BioTrack AI: Near-Real-Time Biofuel Rail Monitoring System

Computer vision monitoring of rail traffic represents an untapped opportunity for commodity intelligence in the renewable fuels market. This research establishes that **no competitor currently offers CV-based commercial rail intelligence for biofuels**—all existing applications focus on safety and inspection. The data latency gap is significant: current best-in-class sources provide weekly aggregate data while physical supply information lags 4-12 weeks. A CV-based system could achieve **near-real-time facility-level granularity** that doesn't exist from any commercial source today.

The proof of concept is achievable within one week using free tools: Virtual Railfan's La Plata, MO camera (Virtualrailfan) (60-100+ trains daily on BNSF's biofuel corridor), YOLOv8 with Roboflow's railcar dataset, and EasyOCR for reporting marks. However, **public camera feeds cannot be productionized** due to explicit Terms of Service prohibitions—commercial deployment requires proprietary camera installation at strategic chokepoints.

Rail camera networks present legal barriers for commercial use

Two major networks dominate public rail camera access: **RailStream** (48 cameras, $8.95-$12.95/month) (Railfanhq) and **Virtual Railfan** (100+ cameras, ~24 free on YouTube). (Union Pacific) Both offer 1080p video quality suitable for detection, but their Terms of Service explicitly prohibit commercial capture and analysis.

Virtual Railfan's terms state unambiguously: "Recording, duplication, distribution or re-streaming Virtual Railfan content is strictly prohibited without prior consent." (Virtualrailfan) RailStream similarly prohibits reproducing, copying, or exploiting content for commercial purposes. (Railstream) YouTube's API Terms of Service add another layer, prohibiting automated systems that send excessive requests and commercial use without approval. (TLDRLegal)

Programmatic access is technically straightforward despite legal constraints. For YouTube streams, the combination of yt-dlp and ffmpeg enables frame extraction:

```bash
yt-dlp --downloader ffmpeg --downloader-args "ffmpeg:-t 3600" URL
ffmpeg -i input.mp4 -vf fps=1 frame_%04d.png
```

The (cap-from-youtube) Python library provides direct OpenCV integration for real-time frame analysis. (PyPI) For non-YouTube streams like RailStream's proprietary player, browser automation would be required—further violating terms.

Critical finding: No existing public cameras are positioned near target biofuel facilities. Diamond Green Diesel Norco, Marathon Martinez, Valero Port Arthur, and ADM Decatur all lack nearby camera coverage. The networks focus on scenic railfan locations rather than industrial facilities.

Private camera deployment requires either railroad right-of-way permits (30-60 day process through UP/BNSF/CSX real estate departments) or agreements with adjacent property owners. Hardware costs range from **$500-8,000** per camera (Hikvision/Axis PTZ with 32x optical zoom, IP67 weatherproofing, 200m+ IR night vision) plus **$5,000-20,000** installation including trenching and connectivity—Virtual Railfan reported $16,000 for their Cajon Pass installation. Union Pacific

---

Strategic monitoring locations cluster around Gulf Coast and Midwest hubs

Rail network topology analysis identifies specific chokepoints where biofuel feedstock movements concentrate. The key facilities and their rail connections:

Diamond Green Diesel Port Arthur, TX receives feedstock via Howard Energy Partners' terminal featuring **16 miles of track, unit train capacity from two Class I railroads** (CPKC and BNSF/UP via Beaumont). The Beaumont Junction area (GCL Junction where KCS connects with UP's Sunset Route) serves as a critical gateway where **all traffic to Port Arthur must pass**.

Diamond Green Diesel Norco, LA connects to Union Pacific through the St. Charles Parish refinery corridor. The facility receives feedstock via rail, truck, and ship, with renewable diesel transported to California markets via pipeline, rail, or ocean carrier. Darling Ingredients

Marathon Martinez, CA (world's largest renewable diesel refinery at 730 million gallons/year) connects to Union Pacific's Martinez Subdivision. Feedstocks including animal fat, soybean oil, corn oil, and UCO arrive via rail and marine terminals. CA The Suisun Bay corridor along Carquinez Strait provides the monitoring chokepoint.

ADM Decatur, IL operates from 1,125 acres Center for Land Use Interpretat… with exceptional rail connectivity: **three Class I railroads interchange** (Norfolk Southern, Canadian National, and CSX) plus the Decatur & Eastern Illinois shortline connecting to Union Pacific. ADM owns 28,000 railcars, and the facility's intermodal ramp handles 150,000 lifts annually.

The feedstock supply chain flows from UCO collectors like **Darling Ingredients** (260+ facilities globally, 90+ U.S. processing locations, 50% owner of Diamond Green Diesel) through rendering facilities to biofuel plants. Midwest feedstock typically routes through Kansas City (the nation's second-largest rail hub where all six Class I railroads have access) before heading to Gulf Coast facilities via BNSF or KCS.

Priority monitoring locations by tier:

- **Tier 1** (Facility approaches): Port Arthur/Beaumont corridor, Norco/St. Charles Parish, Martinez/Carquinez Strait
- **Tier 2** (Chokepoints): Beaumont Junction, New Orleans gateway, Kansas City, Decatur interchange

- **Tier 3** (Route monitoring): KCS mainline Kansas City-Port Arthur, UP Sunset Route Houston-New Orleans

---

Computer vision achieves 85-95% detection accuracy with existing tools

YOLOv8 delivers strong performance for railcar detection without custom training. Hugging Face The **Railway Cars Dataset** on Roboflow (Chris Immel, 1,882 images) includes particularly relevant classes: tank-railcar, hopper-railcar, and critically, **side-reporting-mark and end-reporting-mark detection**—essential for OCR integration.

Model	mAP@50	Parameters	Speed (A100 TensorRT)
YOLOv8n	37.3%	3.2M	0.99ms
YOLOv8s	44.9%	11.2M	1.20ms
YOLOv8m	50.2%	25.9M	1.83ms

Custom-trained YOLOv8 models on railway-specific datasets typically achieve **85-95% mAP**. Tank cars are visually distinctive: cylindrical horizontal body, smooth rounded profile, centralized top dome with safety valve, length 40-60 feet. Wordpress Classification accuracy between car types reaches **90-95%** on clear images.

OCR for railcar reporting marks (format: 2-4 letter prefix + 4-6 digit number, e.g., UTLX 123456) Wordpress presents the primary technical challenge. Comparative analysis shows:

Engine	Character Accuracy	Speed	Best Use Case
Tesseract	~90%	Fast	CPU environments, fine-tunable
EasyOCR	~95%	Medium	GPU acceleration, general purpose
PaddleOCR	~97%	Fast	Best accuracy on alphanumerics

Realistic OCR accuracy expectations:

- Stationary trains: 90-95% character accuracy achievable
- Slow-moving trains (<10 mph): 80-90% with good camera setup
- Fast-moving trains (>30 mph): 60-80% without specialized hardware

The critical insight is that **yard operations and approach tracks** where trains slow down provide the best OCR opportunities. Fast mainline passing is challenging; facility approach tracks where trains decelerate for switching offer ideal capture windows.

Resolution requirements: 1080p minimum for reliable detection; 4K preferred for OCR. Reporting marks (7-9 inch letters on car sides) (National Model Railroad Assoc…) need **25+ pixels per character** for optimal recognition. Frame rate of 30 fps handles speeds up to 60 mph; most yard operations at 10-30 mph work fine with 25 fps.

Technical pipeline architecture enables scalable processing

The detection-to-inference workflow follows a five-stage pipeline: Frame capture → YOLO detection with tracking → Crop ROIs → OCR → Parse and persist to database.

Frame capture uses ffmpeg extracting 1 frame per second (sufficient for railcar detection—trains move slowly enough that higher rates waste resources). (OpenCV) For YouTube streams:

```python
from cap_from_youtube import cap_from_youtube
cap = cap_from_youtube(youtube_url, 'best')
```

For RTSP/private cameras, OpenCV VideoCapture with reconnection logic handles stream interruptions. (OpenCV Q&A Forum) (TutorialsPoint)

YOLO detection with BoT-SORT tracking maintains object identity across frames:

```python
from ultralytics import YOLO
model = YOLO("yolov8m.pt")
results = model.track(frame, persist=True, tracker="botsort.yaml")
for box in results[0].boxes:
    track_id = box.id.int().item()  # Persistent ID across frames
```

Track configuration with (track_buffer: 30) keeps lost tracks for 30 frames, essential for handling momentary occlusions.

Deduplication operates at two levels: same car across multiple frames at one location (5-minute time window), and same car seen at different locations over time (building trip records). The tracking ID from BoT-SORT handles within-session deduplication; database queries handle cross-session matching by reporting mark.

Database schema (PostgreSQL with TimescaleDB for time-series optimization):

```sql
CREATE TABLE sightings (
    sighting_id BIGSERIAL,
    timestamp TIMESTAMPTZ NOT NULL,
    location_id INTEGER REFERENCES locations(location_id),
    car_id INTEGER REFERENCES cars(car_id),
    reporting_mark VARCHAR(10) NOT NULL,
    detection_confidence DECIMAL(4, 3),
    ocr_confidence DECIMAL(4, 3),
    image_path TEXT,
    bounding_box JSONB,
    PRIMARY KEY (sighting_id, timestamp)
);
SELECT create_hypertable('sightings', 'timestamp');
```

Volume estimation uses standard tank car capacity of **25,000 gallons** (range 20,000-34,500) with confidence intervals. Aggregating unique tank cars by facility and time period provides throughput estimates unavailable from any existing data source.

Competitive landscape reveals significant market opportunity

Current biofuel feedstock intelligence providers include **OPIS** (daily spot prices, weekly rack reports, RINs pricing), **Argus Media** (global biofuels coverage, long-term forecasts), and **S&P Global Platts** (~2,000 biofuel price assessments). (S&P Global Commodity Insights) Annual subscriptions range from **$20,000-$150,000** depending on scope.

Rail traffic intelligence from **Railinc** (RailSight suite, real-time shipment tracking) and **AAR** (weekly carload reports) provides aggregate data only—no facility-level granularity. The **STB Waybill Sample** offers detailed origin-destination data but with 12+ month lag and masking requirements preventing facility identification.

The critical gap BioTrack fills: No provider offers real-time or near-real-time facility-level feedstock movement visibility. Current data latency comparison:

Source	Latency	Granularity
OPIS/Argus prices	Daily (EOD)	Regional markets
AAR Weekly Traffic	7-10 days	Aggregate by commodity
EIA Monthly Update	6-8 weeks	National totals
STB Waybill	12+ months	Masked sample
CV Monitoring (potential)	Hours to days	Facility-specific

Who would pay: Trading firms (Cargill, Vitol, Trafigura), renewable diesel producers (Marathon, Valero, Neste), feedstock suppliers (Darling, Baker), and investment firms seeking alternative data. Pricing precedent from maritime tracking (Kpler, Vortexa) suggests **$50,000-$200,000/year** for specialized commodity intelligence. With 50-200 potential enterprise customers, addressable market reaches **$10M-$40M**.

Proof of concept achievable in one week with free tools

Best camera for first results: Virtual Railfan La Plata, MO on BNSF's Southern Transcon—60-100+ trains daily, high tank car frequency (major ethanol corridor), excellent video quality. Alternative: Fort Madison, IA (crew change point with varied traffic).

Day 1 execution plan:

1. **Frame capture** (30 min): Use `cap-from-youtube` or manual screenshot
2. **YOLO detection** (1 hour): Run pre-trained YOLOv8n on captured frames
3. **OCR test** (30 min): Apply EasyOCR to detected regions

```python
```

```
# Quick POC code
from ultralytics import YOLO
import easyocr

model = YOLO('yolov8n.pt')
reader = easyocr.Reader(['en'])

results = model('rail_frame.jpg')
for box in results[0].boxes:
    # Crop detected region, run OCR
    cropped = frame[int(box.xyxy[0][1]):int(box.xyxy[0][3]),
                    int(box.xyxy[0][0]):int(box.xyxy[0][2])]
    ocr_results = reader.readtext(cropped)
```

Success metrics for POC:

- Detection: ≥70% mAP@0.50 (production target ≥85%)
- OCR: ≥60% full mark accuracy (production target ≥85%)
- Sample size: 20+ frames, 10+ tank car detections, 5+ complete OCR reads

Key risks: Video quality variation between cameras, nighttime performance degradation (recommend daylight-only for POC), weather/dirty car impacts on OCR (expect 20-40% initial failure rate).

NOTION DATABASE ENTRIES

Data Sources Registry

Entry 1: Virtual Railfan YouTube Streams

Field	Value
Source Name	Virtual Railfan
Type	Live Video Stream
URL	https://www.youtube.com/c/VirtualRailfan
Coverage	100+ cameras, 50+ locations, 25 states (WSBT)
Quality	1080p maximum

Field	Value
Access Method	YouTube API / cap-from-youtube
Cost	Free (24 cameras) / $4.99-$24.99/mo (full access) Virtualrailfan
Legal Status	ToS prohibits commercial capture
Refresh Rate	Real-time
Priority for BioTrack	HIGH (POC only)

Entry 2: RailStream

Field	Value
Source Name	RailStream
Type	Live Video Stream (proprietary)
URL	https://railstream.net
Coverage	48 cameras, 22 locations
Quality	1080p (moving to 4K)
Access Method	No API - proprietary player
Cost	Free-$12.95/mo
Legal Status	ToS prohibits commercial use
Priority for BioTrack	MEDIUM

Entry 3: FRA Rail Network Data

Field	Value
Source Name	FRA North American Rail Network
Type	GIS Dataset
URL	https://fragis.fra.dot.gov/
Coverage	Complete US rail network

Field	Value
Data Includes	Rail lines, nodes, yards, crossings, mileposts
Access Method	ArcGIS REST API
Cost	Free
Format	FeatureServer, downloadable shapefiles
Priority for BioTrack	HIGH

Entry 4: Roboflow Railway Cars Dataset

Field	Value
Source Name	Railway Cars Dataset (Chris Immel)
Type	Training Dataset
URL	https://universe.roboflow.com/chris-immel/railway-cars Roboflow
Size	1,882 images
Classes	tank-railcar, hopper-railcar, side-reporting-mark, end-reporting-mark Roboflow
Format	YOLOv8, COCO JSON, TFRecord
License	CC BY 4.0 Roboflow
Pre-trained Model	Yes
Priority for BioTrack	CRITICAL

Entry 5: Railinc UMLER

Field	Value
Source Name	Railinc Umler System
Type	Equipment Registry
URL	https://www.railinc.com/rportal/umler
Coverage	2M+ rail equipment records

Field	Value
Data Includes	Car specifications, owner codes, capacity
Access Method	Subscription API
Cost	Enterprise pricing
Priority for BioTrack	HIGH (for validation)

Entry 6: OpenRailwayMap

Field	Value
Source Name	OpenRailwayMap
Type	GIS Dataset
URL	https://www.openrailwaymap.org/
Coverage	Global (crowdsourced)
Data Includes	Tracks, spurs, sidings, signals, switches
Access Method	Overpass API
Cost	Free
Key Tags	railway=rail + service=spur
Priority for BioTrack	HIGH

Agent Registry

Agent 1: Frame Scraper

Field	Value
Agent Name	frame_scraper
Purpose	Capture frames from video streams at configurable intervals
Input	Stream URL (YouTube/RTSP), capture interval, output directory

Field	Value
Output	JPEG frames with timestamp metadata
Dependencies	yt-dlp, ffmpeg, opencv-python, cap-from-youtube
Deployment	Docker container, one instance per stream
Resource Requirements	2GB RAM, minimal CPU
Error Handling	Auto-reconnect on stream failure, exponential backoff
Status	TO BUILD

Agent 2: Car Detector

Field	Value
Agent Name	car_detector
Purpose	Detect and classify railcars in frames, maintain tracking IDs
Input	Frame images from Redis stream
Output	Detections with bounding boxes, car type, confidence, track_id
Model	YOLOv8m fine-tuned on Railway Cars dataset
Dependencies	ultralytics, opencv-python, redis
Deployment	GPU-enabled container (NVIDIA Jetson for edge, cloud GPU for scale)
Resource Requirements	4GB VRAM, CUDA support
Confidence Threshold	0.5 detection, 0.7 high-confidence
Status	TO BUILD

Agent 3: OCR Reader

Field	Value
Agent Name	ocr_reader
Purpose	Extract reporting marks from detected railcar regions

Field	Value
Input	Cropped car images from detection pipeline
Output	Reporting mark text, confidence score, parsed owner code + number
Engine	PaddleOCR (primary), EasyOCR (fallback)
Dependencies	paddleocr, easyocr, opencv-python
Preprocessing	CLAHE contrast enhancement, bilateral filtering
Validation	Regex pattern matching (2-4 letters + 4-6 digits)
Confidence Threshold	0.8 character confidence
Status	TO BUILD

Agent 4: Volume Estimator

Field	Value
Agent Name	volume_estimator
Purpose	Aggregate car counts into volume estimates with confidence intervals
Input	Sightings table queries (unique cars by location, time period)
Output	Volume estimates (gallons), confidence intervals, trends
Methodology	25,000 gal mean capacity, 5,000 gal std deviation
Dependencies	psycopg, scipy, numpy
Aggregation Periods	Hourly, daily, weekly
Confidence Level	95% CI default
Status	TO BUILD

Agent 5: Sighting Deduplicator

Field	Value
Agent Name	deduplicator

Field	Value
Purpose	Prevent duplicate counting of same car at same location
Input	New sighting events
Output	Deduplicated sighting records, frame counts
Logic	5-minute time window per (car, location) pair
Dependencies	redis (caching), postgresql
Status	TO BUILD

Runbook: First Camera Capture Setup

Prerequisites

- Python 3.10+
- pip install cap-from-youtube opencv-python
- Internet connection

Step 1: Identify Target Stream

bash

```
# Virtual Railfan La Plata MO (recommended for POC)
# Search YouTube: "Virtual Railfan La Plata" for current live stream URL
# Example format: https://www.youtube.com/watch?v=STREAM_ID
```

Step 2: Test Stream Access

python

```python
from cap_from_youtube import cap_from_youtube
import cv2

youtube_url = 'https://www.youtube.com/watch?v=YOUR_STREAM_ID'

# Test connection
cap = cap_from_youtube(youtube_url, 'best')
ret, frame = cap.read()
if ret:
    print(f"Success! Frame shape: {frame.shape}")
    cv2.imwrite('test_frame.jpg', frame)
else:
    print("Failed to capture frame")
cap.release()
```

Step 3: Continuous Capture Script

```python
import time
from datetime import datetime
from cap_from_youtube import cap_from_youtube
import cv2

youtube_url = 'YOUR_STREAM_URL'
output_dir = './frames/'
capture_interval = 1  # seconds

cap = cap_from_youtube(youtube_url, 'best')
last_capture = time.time()

while True:
    ret, frame = cap.read()
    if ret and (time.time() - last_capture) >= capture_interval:
        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        filename = f"{output_dir}frame_{timestamp}.jpg"
        cv2.imwrite(filename, frame)
        print(f"Saved: {filename}")
        last_capture = time.time()
```

Step 4: Troubleshooting

Issue	Solution
Stream not found	Verify URL is live, not a past broadcast
Low quality	Check available formats with yt-dlp -F URL
Connection drops	Add try/except with reconnection logic
No train visible	Wait or try different time (peak hours vary)

Runbook: Running Detection on Captured Frames

Prerequisites

- Python 3.10+
- pip install ultralytics easyocr opencv-python
- GPU recommended (CUDA), CPU works but slower

Step 1: Download Pre-trained Model

```python
from ultralytics import YOLO

# Downloads automatically on first use
model = YOLO('yolov8m.pt')
print("Model loaded successfully")
```

Step 2: Run Detection on Single Frame

```python
```

```python
from ultralytics import YOLO
import cv2

model = YOLO('yolov8m.pt')
frame = cv2.imread('frames/frame_20250101_120000.jpg')

results = model(frame)

# Visualize results
annotated = results[0].plot()
cv2.imwrite('detection_result.jpg', annotated)

# Print detections
for box in results[0].boxes:
    cls = results[0].names[int(box.cls)]
    conf = float(box.conf)
    print(f"Detected: {cls} ({conf:.2f})")
```

Step 3: Batch Process Directory

```python
from pathlib import Path
from ultralytics import YOLO

model = YOLO('yolov8m.pt')
frame_dir = Path('./frames/')

for frame_path in frame_dir.glob('*.jpg'):
    results = model(str(frame_path))
    # Process results...
```

Step 4: Fine-tune on Railcar Dataset (Optional)

```python
```

```python
from roboflow import Roboflow
from ultralytics import YOLO

# Download dataset
rf = Roboflow(api_key="YOUR_KEY")
project = rf.workspace("chris-immel").project("railway-cars")
dataset = project.version(1).download("yolov8")

# Train
model = YOLO('yolov8m.pt')
model.train(data=f'{dataset.location}/data.yaml', epochs=50, imgsz=640)
```

Architecture Decision Records

ADR-001: Database Selection

Field	Value
Decision	PostgreSQL with TimescaleDB extension
Status	Accepted
Context	Need to store time-series sighting data with efficient range queries
Options Considered	PostgreSQL, MongoDB, InfluxDB, TimescaleDB
Rationale	TimescaleDB provides time-series optimizations (hypertables, compression, continuous aggregates) while maintaining PostgreSQL compatibility for relational data (cars, locations, trips). MongoDB would require separate time-series handling. InfluxDB lacks relational capabilities for reference data.
Consequences	Requires TimescaleDB extension installation. Enables efficient time-bucket queries for volume aggregation.

ADR-002: Object Detection Framework

Field	Value
Decision	YOLOv8 via Ultralytics
Status	Accepted

Field	Value
Context	Need real-time object detection with tracking capability
Options Considered	YOLOv5, YOLOv8, Detectron2, SSD
Rationale	YOLOv8 offers best speed/accuracy tradeoff, built-in BoT-SORT tracking, active development, simple API. (Roboflow) Roboflow datasets pre-formatted for YOLOv8. Detectron2 more complex, SSD lower accuracy.
Consequences	Requires GPU for optimal performance. Model weights ~25MB (YOLOv8m).

ADR-003: OCR Engine Selection

Field	Value
Decision	PaddleOCR as primary, EasyOCR as fallback
Status	Accepted
Context	Need high-accuracy OCR for alphanumeric reporting marks
Options Considered	Tesseract, EasyOCR, PaddleOCR, Google Cloud Vision
Rationale	PaddleOCR achieves 97% accuracy on benchmarks, outperforming alternatives on alphanumerics. EasyOCR provides good fallback with simpler installation. Cloud APIs add latency and cost. Tesseract requires more preprocessing for comparable accuracy.
Consequences	Requires PaddlePaddle installation (~1GB). Chinese language models included by default (can be stripped).

ADR-004: Stream Processing Architecture

Field	Value
Decision	Redis Streams for frame queue, Celery for async workers
Status	Proposed
Context	Need to process multiple camera streams with detection/OCR workers

Field	Value
Options Considered	Direct processing, Redis Pub/Sub, Redis Streams, RabbitMQ, Kafka
Rationale	Redis Streams provides consumer groups for work distribution, persistence for replay, and XREAD blocking for efficient polling. Lighter than Kafka for current scale. Celery provides proven task queue with retry logic.
Consequences	Redis becomes critical infrastructure. Consider Redis Cluster for high availability at scale.

ADR-005: Camera Strategy for Production

Field	Value
Decision	Deploy proprietary cameras at strategic locations
Status	Proposed
Context	Public camera ToS prohibits commercial use; no cameras near target facilities
Options Considered	License public feeds, scrape public feeds (risk), deploy own cameras
Rationale	Public feeds explicitly prohibit commercial capture. Licensing unlikely to provide facility-specific coverage. Own cameras provide control over quality, positioning, and legal clarity. Cost ($10-30K per location) justified by data value.
Consequences	Requires property/permit negotiations. Initial deployment timeline 60-90 days per location. Consider partnerships with facility operators for mounting access.

LEARNING SYLLABUS PROMPT

Comprehensive Foundation for Building BioTrack AI

Use this prompt with Claude or another AI assistant to build foundational knowledge:

I'm building BioTrack AI, a computer vision system to monitor rail traffic for biofuel feedstock intelligence. I need to understand several technical and domain areas. Please teach me the following topics in a structured way, providing explanations, examples, and practical exercises where appropriate.

Module 1: Rail Industry Fundamentals (Week 1)

1. **Rail network structure**: Explain Class I railroads (UP, BNSF, NS, CSX, CN, CPKC), their geographic coverage, and how freight moves between them. What are interchange points and classification yards?
2. **Railcar types and identification**: Describe the major freight car types (tank cars, hoppers, boxcars, gondolas, flatcars) with visual distinguishing features. Explain the AAR reporting mark system (e.g., UTLX 123456) - what do the letters mean? What's the "X" suffix?
3. **Rail logistics for liquid commodities**: How are bulk liquids (petroleum, chemicals, biofuels, feedstocks) transported by rail? What's a unit train vs. manifest train? Typical tank car capacities?
4. **Rail data sources**: What public data exists about rail traffic? Explain the STB Waybill Sample, AAR weekly reports, and Railinc services. What are STCC codes?

Module 2: Biofuel Supply Chain (Week 1-2)

1. **Renewable diesel production**: Explain the difference between biodiesel, renewable diesel, and ethanol. What feedstocks are used (UCO, tallow, soybean oil, corn oil)? What's the HEFA process?
2. **Major facilities**: Describe Diamond Green Diesel (Norco, Port Arthur), Marathon Martinez, and ADM Decatur - what they produce, their capacity, and feedstock sources.
3. **Feedstock logistics**: How does UCO (used cooking oil) get from restaurants to renewable diesel plants? What's Darling Ingredients' role? How do feedstocks typically move (truck, rail, ship)?
4. **Market structure**: Explain RINs (Renewable Identification Numbers), LCFS credits, and how biofuel policy affects feedstock demand. Who are the major traders?

Module 3: Video Stream Capture (Week 2)

1. **Streaming protocols**: Explain HLS, DASH, and RTSP. How do YouTube live streams work technically? What's the difference between HLS and RTSP?
2. **yt-dlp and ffmpeg**: Walk me through capturing frames from a live YouTube stream. What are the key yt-dlp flags for live streams? How does ffmpeg extract frames at specific intervals?
3. **OpenCV VideoCapture**: Show me how to capture frames from a video stream in Python using OpenCV. How do you handle reconnection when streams drop?
4. **Stream buffering and latency**: What causes latency in live stream capture? How can I minimize delay between real-world events and frame availability?

Module 4: Computer Vision Basics (Week 2-3)

1. **Object detection fundamentals**: Explain the difference between image classification, object detection, and instance segmentation. What are bounding boxes? What's IoU (Intersection over Union)?
2. **YOLO architecture**: How does YOLO (You Only Look Once) work at a high level? What's the difference between YOLOv5, YOLOv8, and YOLO11? What do confidence scores and class probabilities mean?
3. **Transfer learning**: Why is pre-training on COCO useful? How do I fine-tune a YOLO model on a custom dataset? What's the typical training workflow?
4. **Object tracking**: Explain multi-object tracking algorithms like SORT and DeepSORT. How does BoT-SORT maintain object identity across frames? What's the track_buffer parameter?
5. **Hands-on exercise**: Walk me through detecting objects in an image using YOLOv8 with the Ultralytics library. Show the code to load a model, run inference, and extract bounding box coordinates.

Module 5: OCR Techniques (Week 3)

1. **OCR pipeline**: Explain the stages of optical character recognition - detection, recognition, post-processing. What's the difference between document OCR and scene text OCR?
2. **OCR engines comparison**: Compare Tesseract, EasyOCR, and PaddleOCR. What are their strengths and weaknesses? Which is best for alphanumeric text on outdoor surfaces?
3. **Image preprocessing for OCR**: What preprocessing steps improve OCR accuracy? Explain CLAHE, bilateral filtering, thresholding, and morphological operations with code examples.
4. **Handling motion blur**: How can I detect if a frame is blurry? What's the Laplacian variance method? How do I select the sharpest frame from a sequence?
5. **Hands-on exercise**: Show me how to use PaddleOCR to extract text from an image, including preprocessing steps and confidence thresholds.

Module 6: Database Design for Time-Series (Week 3-4)

1. **PostgreSQL fundamentals**: Review relational database concepts - tables, relationships, indexes. How do I design a schema for tracking entities over time?
2. **TimescaleDB**: What is TimescaleDB and why use it for time-series data? Explain hypertables, chunks, and continuous aggregates. How do I create a hypertable?
3. **Deduplication strategies**: How do I prevent counting the same railcar multiple times? What's the logic for time-window based deduplication? Show me a Python implementation.

4. **Query patterns**: Write SQL queries for: (a) counting unique cars per day at a location, (b) tracking a specific car's journey across locations, (c) aggregating volume estimates with confidence intervals.

Module 7: Pipeline Architecture (Week 4)

1. **Message queues**: Explain Redis Streams vs. Pub/Sub. How do I implement a producer-consumer pattern for frame processing? What are consumer groups?
2. **Async task processing**: How does Celery work? Show me how to define tasks for detection and OCR, and how to chain them together.
3. **Error handling and retry logic**: What happens when OCR fails? How do I implement retry with exponential backoff? How do I handle stream disconnections?
4. **Monitoring and observability**: How do I track pipeline health? What metrics should I monitor (frames processed, detection rate, OCR success rate)?

Final Project Exercise

Design and implement a minimal end-to-end pipeline that:

1. Captures one frame from a rail camera URL
2. Detects railcars using YOLOv8
3. Crops detected regions and runs OCR
4. Parses reporting marks and prints results
5. Stores the sighting in a PostgreSQL table

Provide the complete Python code with comments explaining each step.

Estimated completion time: 4 weeks at 10-15 hours/week

Prerequisites: Basic Python programming, familiarity with command line, willingness to install packages and run experiments

Resources to have ready: Google Colab account (free GPU), Roboflow account (free tier), sample rail camera video clips or screenshots