Portfolio

Srecharan Selvam

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

Research Experience

Vision-Language-Action Enhanced Robotic Leaf Grasping: A Hybrid Foundation Model Approach

Project Overview:

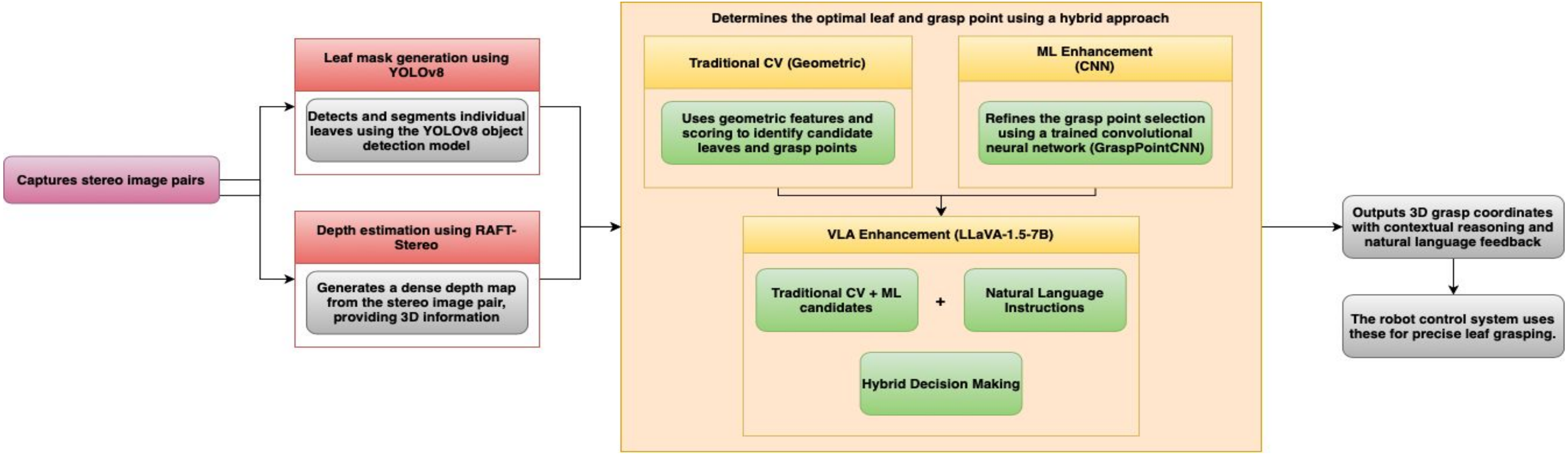
A real-time vision-language-action system for autonomous robotic leaf manipulation combining foundation models with traditional computer vision and deep learning. This hybrid system integrates YOLOv8 for leaf segmentation, RAFT-Stereo for depth estimation, LLaVA-1.5-7B for contextual reasoning, and a 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, ML predictions, and VLA reasoning to achieve superior contextual grasping performance.

GitHub Repository: LeafGrasp-Vision-ML

Key Technologies and Skills Used:

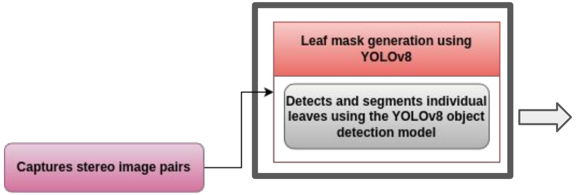
Languages: Python, C++
Frameworks: PyTorch, Transformers, CUDA, OpenCV, Scikit-learn, Numpy, Pandas, Matplotlib, ROS2, MLflow
Foundation Models: LLaVA-1.5-7B, LoRA Fine-tuning, Vision-Language-Action Integration, Multimodal Reasoning
Computer Vision: Instance Segmentation, Depth Estimation, Point Cloud Processing, SDF, 3D Perception
Deep Learning: CNN Architecture Design, Self-Supervised Learning, Model Training & Optimization, Attention Mechanisms
Production Optimization: TensorRT, Custom CUDA Kernels, Docker Containerization, AWS EC2

Pipeline:

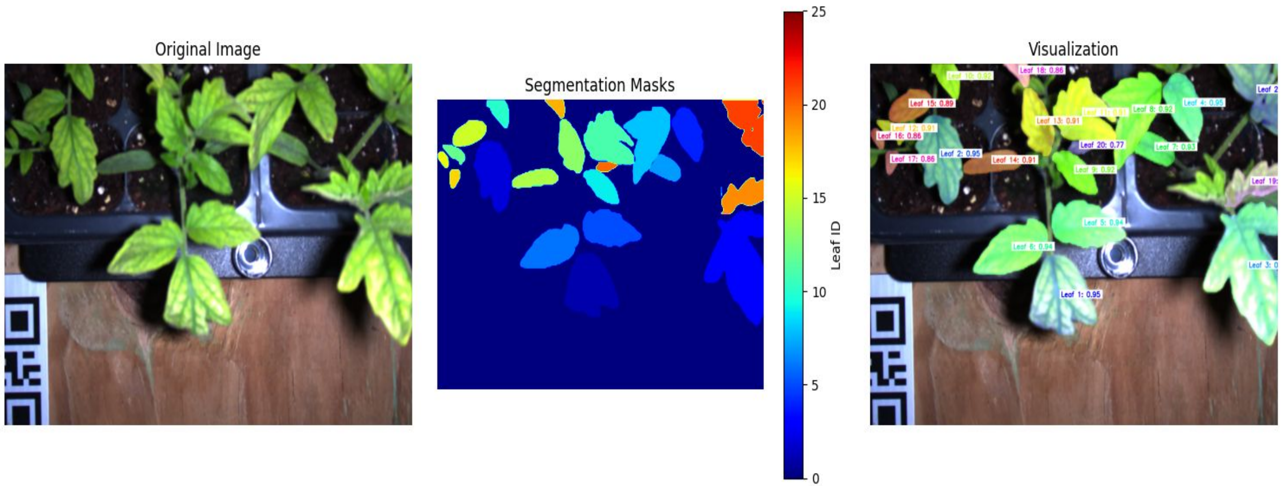


High-Level System view

YOLOv8 Leaf Instance Segmentation



High-Level System view

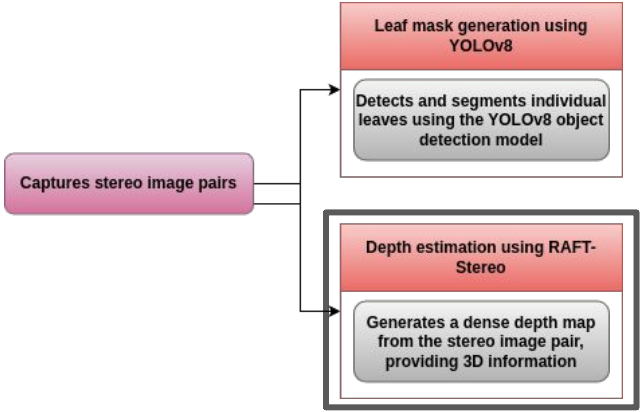


Output from YOLOv8 pipeline: Left: Original RGB image showing tomato plant; Center: Generated segmentation masks; Right: visualization with leaf IDs and confidence scores

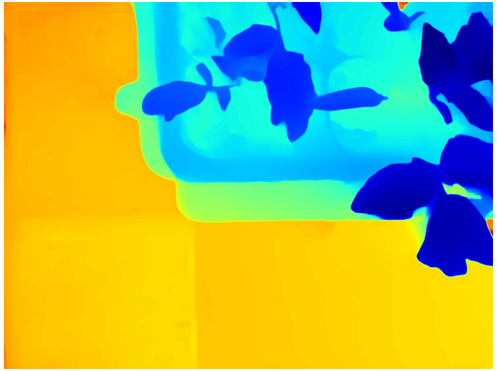
Key Details:

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

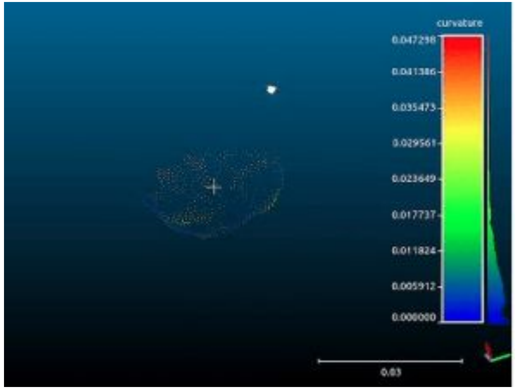
High-Precision Depth Estimation with RAFT-Stereo



High-Level System view



Depth Map



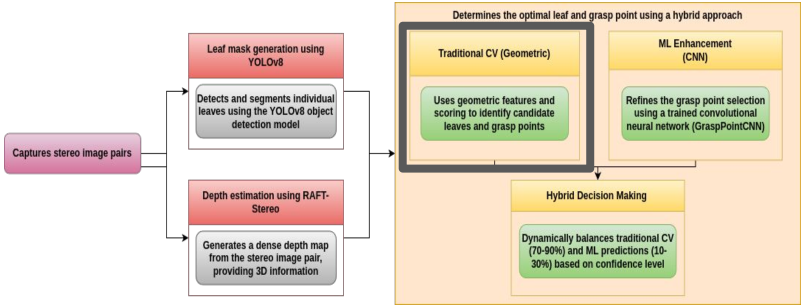
Geometric capture of leaf surface

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

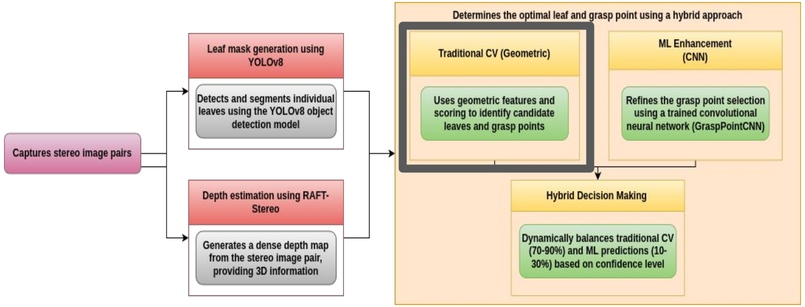
Traditional CV Pipeline: Geometric Grasp Point Selection



High-Level System view

Original camera view with leaf midrib

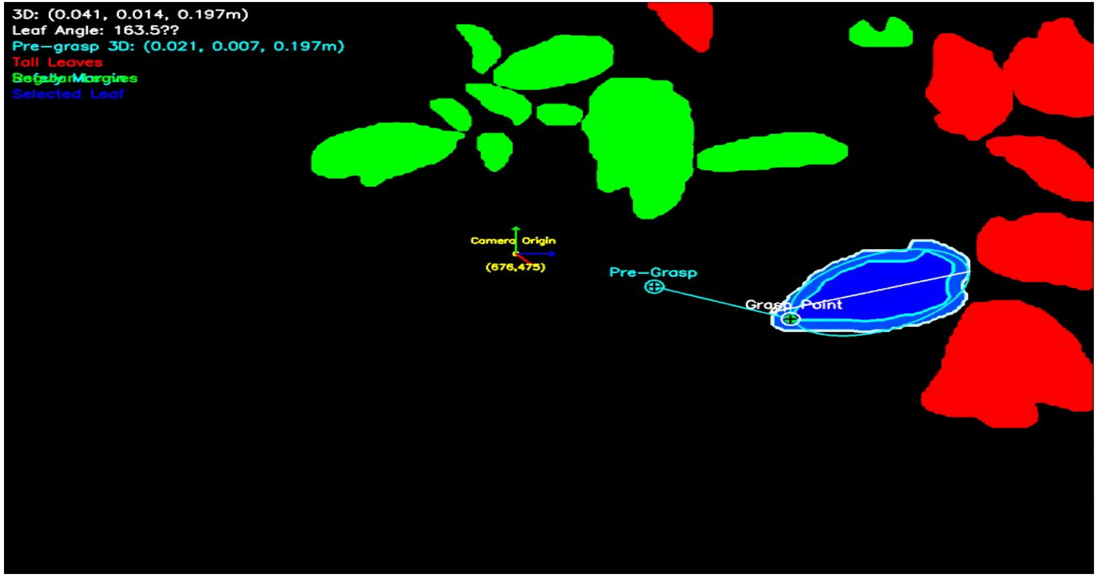
Traditional CV Pipeline: Geometric Grasp Point Selection



High-Level System view

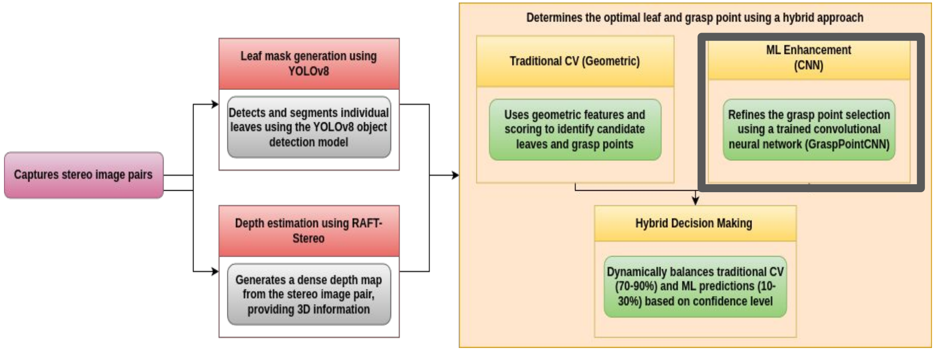
Optimal Leaf Selection (Pareto Optimization):
- Clutter Score (35%): Leaves that are isolated
- Distance Score (35%): Leaves that are closer to camera
- Visibility Score (30%): Leaves that are fully visible

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

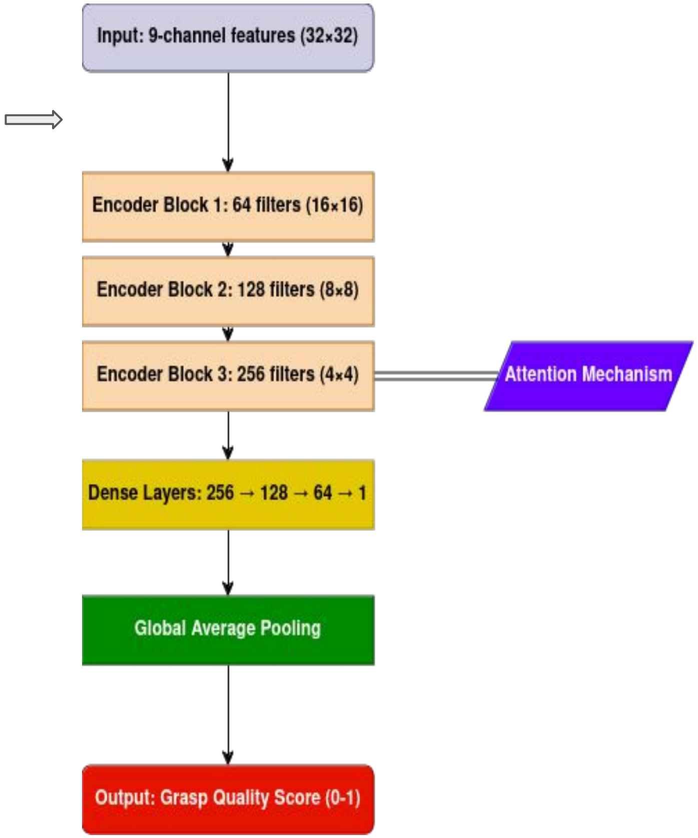


Segmented leaves with optimal leaf & grasp point visualization

GraspPointCNN: ML-Based Grasp Refinement



High Level System view



GraspPointCNN Architecture

Self-Supervised Learning Pipeline:

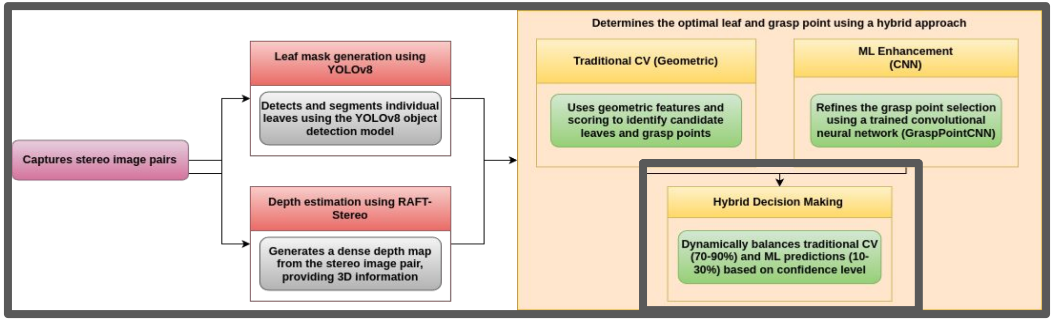
- Input: 9-channel features (32×32)

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

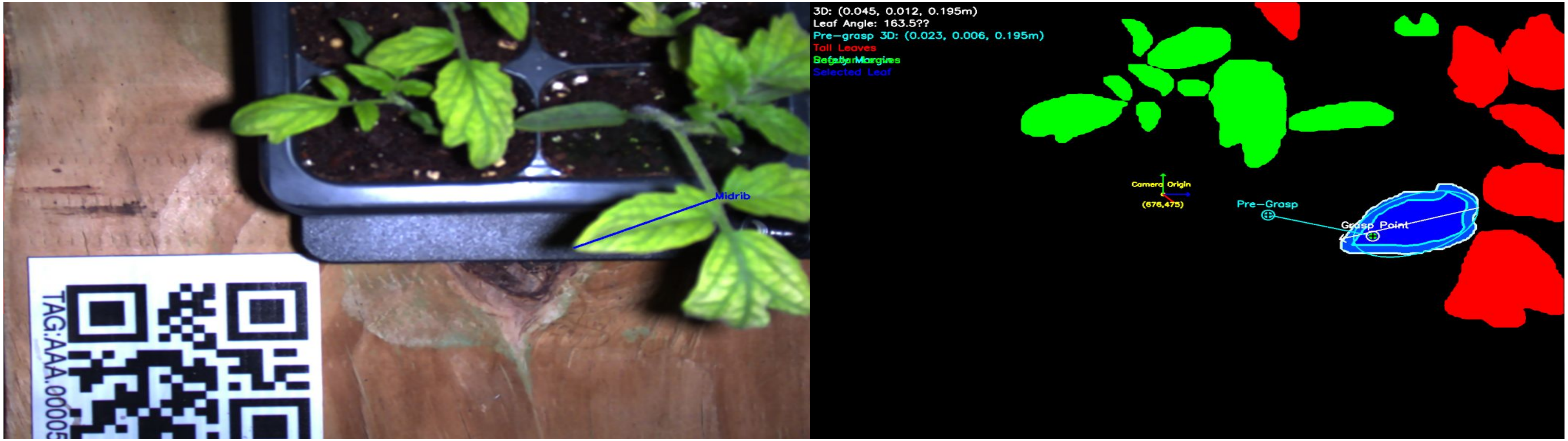
- Training Configuration:

- 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

Hybrid Decision Integration: Combining CV & ML

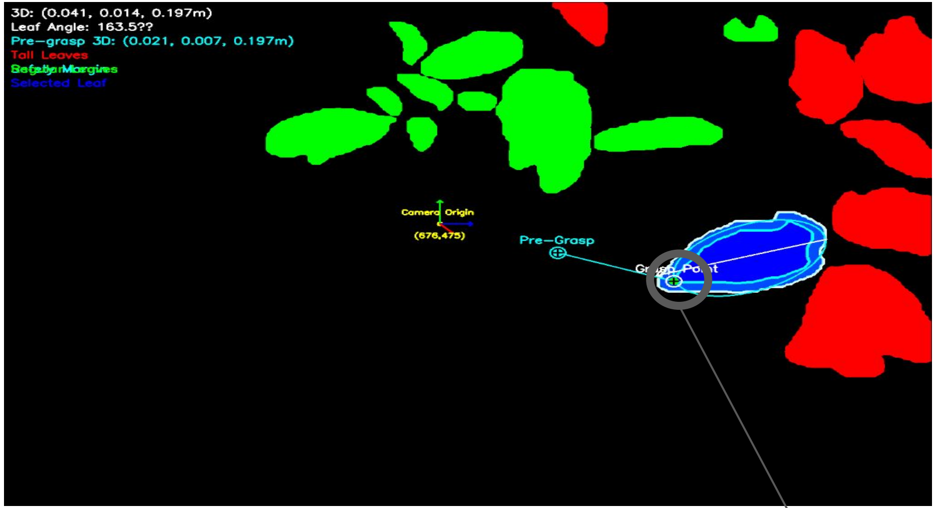


High-Level System view

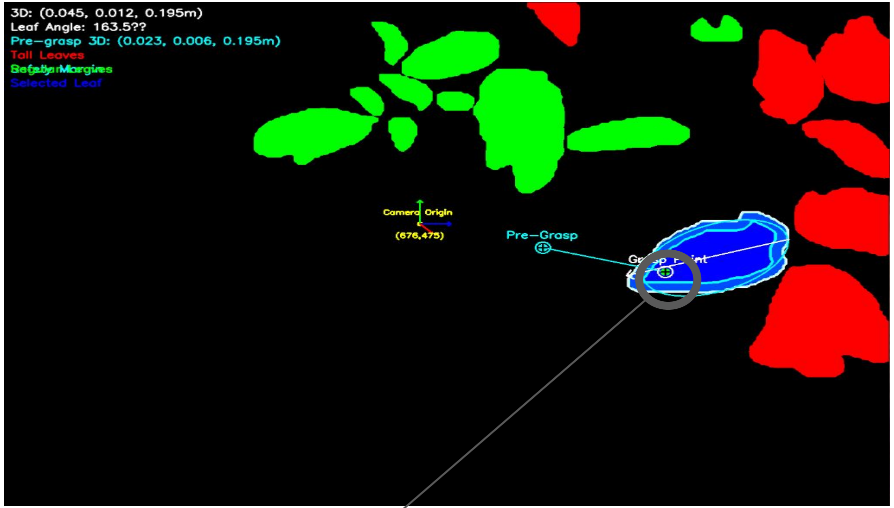


Hybrid CV-ML grasp point selection: Left - Original camera view with leaf midrib ; Right - Segmented leaves with grasp point visualization

Traditional CV vs Hybrid Decision System



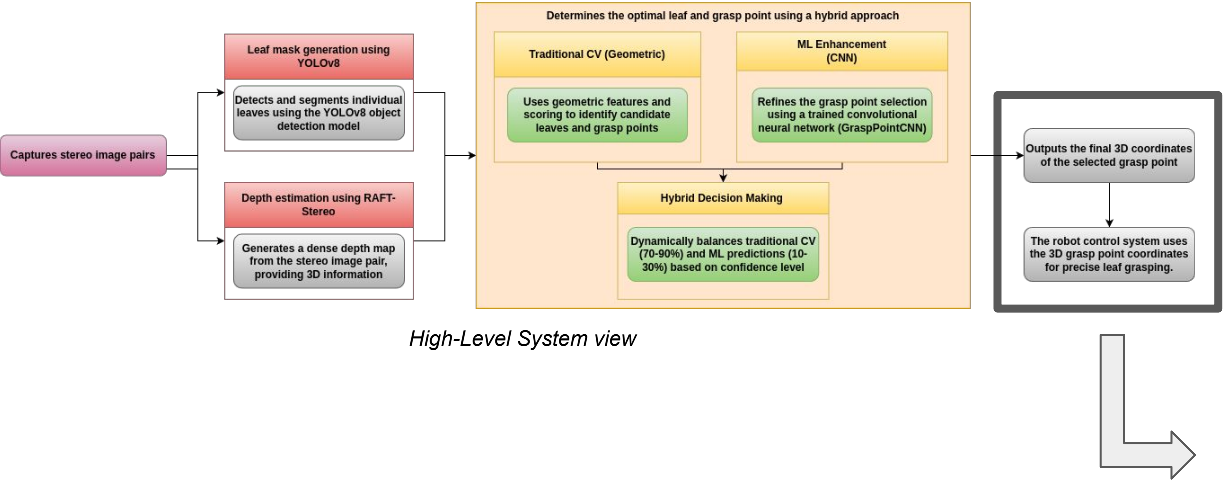
Traditional CV grasp point selection



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.

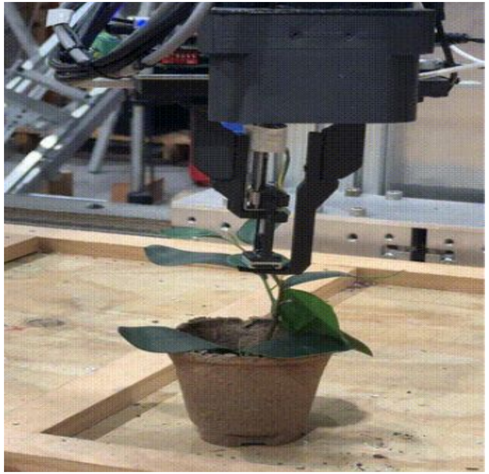
REX Robot Integration for Leaf Grasping



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:

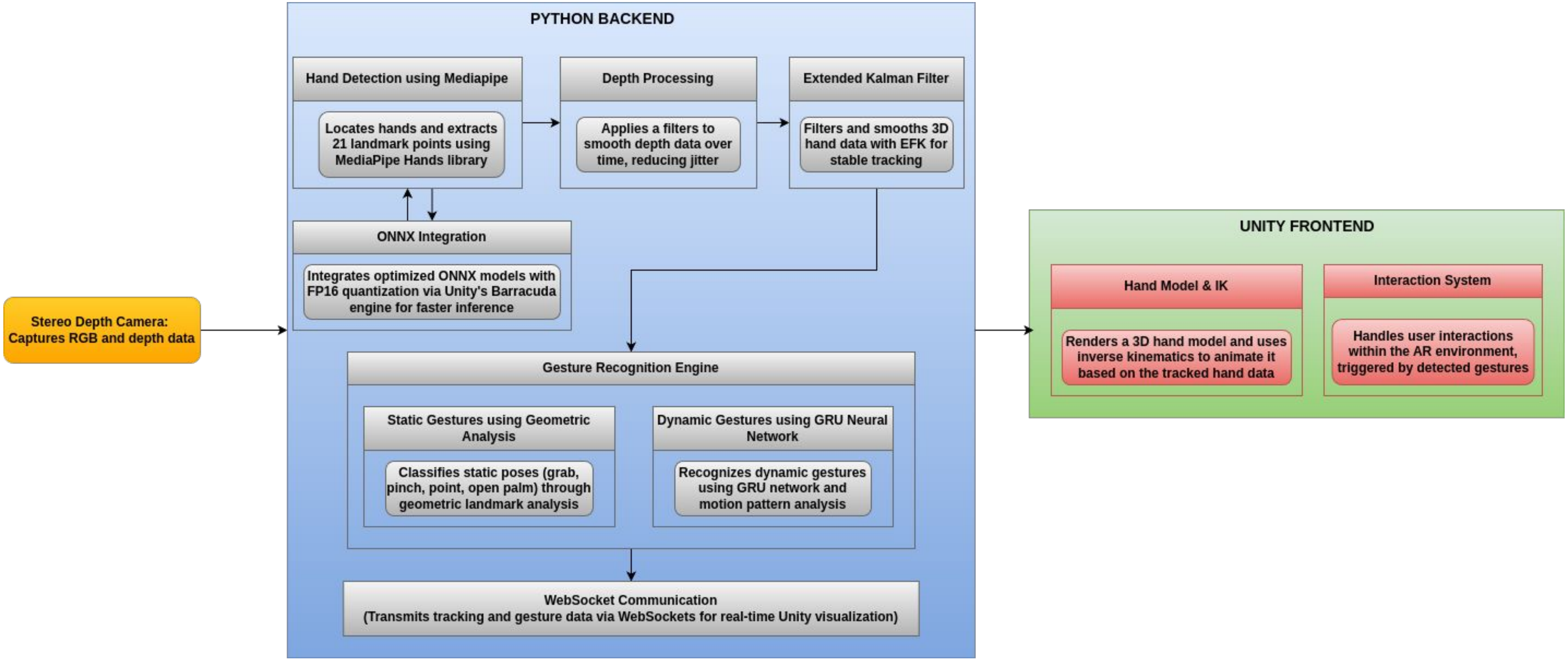
A comprehensive hand tracking and gesture recognition system built for augmented reality applications in automotive training. This system combines advanced computer vision techniques with deep learning to enable intuitive interaction with virtual HVAC components. The architecture integrates Extended 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:

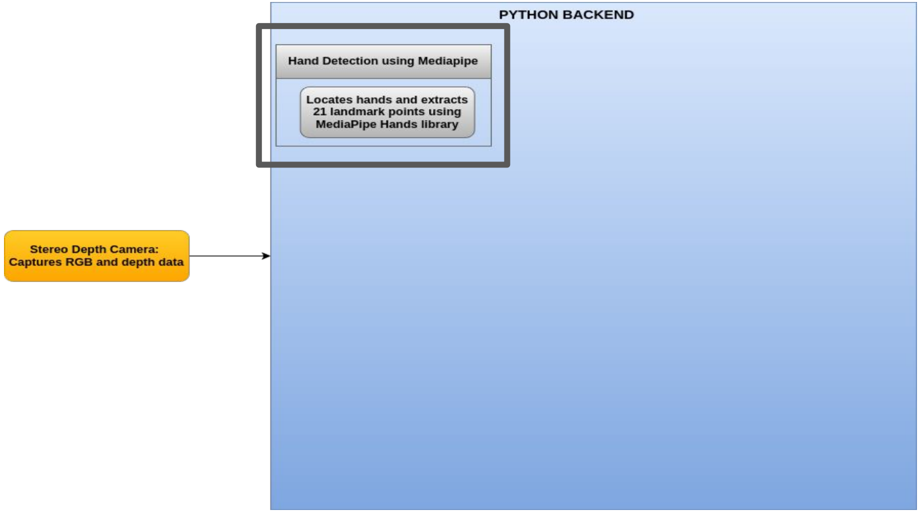
Languages & Frameworks: Python, PyTorch, ONNX, scikit-learn, MediaPipe, Websockets, OpenCV, NumPy
Machine Learning: 3D Tracking, Landmark Detection, Depth Sensing, Geometric Analysis, Kalman Filtering
Deep Learning: Recurrent Neural Networks (GRU), Model Optimization (ONNX), Inference (Unity Barracuda)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

Pipeline:

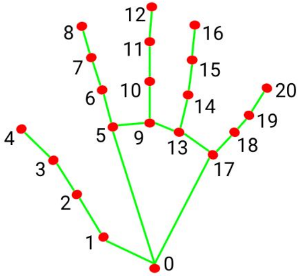


System Architecture and Data Flow

Hand Detection and Landmark Extraction with MediaPipe



System Architecture and Data Flow

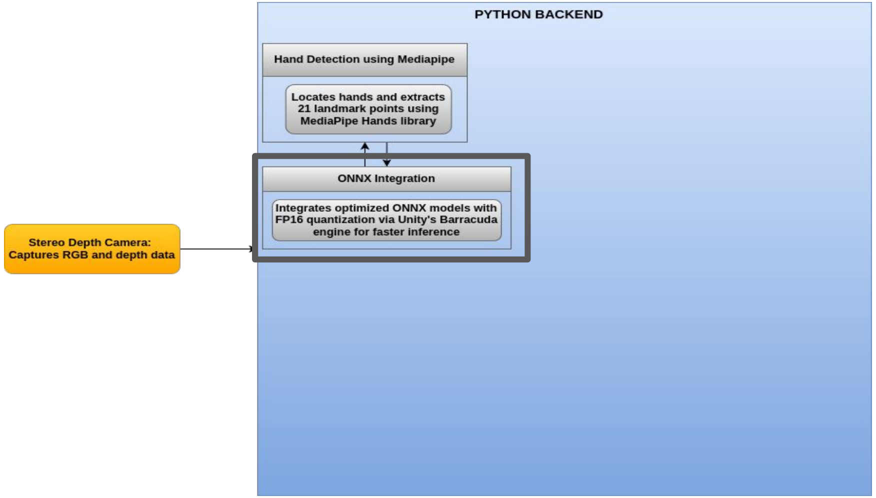


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP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

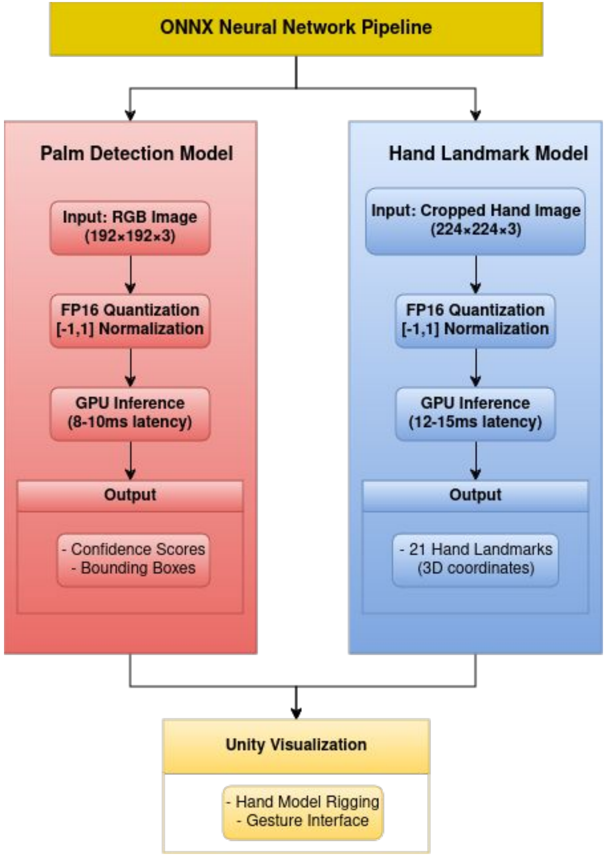
Uses Google's MediaPipe Hands library to detect hand presence and location in the RGB image

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

Neural Network Optimization with ONNX



System Architecture and Data Flow

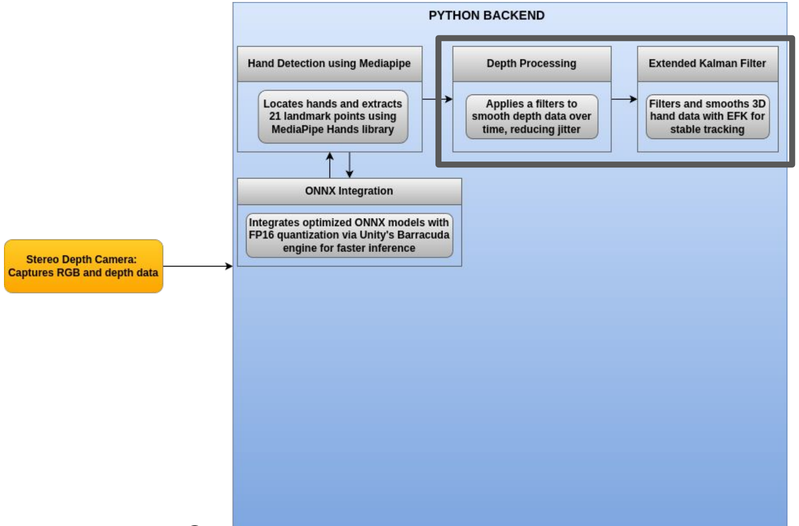


Two-stage ONNX pipeline

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

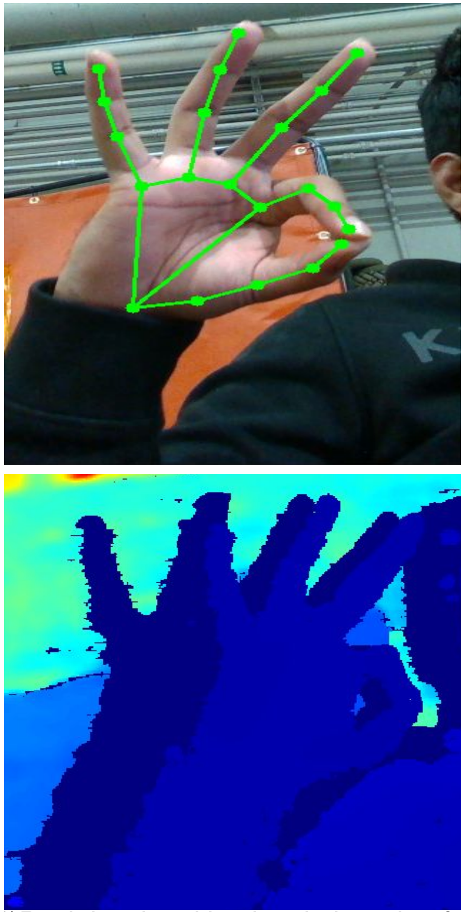


System Architecture and Data Flow

- 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

(1)

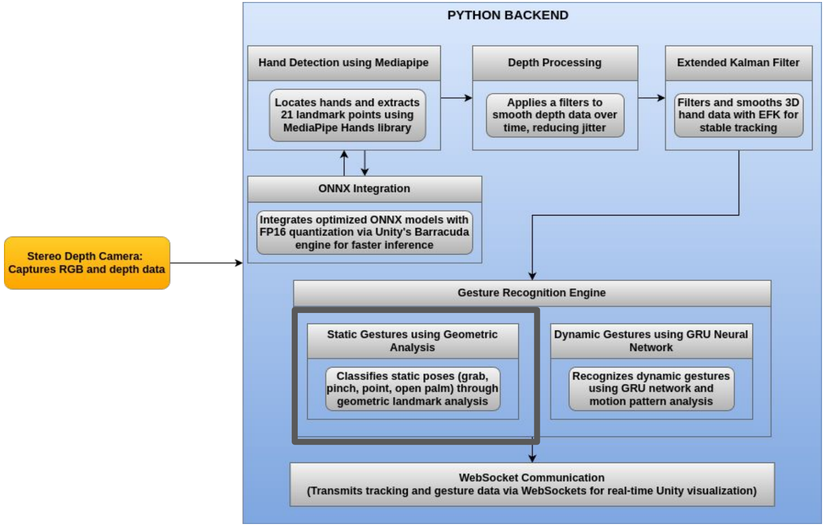
(2)



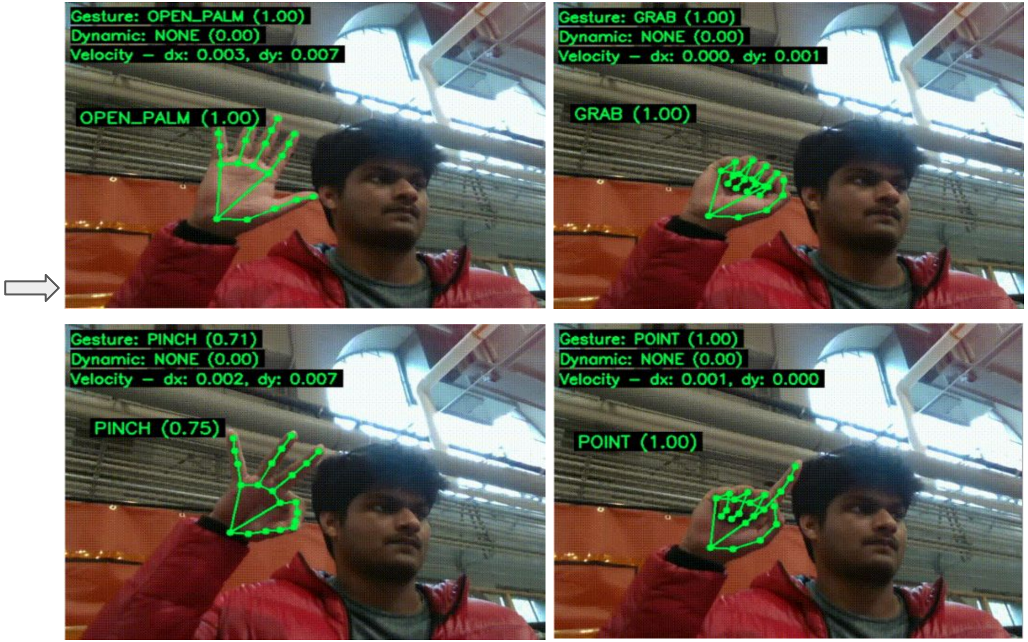
(1)Real-time hand landmark detection &
(2) Visualization of depth map using Filter

Multi-stage depth filtering for 3D hand tracking

Static Gesture Recognition



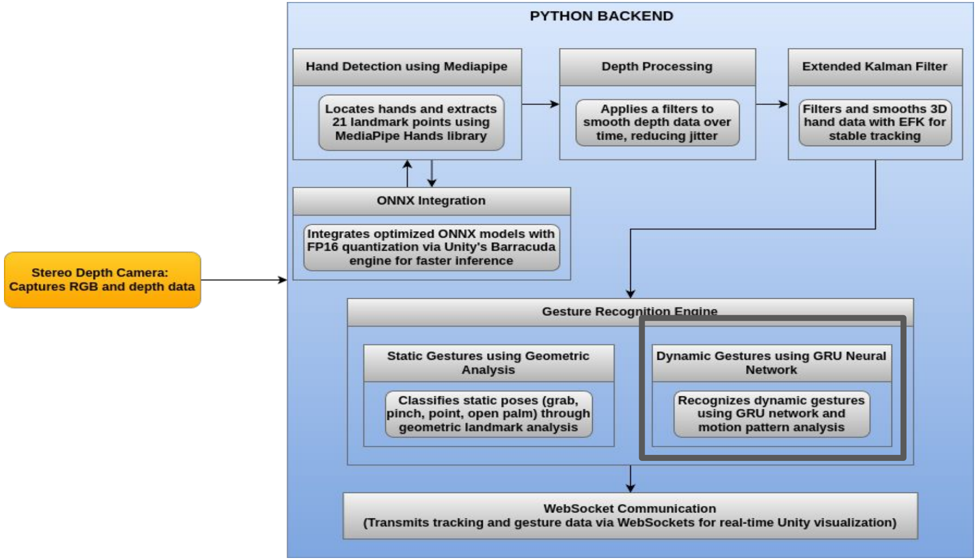
System Architecture and Data Flow



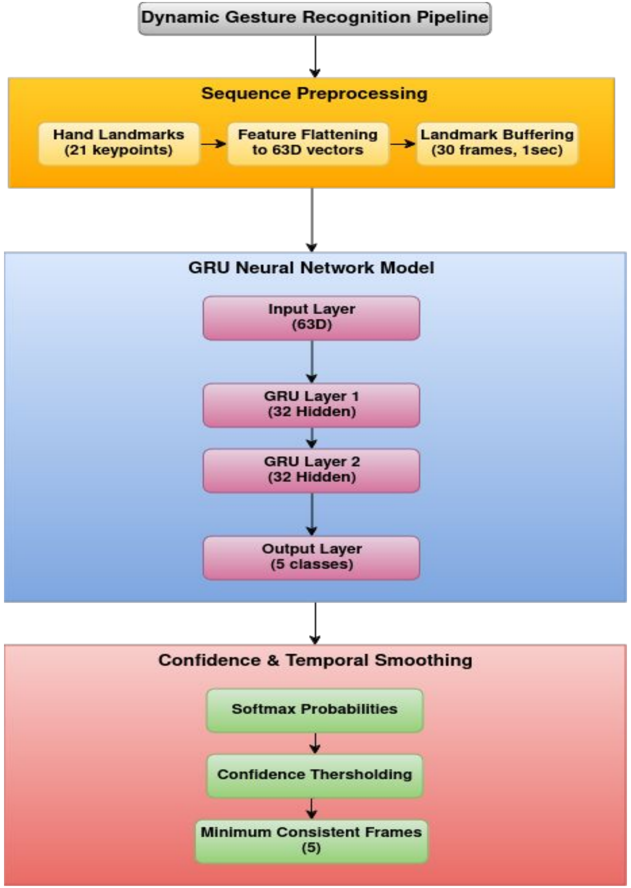
Real-time detection of OPEN_PALM, GRAB, PINCH, and POINT gestures

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



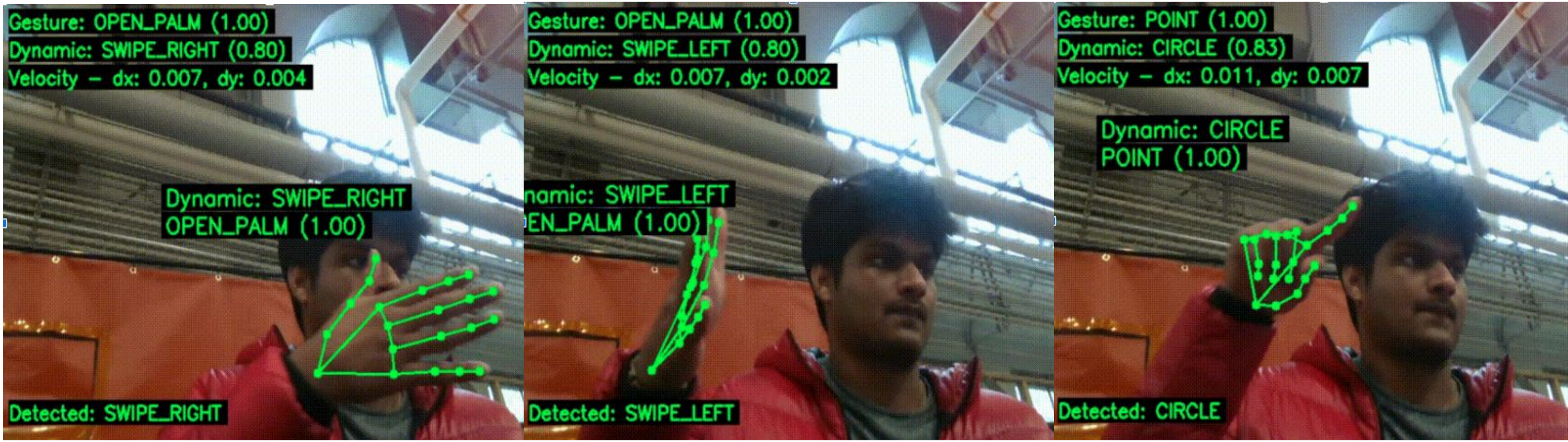
System Architecture and Data Flow



GRU Architecture Diagram

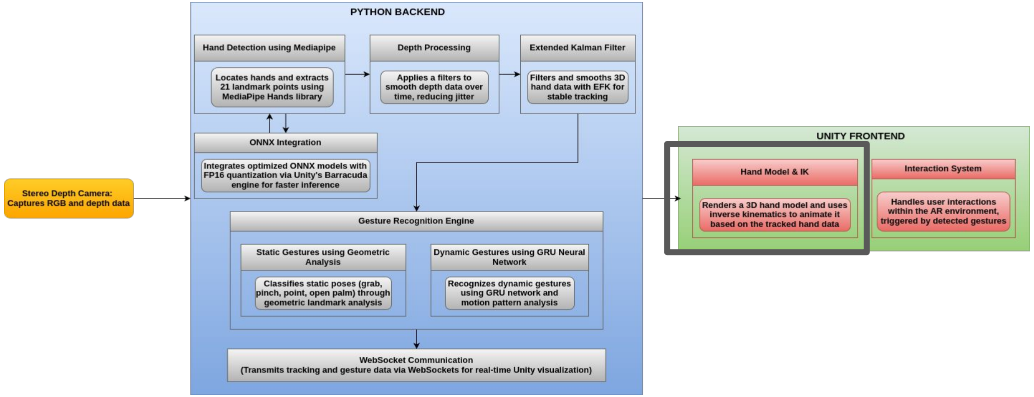
- A GRU network is a type of recurrent neural network (RNN) that is well-suited for processing sequential data, such as the time series of hand landmark positions.
- The GRU model was trained on a custom dataset of dynamic gestures.
- Input: The model takes a sequence of 30 frames of hand landmark data (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.

Dynamic Gesture Recognition

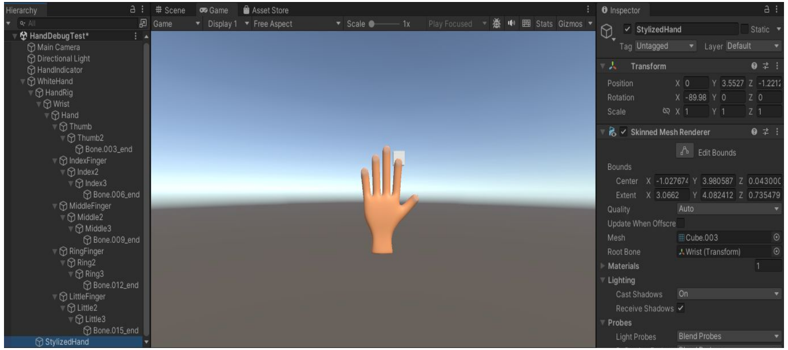


Real-time detection of dynamic gestures (SWIPE_RIGHT, SWIPE_LEFT, CIRCLE) using the GRU model and motion pattern analysis

Unity Frontend: Hand Rigging and Interaction



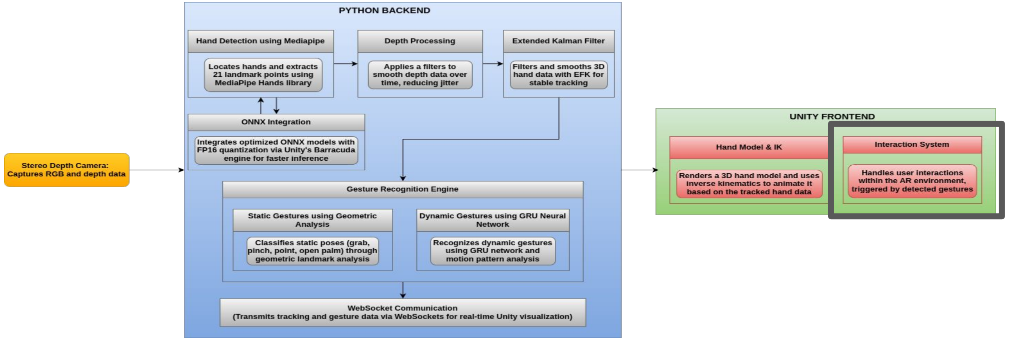
System Architecture and Data Flow



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:

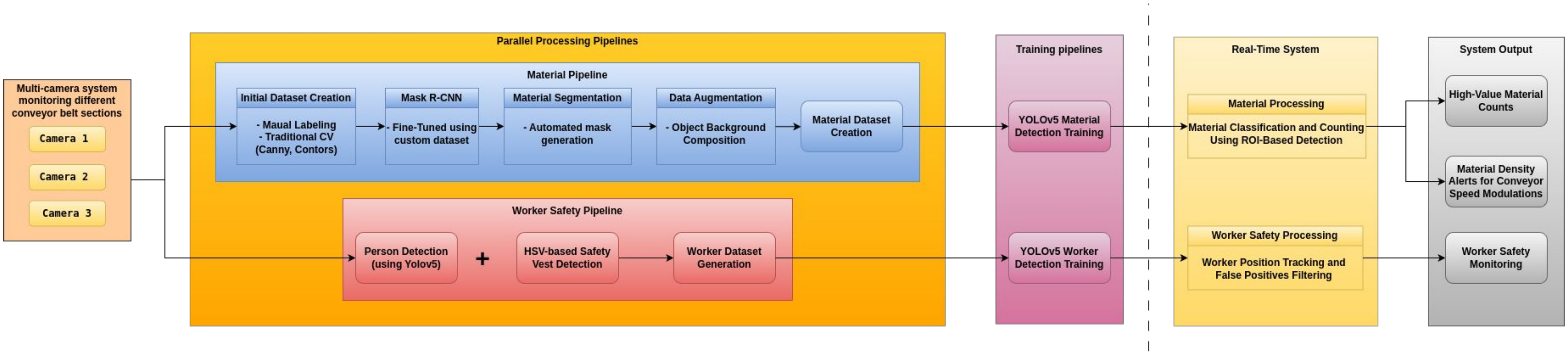
A real-time computer vision system for enhancing high-value material recovery and worker safety monitoring on industrial conveyor belts. This system combines YOLOv5 for precise material detection and segmentation with intelligent background subtraction for motion analysis. The architecture features camera-specific 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:

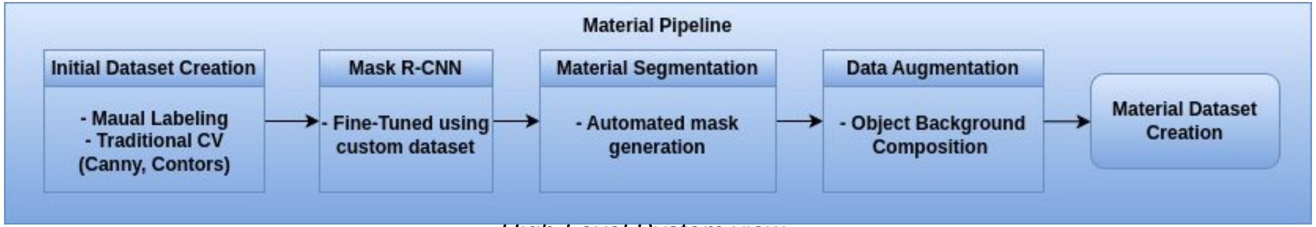
Languages: Python, C++
Frameworks: PyTorch, OpenCV, scikit-learn, NumPy, Pandas
Machine Learning: Model Training, Fine-tuning, Data Augmentation, Model Evaluation (mAP, Precision, Recall)
Computer Vision: Object Detection (YOLOv5), Instance Segmentation (Mask R-CNN), Multi-Camera Systems, Region of Interest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

Pipeline



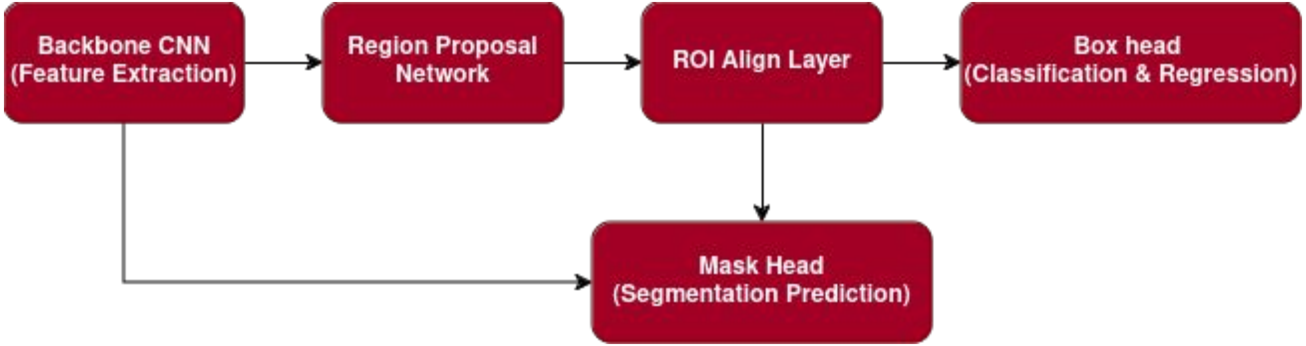
End-to-end pipeline for automated material sorting and worker safety, with parallel processing for offline training and real-time operation

Semi-Automated Data Annotation and Augmentation



High-Level System view

Initial Dataset Creation using labelme

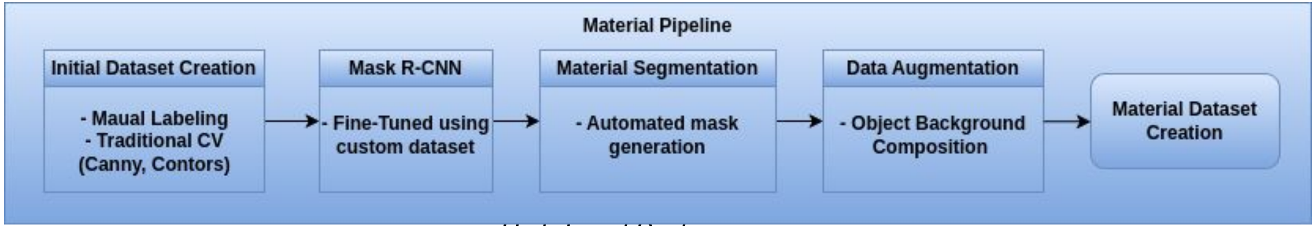


Simple Mask R-CNN Architecture Diagram

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.

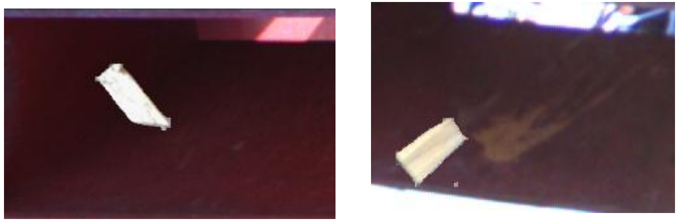
Data Augmentation and Dataset Creation



High-Level System view

Initial Dataset Creation using labelme

Automated Masked & Segmented images

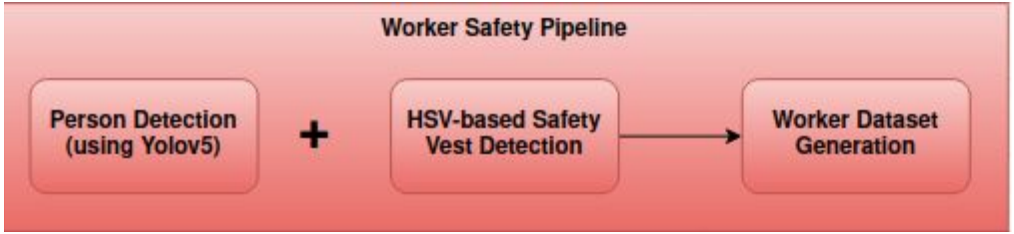


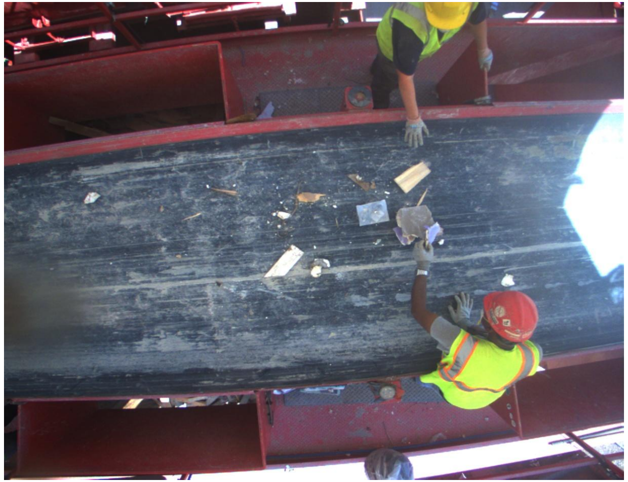
Data Augmentation (Object-Background Compositing)

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.

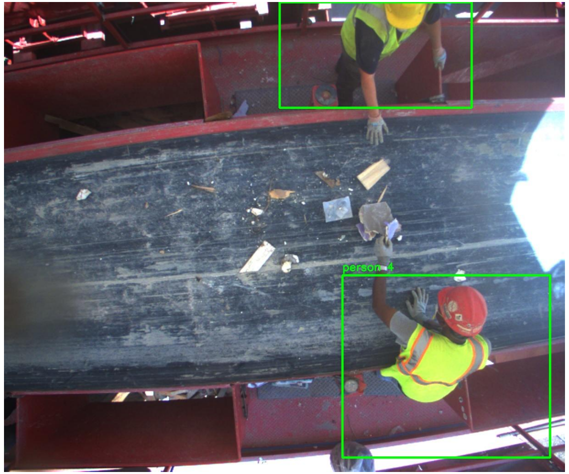
Data Augmentation: To create a robust and diverse training dataset, a sophisticated data augmentation strategy was employed. Segmented material images (generated by the Mask R-CNN model) were overlaid onto images of the empty conveyor belt bins

Worker Detection Dataset Creation





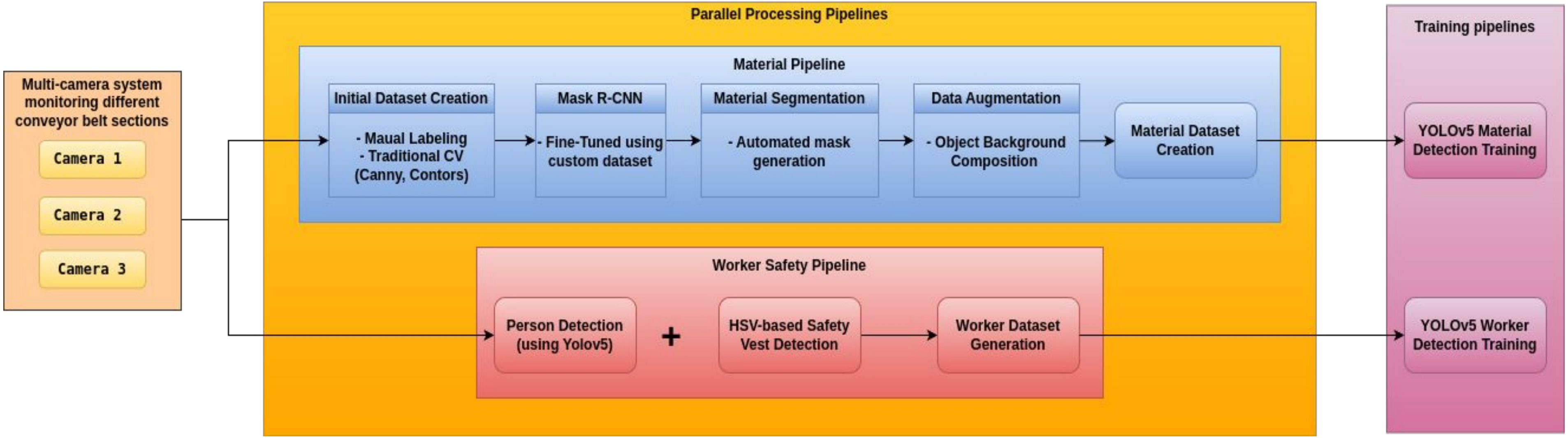
Raw image from the camera



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.

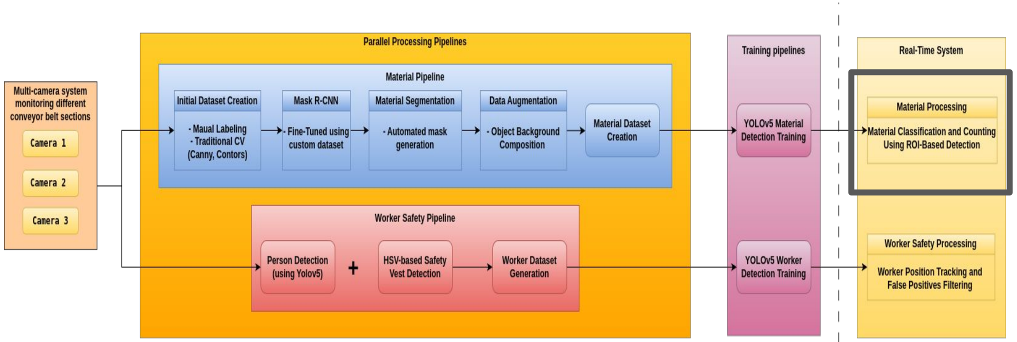
Detection Model Training and Performance



Performance:

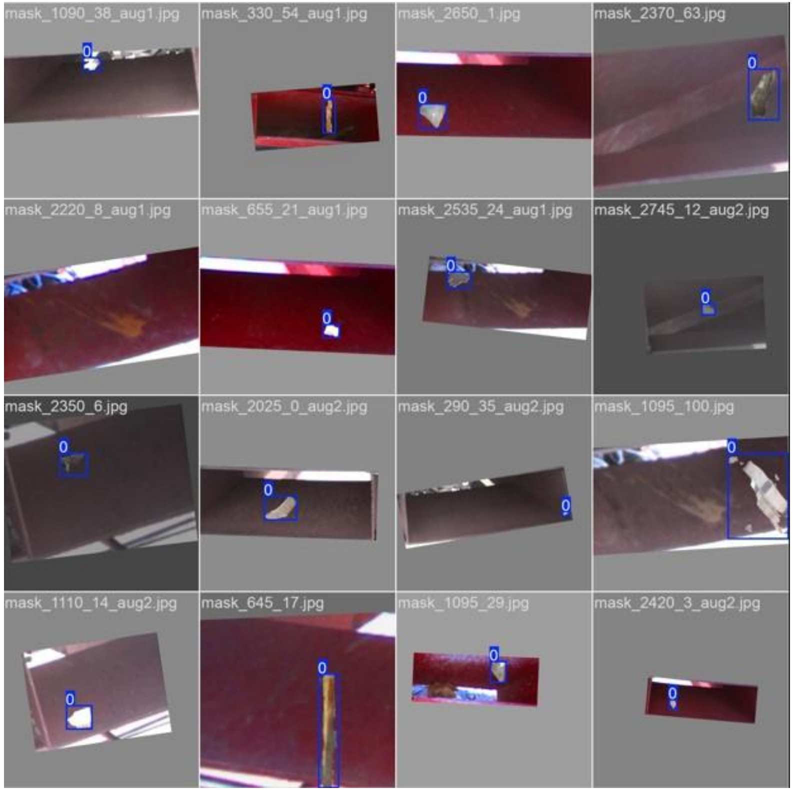
- Material Detection Model: 96.8% mAP@[0.5:0.95]
- Worker Detection Model: 74.5% mAP@[0.5:0.95]

Real-Time Processing System



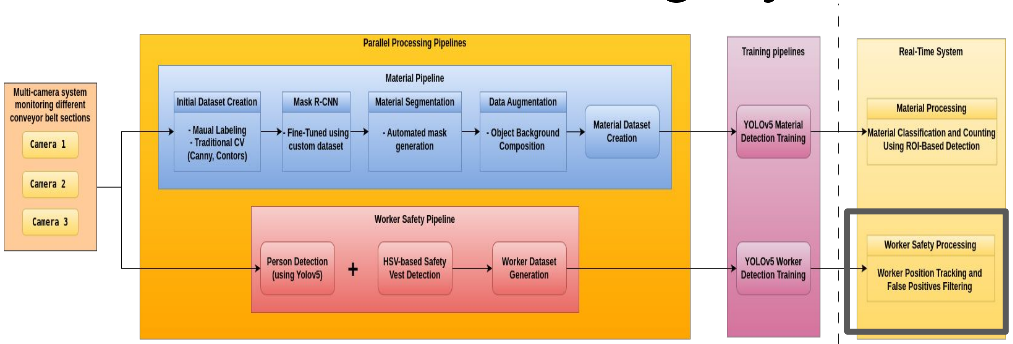
High-Level System view

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

Examples of False Positives

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.

Projects

VLM-Based Tool Recognition System for Industrial Safety Applications

Project Overview:

SafetyVLM is a production-ready system that enhances Vision-Language Models (VLMs) for specialized technical domains, with a focus on industrial tool recognition, usage instruction, and safety guidance. While VLMs have demonstrated remarkable capabilities in general visual understanding tasks, their application to safety-critical domains often lacks domain-specific knowledge and suffers from hallucinations. This project addresses this gap by fine-tuning state-of-the-art VLMs (Qwen2.5-VL-7B and Llama-3.2-11B-Vision) with a custom dataset of 8,458 tool images enriched with 29,567 safety annotations.

GitHub Repository: VirtuHand

Key Technologies and Skills Used:

Languages & Frameworks: Python, PyTorch, Transformers, Unsloth, TRL, LangChain, OpenCV, Pandas, NumPy
Machine Learning: PEFT, LoRA, RAG, RLHF, GRPO, LLM-guided Prompt Engineering, Vector Embeddings, Semantic Search
Deep Learning: Multi-modal Learning, Vision+Language Fine-tuning, Quantization, Gradient Checkpointing, Attention Mechanisms
Cloud & Infrastructure: AWS SageMaker, Kubernetes, Docker, MLOps Pipelines, Model Versioning, Distributed Training
APIs & Databases: OpenAI API, Pinecone Vector DB, FAISS, REST APIs, Cloud Storage, Real-time Processing

Motivation

THE PROBLEM:

- General VLMs struggle with domain-specific applications
- Industrial safety requires accurate, comprehensive guidance
- Hallucinations in safety information are dangerous
- Need specialized models for safety-critical domains

OUR APPROACH:
- Multi-strategy fine-tuning (Vision, Language, Combined)
- RAG pipeline with LangChain + Pinecone
- RLHF using GRPO for alignment
- Comprehensive evaluation framework

DATASET SCALE & DIVERSITY:
- 8,458 images with comprehensive safety annotations
- 29,567 annotations across 17 tool categories

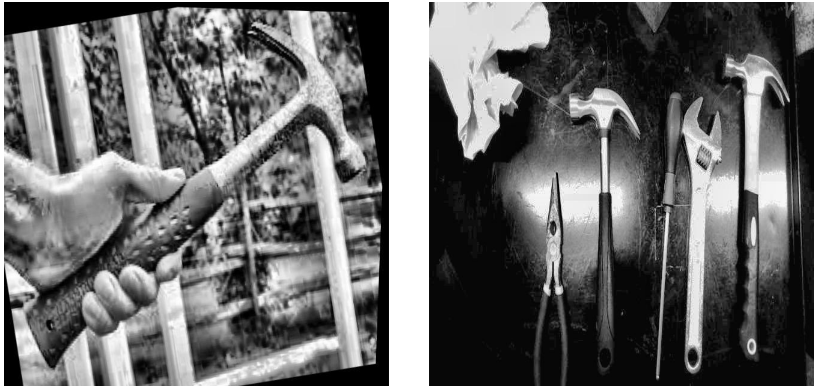


Fig.1 Sample tools from safety dataset: wrenches, pliers, and industrial equipment

System Architecture Overview

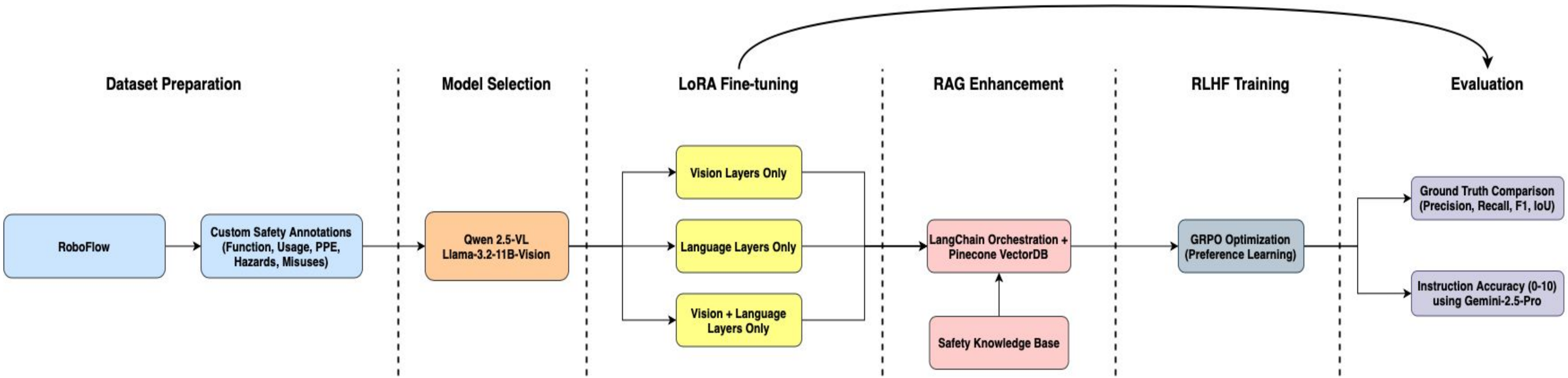


Fig.3 Complete pipeline architecture: from dataset preparation through evaluation, featuring LangChain orchestration and Pinecone vector database for production RAG implementation.

Dataset Creation & Safety Annotation Pipeline

LLM-GUIDED ANNOTATION INNOVATION:

- Automated Safety Metadata Generation: Structured prompts + few-shot learning
- Comprehensive Coverage: Function, PPE, hazards, common misuses
- Scaling Achievement: Manual → Automated annotation pipeline

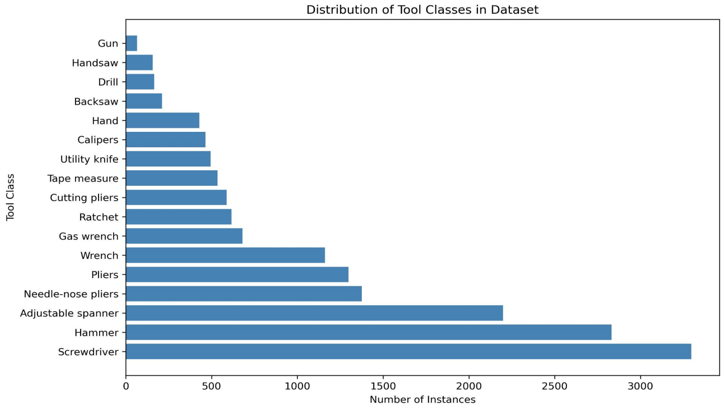


Fig.2 Distribution of 17 industrial tool classes in the Tool Safety Dataset.

Evaluation Prompt

```
Analyze this image and identify ALL
mechanical tools present. Return ONLY
a valid JSON object in the following
format:

{
  "detected_tools":
      ["tool_1", ...],
  "bounding_boxes": [
    { "tool": "tool_1",
      "bbox": [x1,y1,x2,y2]
     },
    ...
  ],
  "detailed_analysis": {
    "tools_information": [
      {
        "tool": "tool_1",
        "primary_function": "...",
        "safety_considerations": {
          "required_ppe": "...",
          "primary_hazards": [...],
          "common_misuses": [...]
        }
      },
      ... // Similar for other tools
    ]
  }
}
Important: Return ONLY JSON object,
no additional text or explanation.
```

Fig.4 Structured prompt used to evaluate VLM performance.

Model Selection & Fine-tuning Strategy

BASE MODEL SELECTION:

- Qwen-2.5-VL-7B: Strong multimodal performance, efficient inference
- Llama-3.2-11B-Vision: Larger capacity, robust vision understanding

THREE FINE-TUNING STRATEGIES:

- Vision-only (V): Fine-tune vision encoder layers only
- Language-only (L): Fine-tune language model components only
- Vision+Language (VL): Fine-tune both modalities jointly

TECHNICAL IMPLEMENTATION:

- LoRA: Parameter-efficient fine-tuning (r=16, α=16)
- 4-bit Quantization: Memory optimization for 8GB VRAM
- Unsloth Integration: 2x faster training, reduced memory footprint

LoRA Configuration

```
FastVisionModel.get_peft_model(
    model,
    finetune_vision_layers=True,
    finetune_language_layers=True,
    finetune_attention_modules=True,
    finetune_mlp_modules=True,
    r=16,
    lora_alpha=16,
    lora_dropout=0,
    bias="none",
    random_state=3407,
    use_rslora=False
)
```

Fig.5 LoRA Config for Fine-tuning VLMs using Unsloth.

Fine-tuning Results Comparison

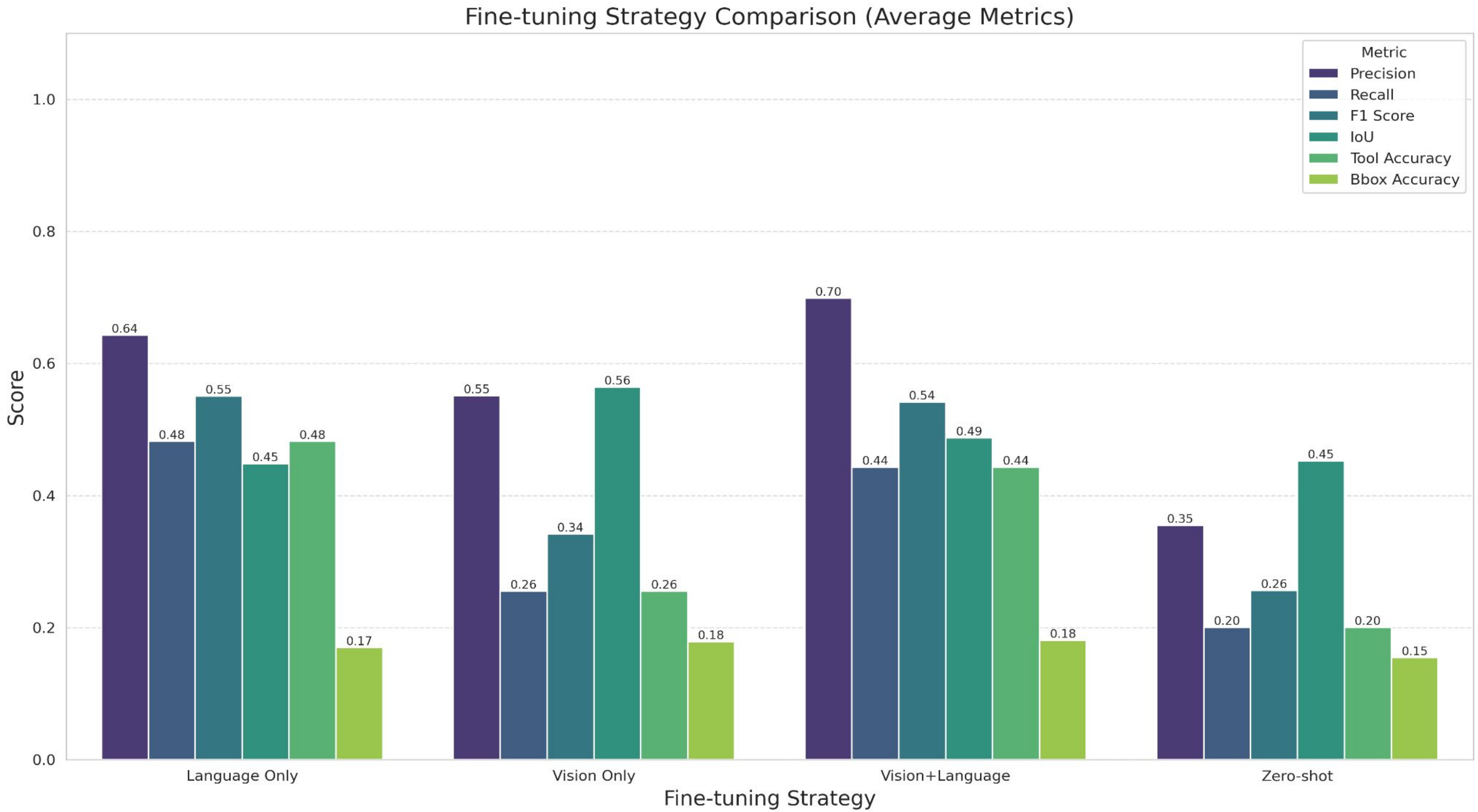


Fig.6 Performance comparison of different fine-tuning strategies across multiple evaluation metrics. Vision + Language outperforms others in both instruction quality and bounding box accuracy.

LangChain + Pinecone RAG Implementation

THE PROBLEM:

- Hallucinations in safety-critical information are dangerous
- Knowledge gaps in domain-specific applications

RAG SOLUTION:

- LangChain Orchestration: Workflow management + chain composition
- Pinecone Vector Database: Cloud-native, scalable vector storage
- Sentence Transformers: Semantic embeddings for tool safety knowledge

LangChain RAG Pipeline Implementation

```
class LangChainRAG:
    def __init__(self, model_path):
        self.vlm = UnslothVLM(model_path)
        self.vectorstore = PineconeVectorStore(
            embedding=SentenceTransformers()
        )
        self.retriever = self.vectorstore.as_retriever()

    def process_image(self, image):
        # Step 1: Identify tools
        tools = self.vlm.generate(image, "Identify tools")

        # Step 2: Retrieve safety context
        docs = self.retriever.invoke(tools)
        context = self.format_docs(docs)

        # Step 3: Enhanced generation
        prompt = f"Tools: {tools}\nContext: {context}"
        return self.vlm.generate(image, prompt)

    def format_docs(self, docs):
        return "\n".join([doc.content for doc in docs])
```

Fig.7 LangChain orchestration with Pinecone for contextual safety retrieval.

RAG Performance Impact

Metric	Standard VLM	RAG-Enhanced	Improvement
Hallucination Rate	45%	18%	60% reduction
Safety Info Completeness	62%	89%	43% increase
Technical Accuracy	73%	91%	28% improvement
Inference Latency	120ms	320ms	+167% overhead

Table. 1 Quantified Improvements

KEY INSIGHTS:

- Dramatic hallucination reduction from 45% to 18%
- Comprehensive safety coverage improved to 89%
- Latency trade-off motivates GRPO alignment approach

GRPO: Internalizing RAG Knowledge Without Latency

THE PROBLEM:

- RAG adds 200ms+ inference overhead (retrieval bottleneck)

GRPO SOLUTION:

- Preference Learning: RAG responses = "chosen", Standard = "rejected"
- No Reward Model: Direct optimization using preference pairs
- Knowledge Internalization: Model learns RAG behavior patterns

GRPO Preference Learning Implementation

```
# Generate preference pairs for GRPO training
for image in dataset:
    # Standard response (rejected)
    standard = model.generate(image, basic_prompt)

    # RAG-enhanced response (chosen)
    context = retrieve_safety_info(image)
    rag_prompt = f"{basic_prompt}\nContext: {context}"
    rag_response = model.generate(image, rag_prompt)

    # Create preference pair
    preference_pairs.append({
        'image': image,
        'chosen': rag_response,    # Preferred
        'rejected': standard       # Not preferred
    })

# Train with GRPO
trainer = GRPOTrainer(
    model=model,
    train_dataset=preference_pairs,
    beta=0.1  # Controls preference strength
)
```

Fig.8 GRPO training using preference pairs to internalize RAG knowledge.

GRPO Performance Results: Efficiency Without Compromise

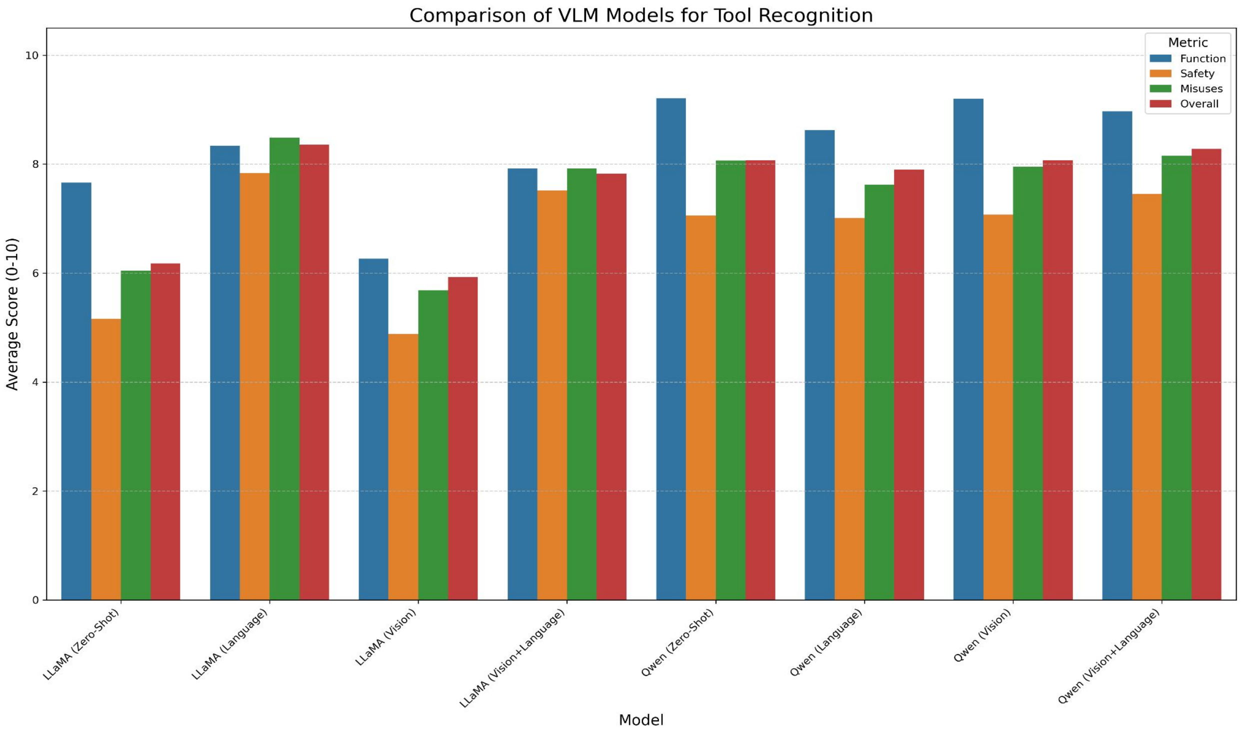
Metric	Standard VLM	RAG-Enhanced	GRPO-Aligned	Improvement
Hallucination Rate	45%	18%	23%	79% of RAG gains
Safety Info Completeness	62%	89%	81%	90% of RAG gains
Technical Accuracy	73%	91%	84%	75% of RAG gains
Inference Latency	120ms	320ms	155ms	98% latency reduction
Overall Score	6.2	8.7	8.1	84% of RAG gains

Table. 1 Performance Comparison

STRATEGIC ADVANTAGE:

- Best of both worlds: Accuracy gains + real-time performance
- Simplified architecture: No vector databases or retrieval APIs
- Cost efficiency: Reduced computational overhead in production

Evaluation Framework



TWO-TIER EVALUATION APPROACH:

- Detection Metrics: Precision, Recall, F1, IoU for tool identification
- Safety Content Quality: LLM-based assessment of information completeness

LLM-AS-A-JUDGE METHODOLOGY:

- OpenAI GPT-4o-mini as expert evaluator
- Multi-dimensional scoring: Tool ID, Function, Safety, Misuses (0-10 scale)

Results: Progressive Improvement Through Our Pipeline

Model	Tool Identification	Safety Instruction Quality
Basic Detection + LLM	Pliers	Gripping, bending, and manipulating objects with enhanced leverage and control
Zero-Shot VLM (Qwen)	Pliers	Cutting and Gripping
Fine-tuned VLM (Qwen)	Needle-Nose Pliers	Precision gripping, bending, and manipulating small wires and components. Wear safety gloves
RAG-Enhanced	Needle-nose pliers	Precision gripping, bending, and manipulating small wires and components in tight spaces. **Required PPE:** Safety glasses, work gloves. **Primary hazards:** Pinch points between handles, sharp wire ends, eye injury from flying debris. **Common misuses:** Using as wrench, applying excessive force on hardened materials
GRPO-Optimized	Needle-nose pliers	Precision gripping and manipulation of small wires and components in confined areas. **Safety considerations:** Use safety glasses and gloves, avoid pinch points, inspect tool condition before use, maintain proper grip

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

Project Overview:

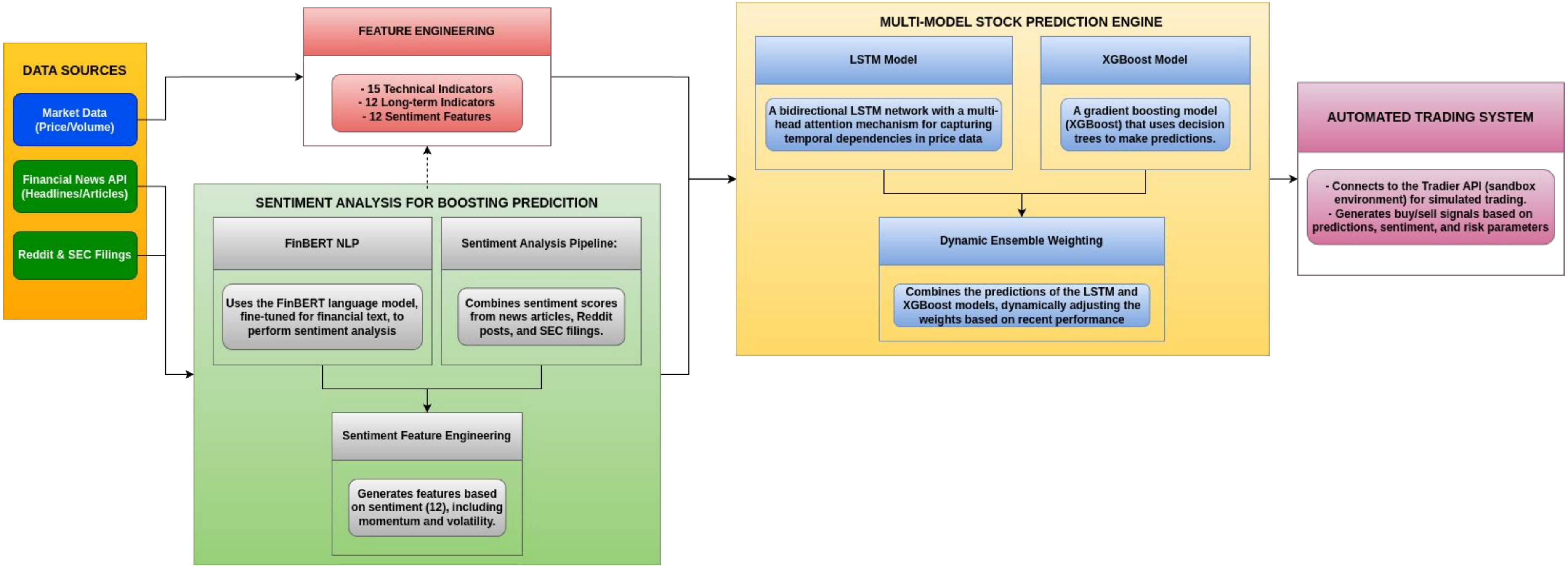
An end-to-end automated stock trading system that combines machine learning price prediction with NLP-based sentiment analysis. The system features a bidirectional LSTM with attention mechanism and XGBoost ensemble for multi-timeframe price forecasting, and integrates FinBERT for real-time sentiment 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:

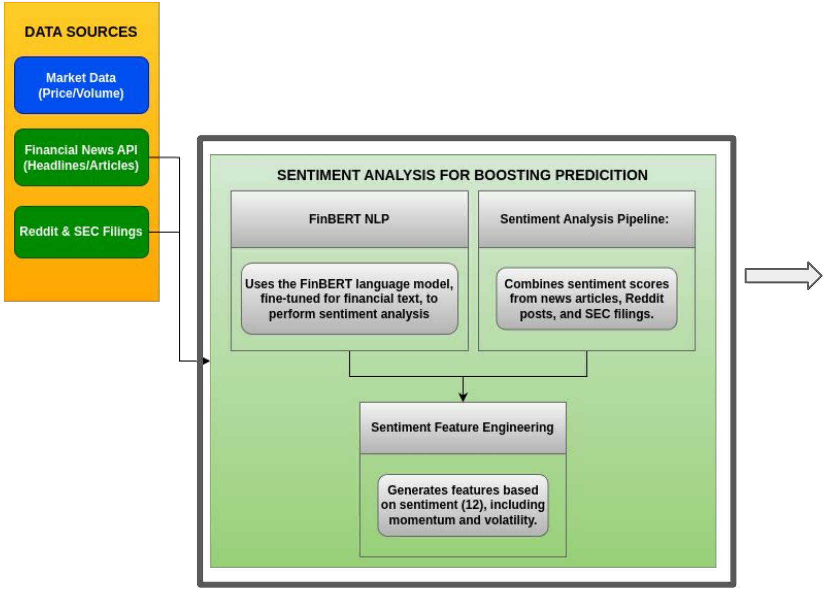
Languages & Frameworks: Python, PyTorch, TensorFlow, CUDA, Scikit-learn, Pandas, NumPy, Hugging Face Transformers
Machine Learning: Gradient Boosting (XGBoost), Feature Engineering, Regression, Time Series Forecasting
Deep Learning: LSTM Networks (Bidirectional, Attention), Model Ensembling, Model Training & 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:

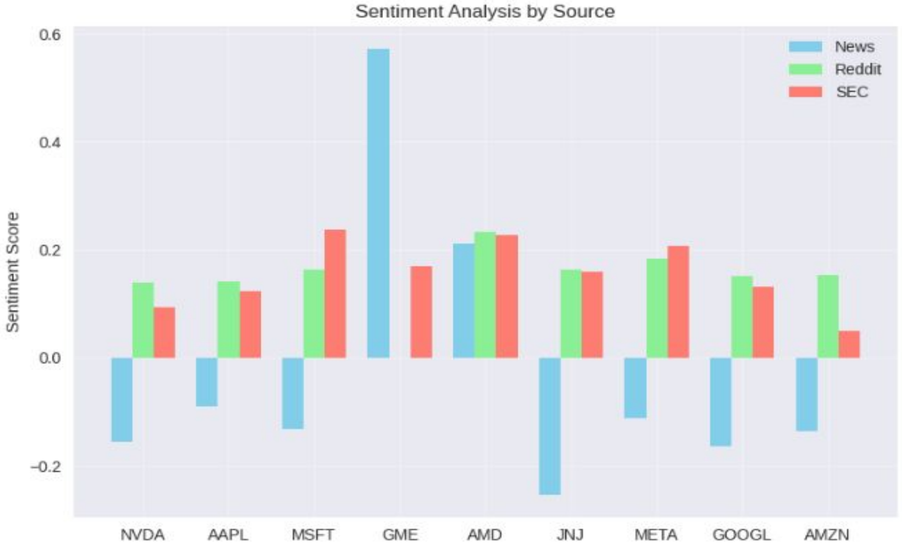


System Architecture and Data Flow

Multi-Source Sentiment Analysis:



System Architecture and Data Flow

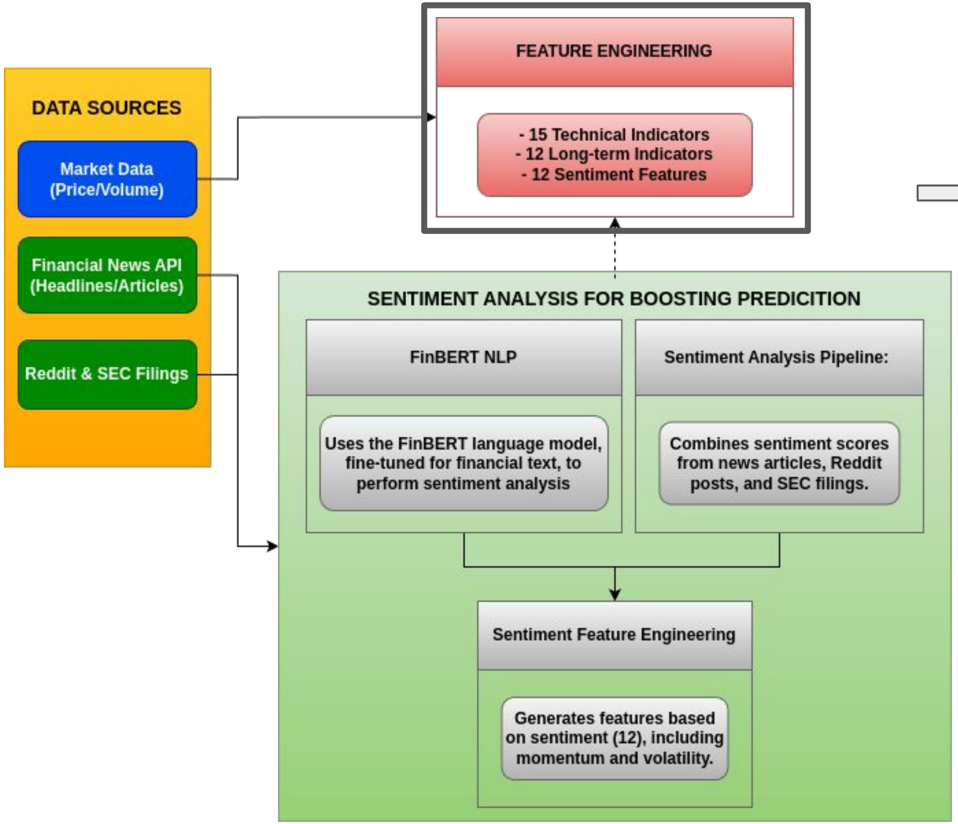


Sentiment scores for selected stocks; Positive values indicate positive sentiment, negative values indicate negative sentiment, and values near zero indicate neutral sentiment

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%)

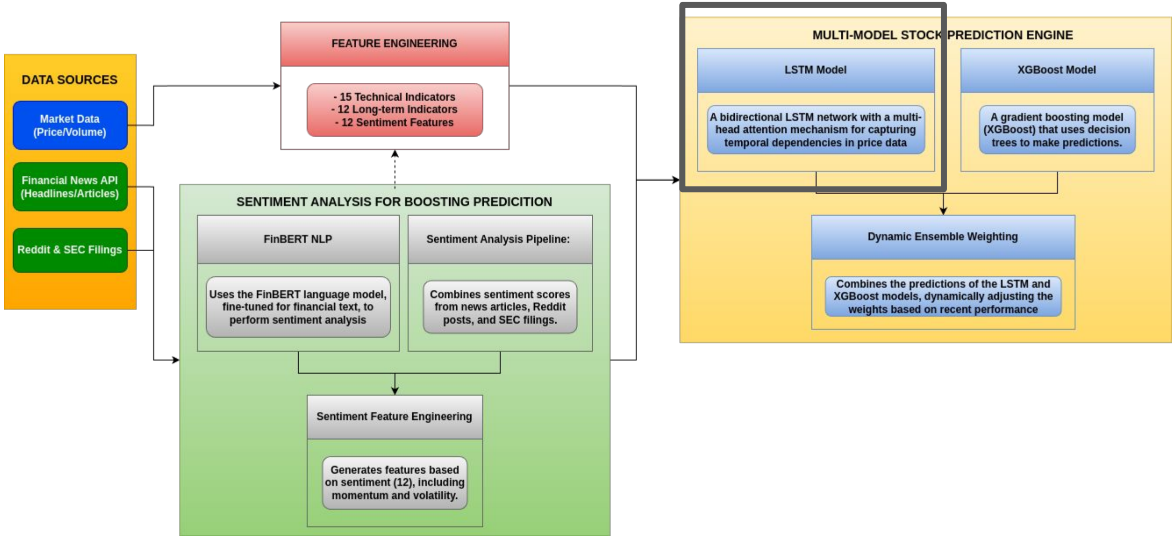
Feature Engineering:



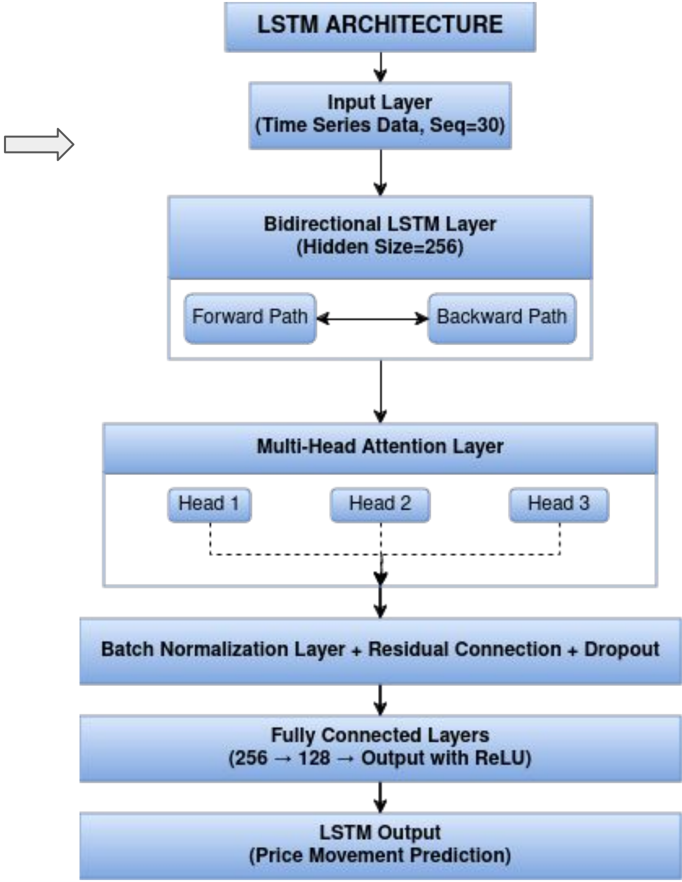
System Architecture and Data Flow

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

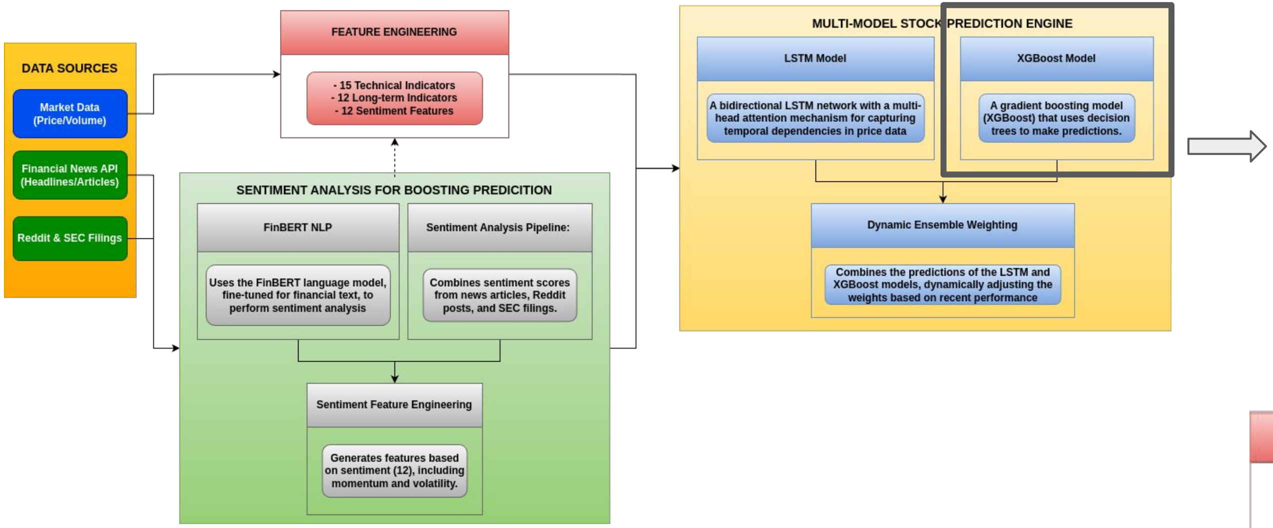


LSTM Model Architecture

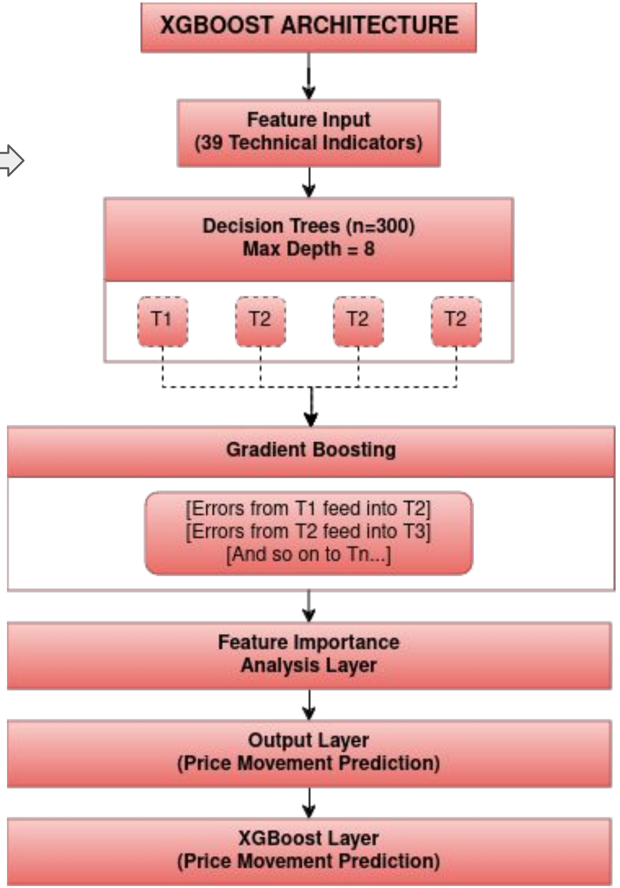
LSTM Neural Network:

- 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.

XGBoost for Price Prediction



System Architecture and Data Flow

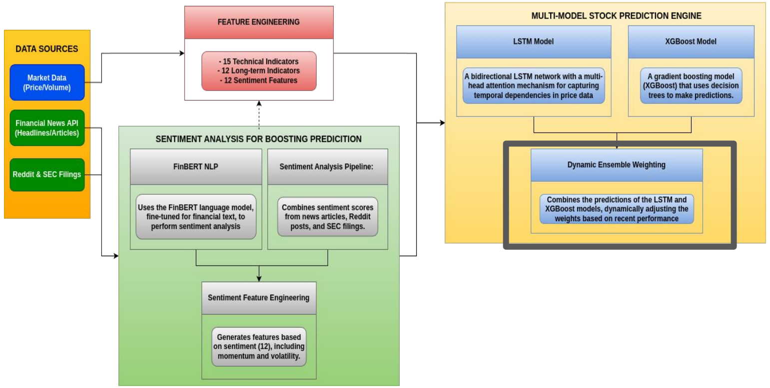


XGBoost Model Architecture

XGBoost Ensemble:

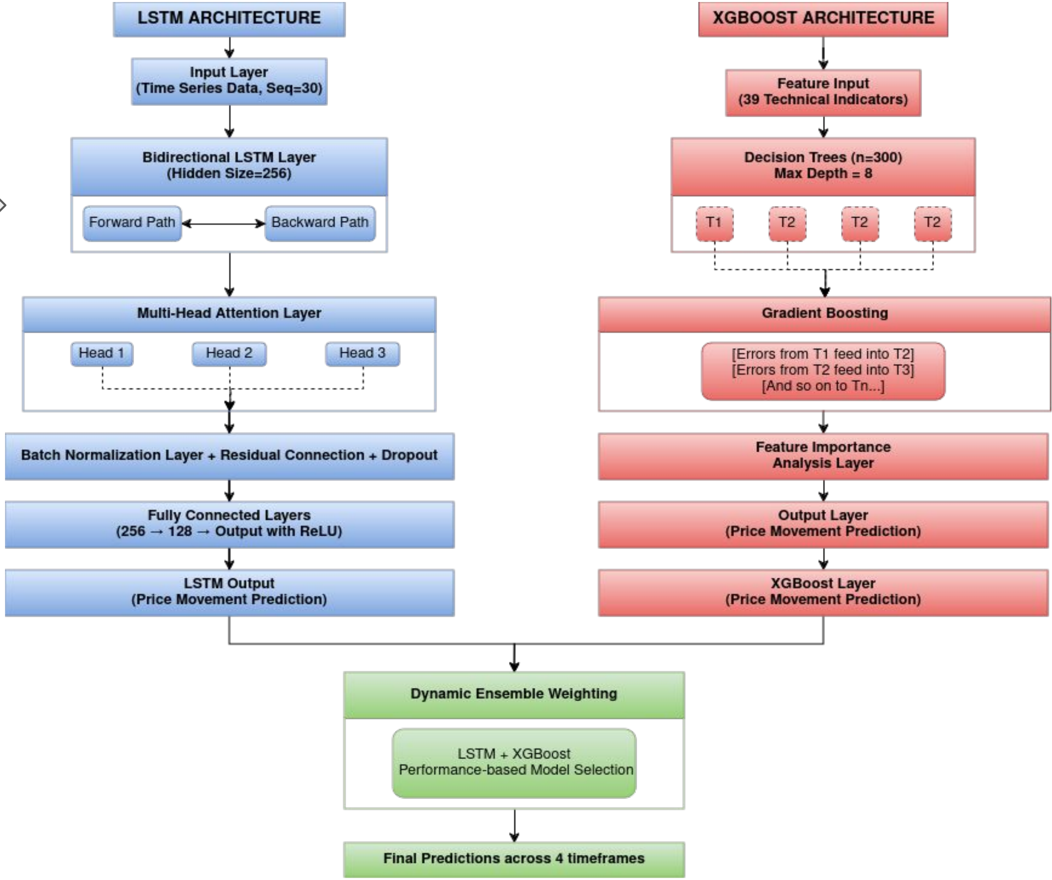
- Gradient boosting with 300 decision trees
- Feature engineering including 39 market indicators
- Min child weight:3, Subsample:0.8, Max depth:8

LSTM-XGBoost Ensemble for Price Prediction



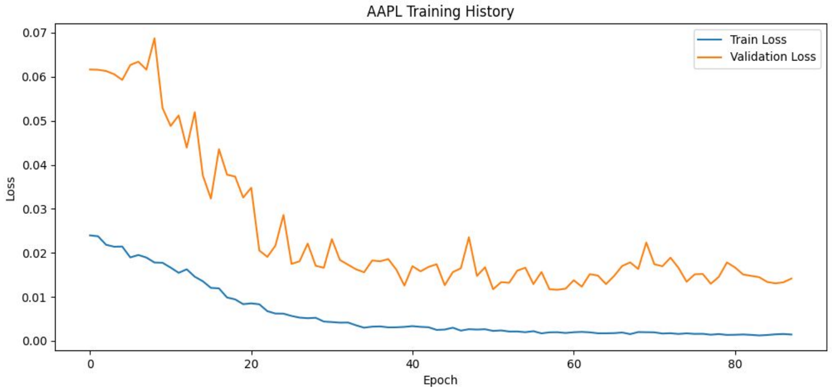
System Architecture and Data Flow

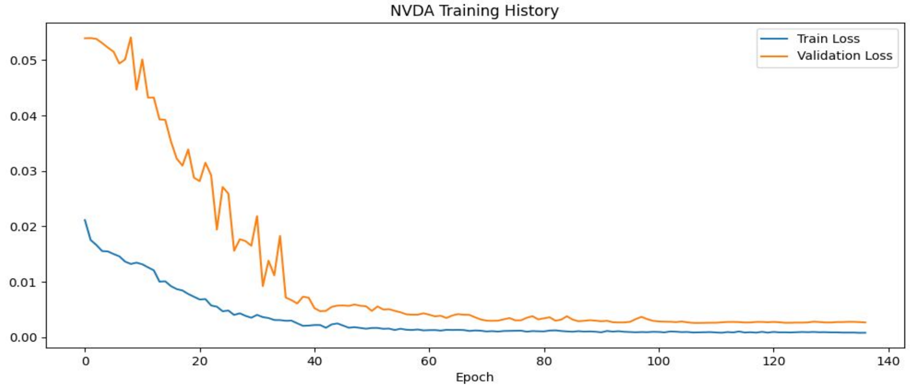
The trained ensemble model achieves a directional accuracy of 55-65% across multiple timeframes (5min, 15min, 30min, 1h) and a mean absolute error of 0.3-0.4% on normalized returns for multiple stocks.



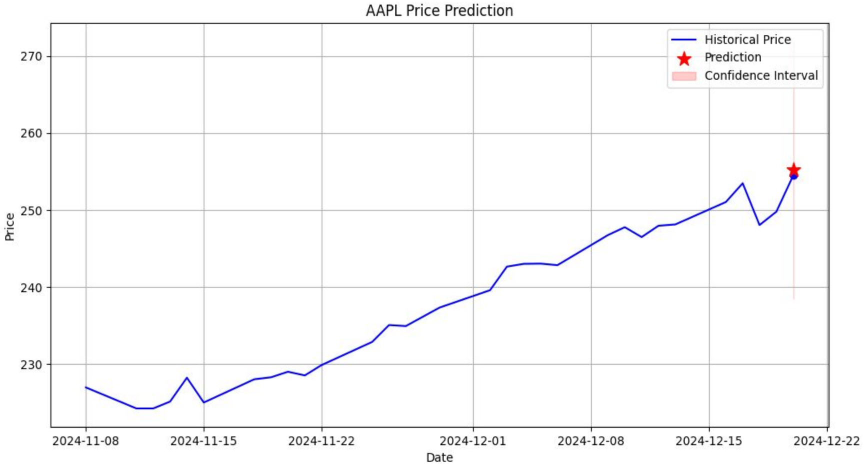
LSTM-XGBoost Ensemble Architecture

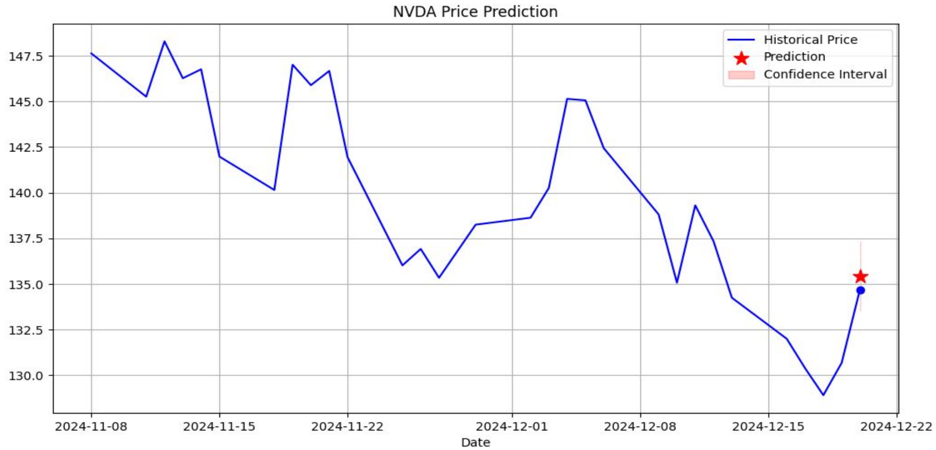
LSTM-XGBoost Ensemble Model Prediction Results with Examples





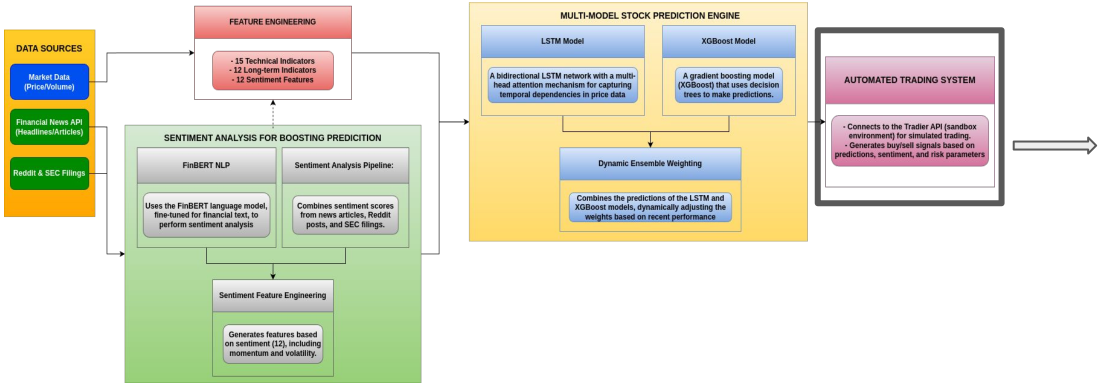
Training Convergences



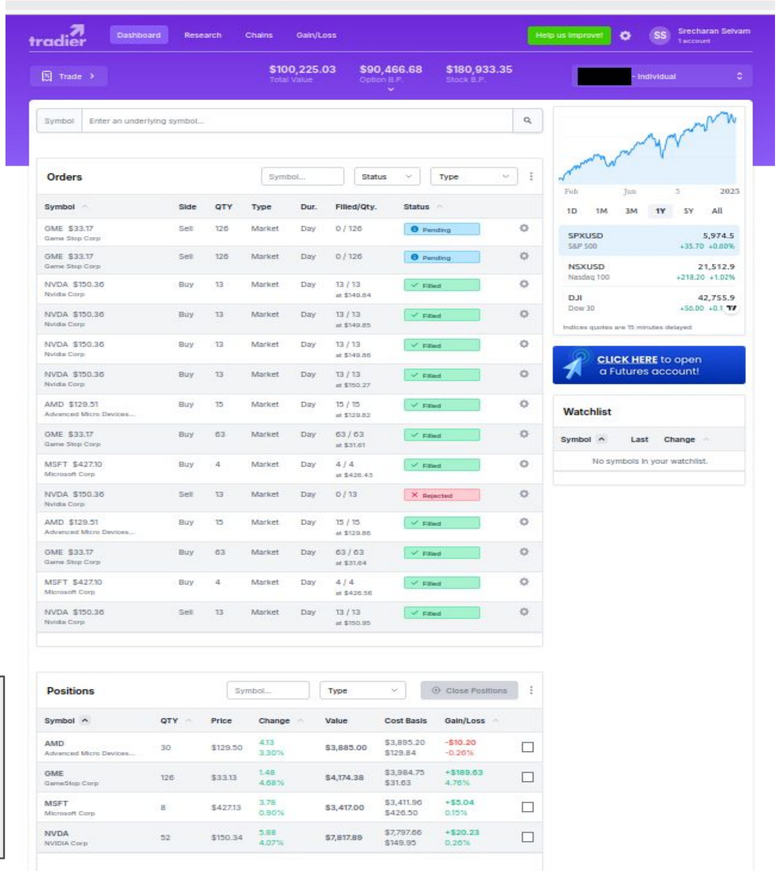


Price Predictions

Automated Trading System: Real-World Implementation



System Architecture and Data Flow



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

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

GenAI for Synthetic Data Augmentation: GANs, VAEs & Diffusion

Project Overview:

A comprehensive generative AI system that combines comparative analysis of three leading architectures with practical validation through synthetic data augmentation for computer vision applications. 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 on the challenging CUB-200-2011 bird dataset. Beyond theoretical comparison, the work validates practical utility through a synthetic data augmentation pipeline that improves bird species classification performance. WGAN-GP emerged as the strongest performer with 33.07 FID score and proved most effective for data augmentation, achieving 5.1% accuracy improvement on full datasets and 15.7% boost in data-scarce scenarios through ResNet-50 validation.

GitHub Repository: GenVision

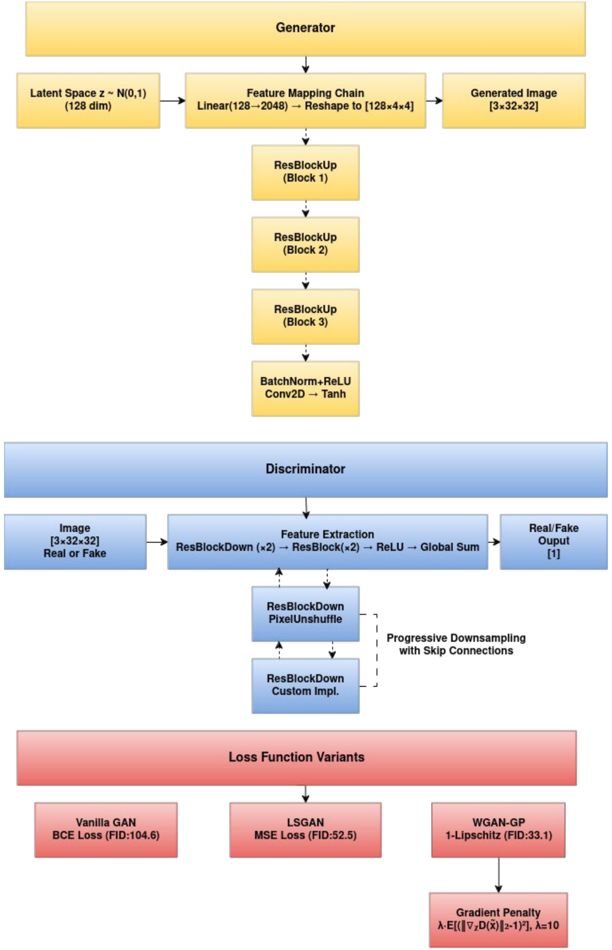
Key Technologies and Skills Used:

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, Feature Visualization, Image Classification, Data Augmentation, Synthetic Data Generation
Data Science: Dataset Creation, Performance Metrics (FID, Precision, Recall), Data Scarcity Solutions, Augmentation Strategies

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]

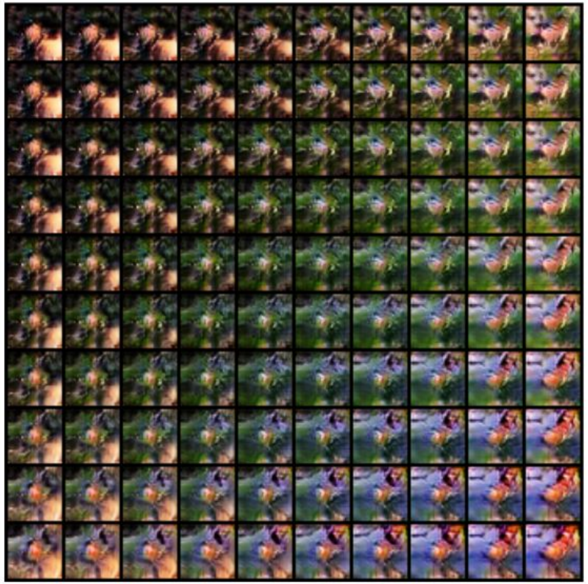


Discriminator:

- 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

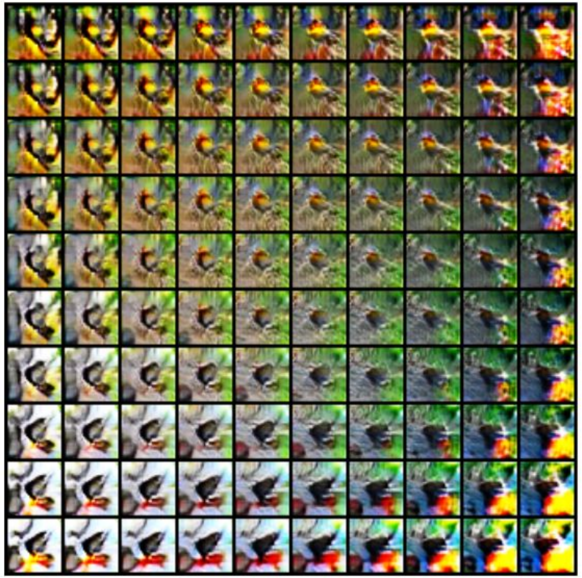


Vanilla GAN Latent Space Interpolations

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

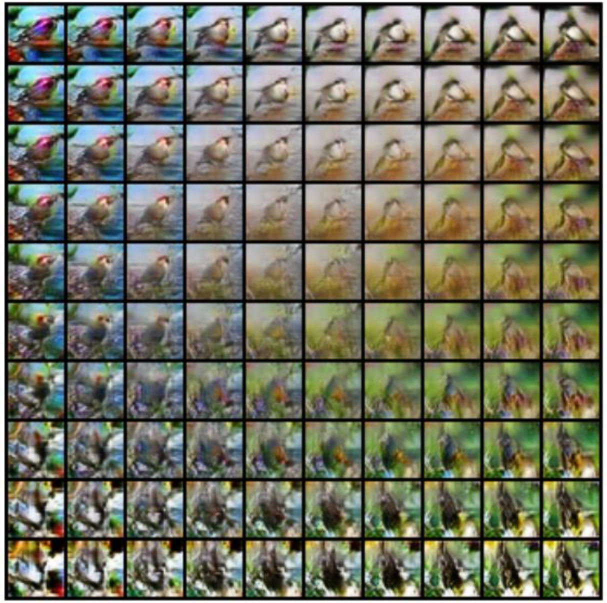


LS-GAN Latent Space Interpolations

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



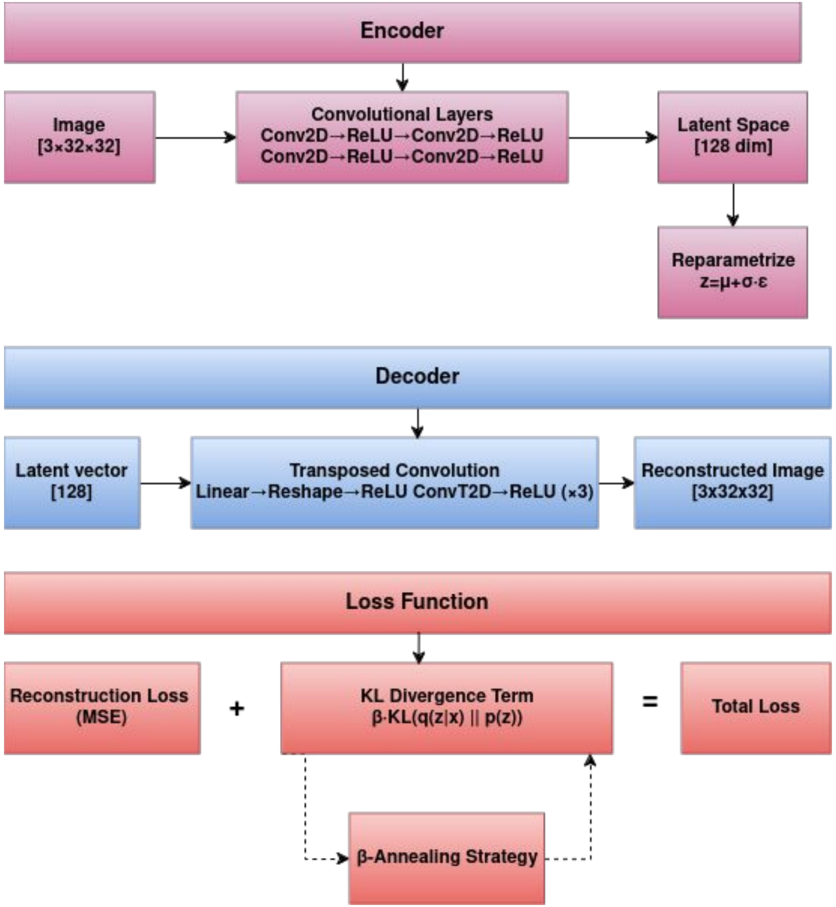
WGAN-GP Latent Space Interpolations

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: $z = \mu + \sigma * \varepsilon$ where $\varepsilon \sim N(0,1)$



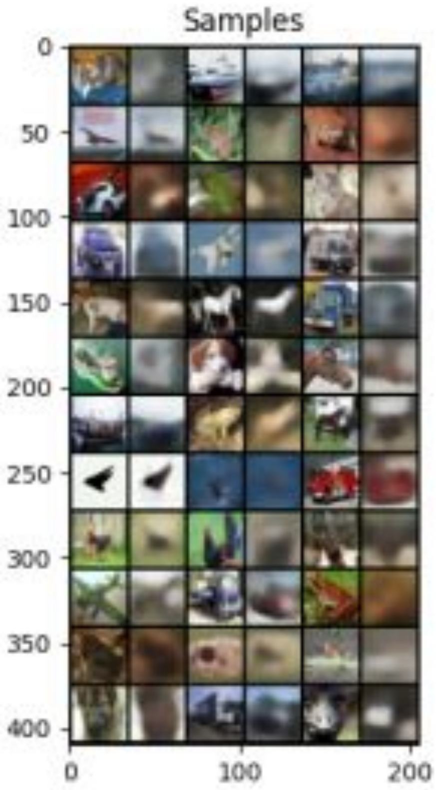
Decoder:

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

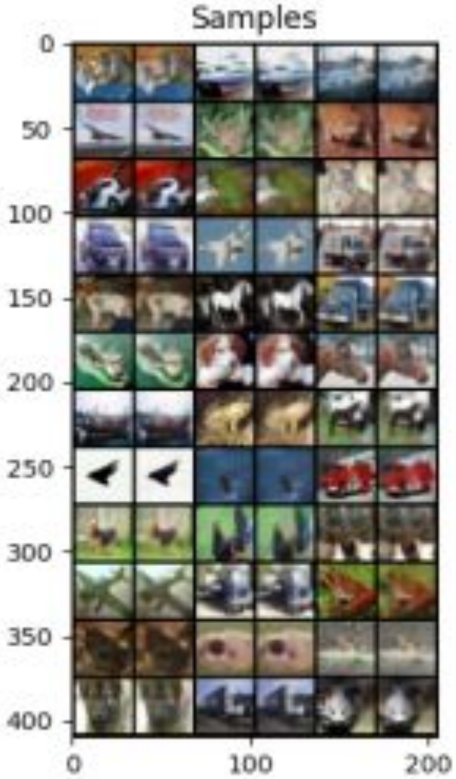
Loss Function:

- 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

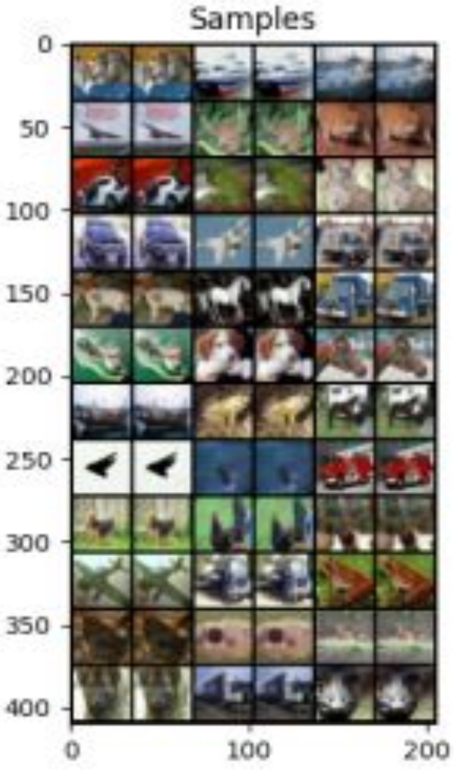
VAE Latent Space Exploration



Reconstruction sample: (size 16)



Reconstruction sample: (size 128)



Reconstruction sample: (size 1024)

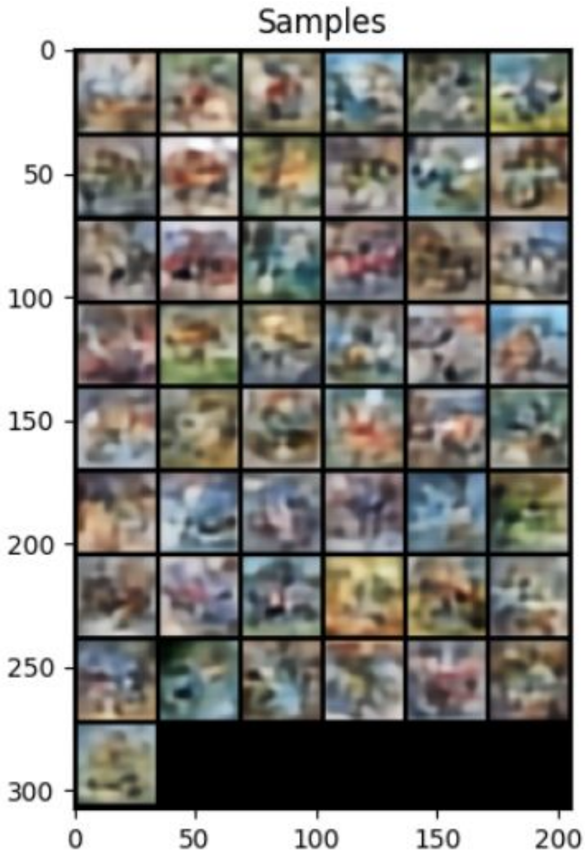
Reconstruction loss: ~200

Reconstruction loss: ~200 → ~75

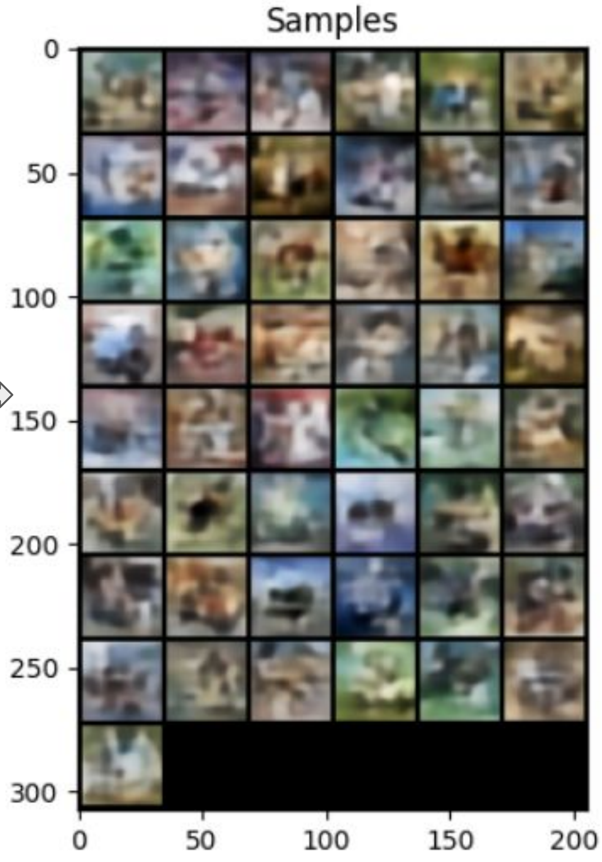
Reconstruction loss: ~200 → ~75 → ~40

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.



β-Annealing

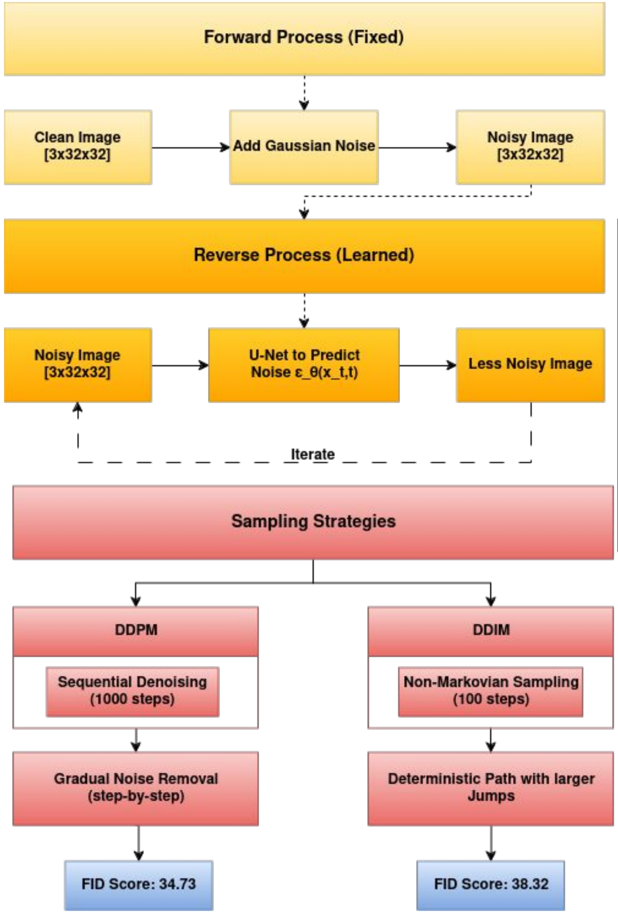


The left image shows samples generated with a fixed β value of 0.8. The right image shows samples generated after training with β-annealing (linearly increasing β from 0 to 0.8 over the first 20 epochs)

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):

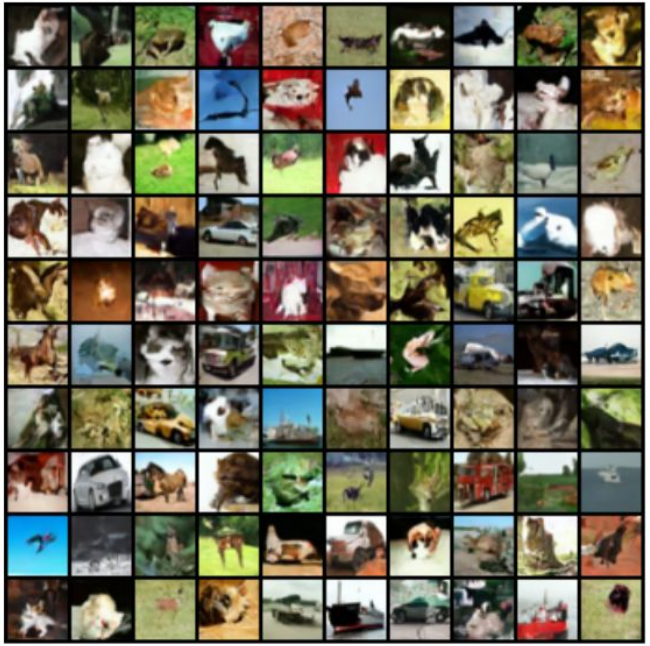
- Input: Noisy image at timestep t
- Architecture: U-Net predicts noise component ε_θ (x_t, t)
- Process: Iterative denoising by subtracting predicted noise
- Output: Progressively cleaner image, approaching the original distribution

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