Pro-Vigil Sr. AI Engineer Interview: Sample Answers

✅ Q1: How would you detect multiple objects in a video stream?

Answer: To detect multiple objects in a video stream, I would use a real-time object detection model such as YOLOv8 or SSD. I would:

1. Access video frames via OpenCV or GStreamer.
2. Preprocess each frame (resize, normalize).
3. Run inference using the object detection model.
4. Post-process using Non-Maximum Suppression (NMS) to remove duplicate boxes.
5. Overlay bounding boxes and class labels on frames.
6. Optionally, use Deep SORT for object tracking across frames.

For speed, I would optimize using TensorRT or ONNX.

✅ Q2: What's the difference between YOLOv8 and Faster R-CNN?

Answer:

- **YOLOv8** is a one-stage detector that processes an image in a single forward pass, making it ideal for real-time applications.
- **Faster R-CNN** is a two-stage detector (Region Proposal + Classification), offering higher accuracy but slower inference.
- YOLOv8 is suitable for edge deployment due to its speed; Faster R-CNN is often better for offline or cloud use cases where precision matters more.

✅ Q3: How do you optimize inference time in video detection models?

Answer:

1. Convert model to ONNX or TensorRT for acceleration.
2. Reduce input resolution (trade-off with accuracy).
3. Use batch inference where possible.
4. Deploy using efficient APIs (e.g., FastAPI with async support).
5. Use edge devices with GPU/TPU support.
6. Skip frames (process every nth frame).
7. Profile with tools like NVIDIA Nsight or PyTorch profiler.

✅ Q4: How does temporal context help in surveillance applications?

Answer: Temporal context enables us to:

- Detect patterns over time (e.g., loitering, pacing).
- Track objects for behavior analysis.
- Reduce false positives using motion continuity.
- Detect anomalies (e.g., sudden stop or fall).

It's especially useful when combined with LSTMs or 3D CNNs.

✅ Q5: Explain background subtraction and its limitations.

Answer: Background subtraction isolates moving objects by comparing each frame to a static or dynamic background model.

Limitations:

- Fails in dynamic backgrounds (trees, water).
- Poor performance in low light or shadows.
- Cannot distinguish object classes.
- Sensitive to camera jitter.

For modern pipelines, it's usually replaced or supplemented by learned models.

✅ Q6: Have you worked with tracking algorithms?

Answer: Yes, I've used **Deep SORT** and **ByteTrack**.

- Deep SORT: Combines object detection with a Kalman filter + appearance descriptor.
- ByteTrack: Tracks both high and low confidence detections for better robustness.

I use them for re-identifying people/vehicles in crowded scenes.

✅ Q7: What is the role of BatchNorm, Dropout, and Residual Connections?

Answer:

- **BatchNorm**: Normalizes activations to stabilize and speed up training.
- **Dropout**: Prevents overfitting by randomly zeroing nodes during training.
- **Residual Connections**: Allow gradient flow in deep networks, solving vanishing gradient problems (ResNet-style).

✅ Q8: Explain Attention Mechanism in vision tasks.

Answer: Attention lets models focus on the most relevant parts of an image.

- In **ViT** (Vision Transformers), it learns pairwise relationships between patches.
- In **CNN + Self-Attention**, it enhances feature maps by weighing channels/spatial areas differently.

Benefits include better localization and feature understanding.

✅ Q9: How do you prevent overfitting with repeated frames?

Answer:

- Use **frame sampling** (skip similar frames).
- Apply **data augmentation** (brightness, crop, jitter).
- Regularize with **Dropout**, **L2**, and **early stopping**.
- Limit sequence length in temporal models.

✅ Q10: How do you deploy a model that processes live video feeds?

Answer:

1. Capture video using OpenCV or RTSP stream.
2. Run inference using a preloaded YOLO model.
3. Serve the pipeline using **FastAPI** or **Flask**.
4. Use **Docker** for containerization.
5. Deploy to edge device or GPU cloud VM.
6. Stream processed video (e.g., MJPEG) or return JSON results.

✅ Q11: How do you monitor model drift in detection systems?

Answer:

- Log confidence scores and object counts over time.
- Use **EvidentlyAI** or **custom dashboards**.
- Alert on shifts in input image stats (brightness, objects per frame).
- Regularly retrain or fine-tune on new data.

✅ Q12: What tools do you use for CI/CD in ML?

Answer:

- GitHub Actions / GitLab CI for automation.
- Docker + Makefiles for builds.
- MLflow for model versioning.
- AWS/GCP CI hooks for pushing models to production.
- Airflow/Kubeflow for orchestration.

✅ Q13: Have you containerized ML workloads?

Answer: Yes. I create Dockerfiles with:

- Python, Conda, and model dependencies
- ONNX or PyTorch models inside
- Expose APIs via FastAPI/Flask
- Use docker-compose or K8s for deployment

✅ Q14: Real-time alerting from inference output?

Answer: I push detection outputs to:

- Kafka or MQTT
- Then trigger webhook/email/SMS via backend logic (e.g., Twilio, Zapier)

Also use Redis for pub/sub in lightweight deployments.

✅ Q15: Design a system that watches live feed and sends alerts <2s

Answer: Architecture:

- Ingest RTSP feed using GStreamer/OpenCV
- Run YOLOv8 on GPU
- Alert logic (e.g., person after 9 PM) in Python
- Push to FastAPI or Kafka queue
- Send alert via webhook or SMS (async)

Ensure inference is <500ms, alert trigger <500ms.

✅ Coding Example 1: Frame Differencing Motion Detector

```
import cv2
```

```python
cap = cv2.VideoCapture(0)
ret, prev_frame = cap.read()
prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)

while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    diff = cv2.absdiff(prev_gray, gray)
    _, thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)
    cv2.imshow("Motion", thresh)
    prev_gray = gray
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

✅ **Behavioral: Tell us about a time your model failed in production.**

Answer: In a vehicle detection pipeline, the model misclassified shadows as vehicles due to data bias. I resolved it by:

- Adding nighttime & shadowed examples
- Using CLAHE for contrast enhancement
- Retraining and monitoring post-deployment

✅ **Behavioral: How do you trade off speed vs. accuracy?**

Answer: Depends on use-case. For real-time surveillance, I:

- Use lightweight models (YOLO-tiny, MobileNet)
- Adjust input resolution
- Calibrate confidence thresholds

In some cases, I ensemble two models (fast + slow) and defer to the slower one when uncertainty is high.

✅ **Behavioral: How do you mentor junior engineers?**

Answer:

- Pair programming sessions weekly
- Use code reviews as teaching opportunities
- Assign small independent projects with structured feedback
- Encourage them to demo work regularly and ask questions

✅ **Behavioral: Have you worked with edge devices?**

Answer: Yes, deployed YOLOv5 on NVIDIA Jetson Nano. Challenges included:

- Memory optimization
- Using TensorRT for acceleration
- Managing thermal throttling

I packaged the model using Docker and monitored via a lightweight dashboard.
