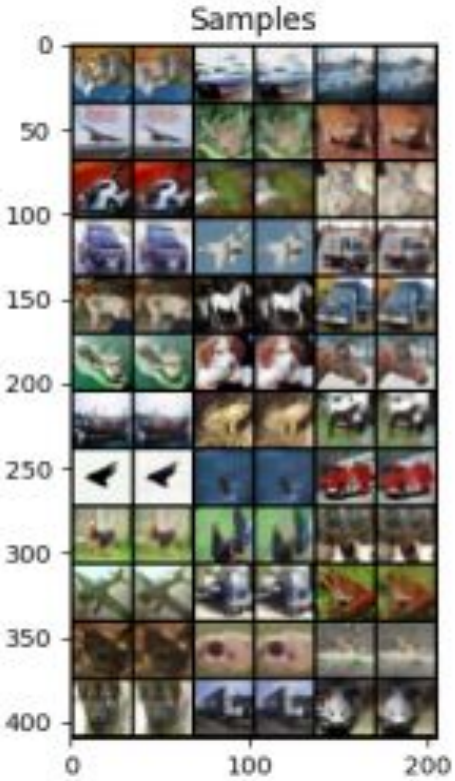
**Loss Function**:

- Reconstruction Loss: MSE between input and reconstructed image
- KL Divergence: Regularizes latent space toward standard normal distribution
- β-Annealing: Gradually increases β during training for better latent organization

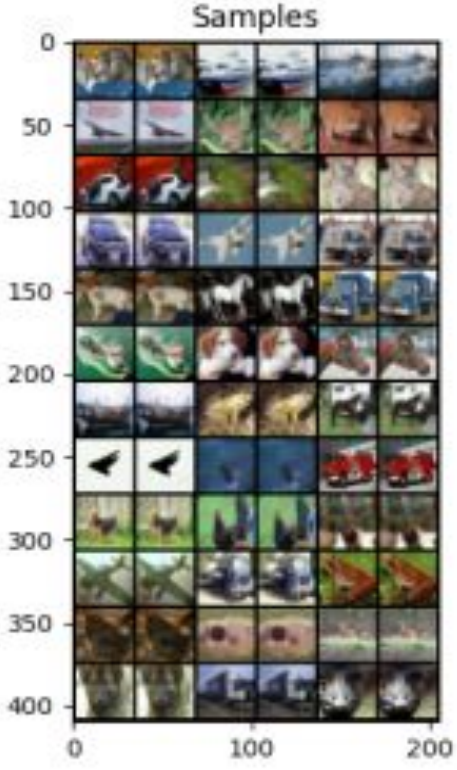
# VAE Latent Space Exploration




*Reconstruction sample: (size 16)*




*Reconstruction sample: (size 128)*

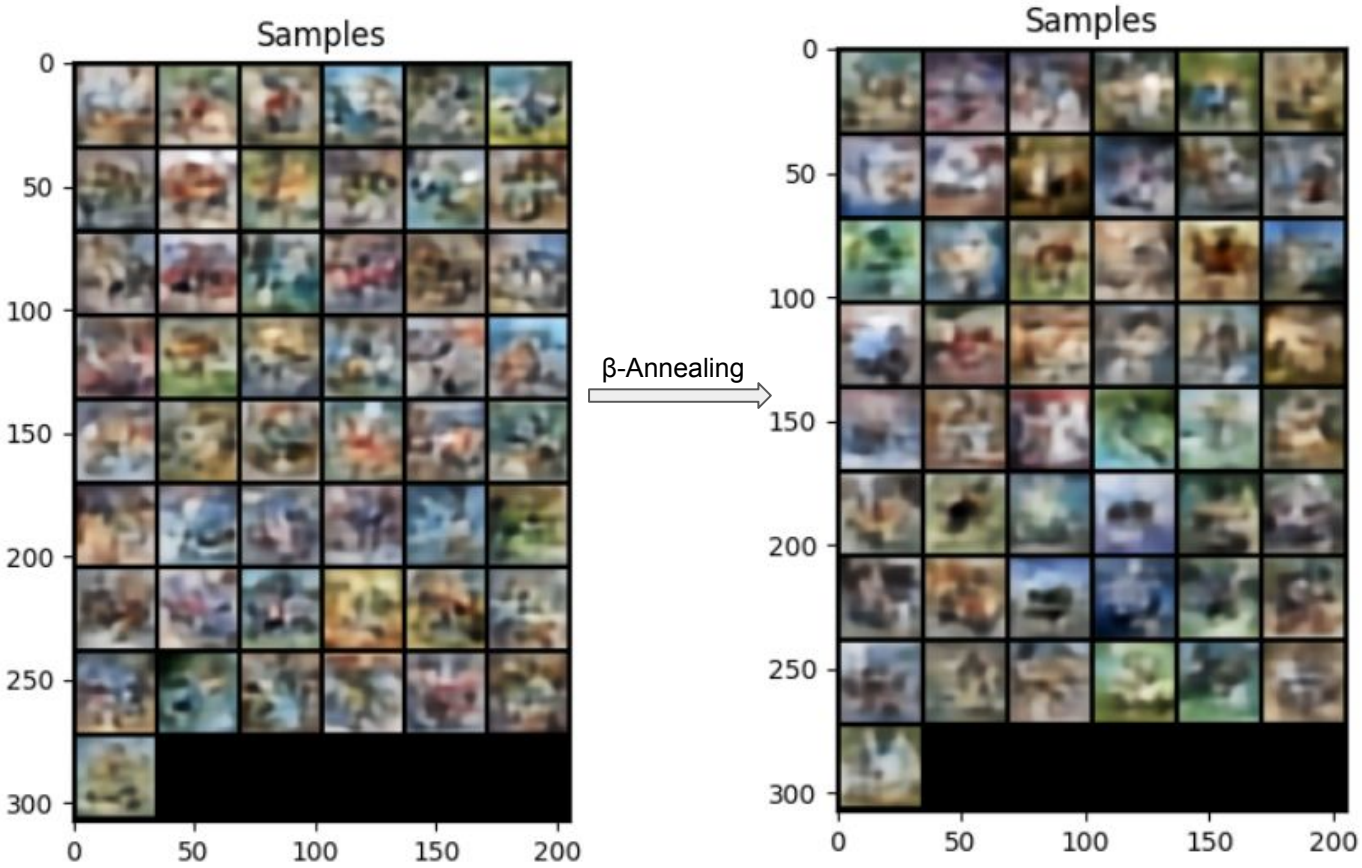



*Reconstruction sample: (size 1024)*

| | | |
|---|---|---|
| Reconstruction loss: ~200 | Reconstruction loss: ~200 → ~75 | Reconstruction loss: ~200 → ~75 → ~40 |

# VAE Sample Generation and β-Annealing

To further improve the VAE's performance, we investigated the use of β-annealing. β controls the weight of the KL divergence term in the loss function. By gradually increasing β during training, we encourage the model to learn a more disentangled and well-structured latent space, which often leads to better sample quality.

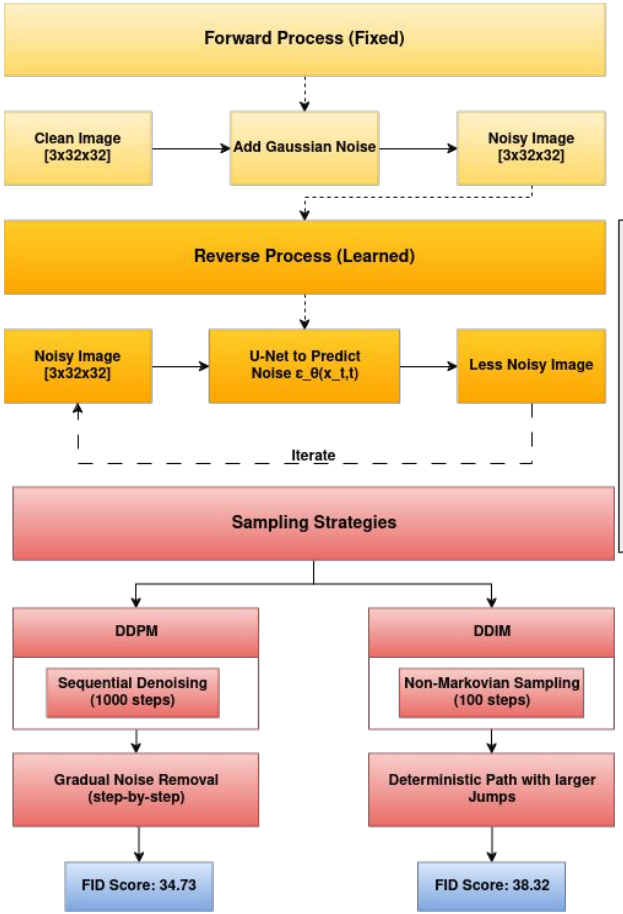



*The left image shows samples generated with a fixed β value of 0.8. The right image shows samples generated after training with β-annealing (linearly increasing β from 0 to 0.8 over the first 20 epochs)*

# Diffusion Model Architecture with DDPM & DDIM Sampling

**Forward Process (Fixed)**:

- Input: Clean RGB image [3×32×32]
- Process: Sequential addition of Gaussian noise over T timesteps (not learned)
- Output: Pure Gaussian noise after T steps




**Reverse Process (Learned)**:

- Input: Noisy image at timestep t
- Architecture: U-Net predicts noise component ε_θ (x_t, t)
- Process: Iterative denoising by subtracting predicted noise
- Output: Progressively cleaner image, approaching the original distribution

**Sampling Strategies**:

- DDPM: Markovian process requiring ~1000 sequential denoising steps
- DDIM: Non-Markovian approach allowing larger jumps with only ~100 steps
- Trade-off: DDIM sacrifices some quality for 10× faster sampling

# Diffusion Model Results

*DDPM Samples*

*DDIM Samples*

DDPM (left) achieves slightly better FID score (34.73) with 1000 sampling steps, while DDIM (right) maintains comparable quality (FID 38.32) with only 100 steps, demonstrating a significant efficiency improvement with minimal quality loss.

# Thank You!

sselvam@andrew.cmu.edu

linkedin.com/srecharan

srecharan.com

GitHub