System Architecture

Overall Layered View (high-level)

1. **Device / Input Layer**
    - Webcam (USB / built-in) or video source (file/RTSP)
2. **Application / Presentation Layer**
    - Streamlit web UI (demo_application/main.py)
    - CLI visualization (model_visualization.py)
3. **Detection & Inference Layer**
    - CardGameDetector (in demo_application/utils/card_game_detector.py)
    - YOLOv8 model (.pt files in final_models/)
4. **Post-processing & Business Logic**
    - Aggregation, parsing (class → rank/suit), formatting results
    - Game logic (optional legacy game_logic.py)
5. **Model Training & Data Layer**
    - Datasets (synthetic, real, augmented) under data/
    - Training/validation scripts in model_utils/ (train.py, val.py, predict.py)
6. **Storage & Artifacts**
    - Model weights (final_models/)
    - Training runs & metrics (runs/)
    - Logs, test images, presentations, docs
7. **Ops / Deployment**
    - Local desktop (Streamlit), containerized service, or cloud deployment
    - GPU acceleration (PyTorch + CUDA) or CPU fallback

Component Breakdown (detailed)

1) Input / Capture Subsystem

- **Components**
  - cv2.VideoCapture instance (configured in main.py).
  - Camera configuration: resolution 640×480, camera index 0 by default.
- **Responsibilities**
  - Acquire frames at target FPS.
  - Provide stable preview frame to UI (avoiding double-read).
  - Allow alternate inputs (video file, RTSP) for testing.
- **Notes**
  - Capable of switching camera indices for multi-camera support.
  - Should expose camera health (isOpened) and fallback logic.

2) Presentation Layer (Streamlit)

- **Components**
  - demo_application/main.py (UI layout, buttons, session state).
  - Media / assets under demo_application/media/.
- **Responsibilities**
  - Provide controls: *Take Snapshot*, *Clear Results*, model selection.
  - Display live preview, detected cards list, counts and confidence.
  - Manage user permissions (webcam access) and session state.
- **Design Considerations**
  - Keep inference off the UI thread (use worker thread or subprocess).
  - Cache loaded model across requests to avoid reload latency.

3) Detection / Inference Layer (CardGameDetector)

- **Components**
  - CardGameDetector class with methods:
    - detect_on_frame(frame) — inference wrapper around YOLOv8.
    - aggregate_detections(detections_list) — multi-frame voting/aggregation.

        - parse_cards(class_ids) — map model class IDs to human-readable cards.

- **Model**
    - YOLOv8 Medium: yolov8m_synthetic.pt (52 classes) as default.
    - Alternative models: yolov8m_tuned.pt, yolov8m_real.pt, yolov8m_aug.pt.
- **Inference Flow**

1. Accept preprocessed frame (resize/pad to MODEL_INPUT_SIZE).
2. Run model inference (PyTorch/Ultralytics API).
3. Post-filter by confidence threshold (0.15).
4. Append detections to per-snapshot buffer (NUM_FRAMES = 10).
5. Aggregate (occurrence count ≥ AGGREGATION_THRESHOLD).
6. Parse to ["10♠", "Ah♥"] formatted output.

- **Performance**
    - Should utilize GPU if available; otherwise CPU path used.
    - Keep model loaded in memory (singleton/cached) with thread-safe access.

4) Post-Processing & Aggregation

- **Functions**
    - Non-max suppression (model does this; tune if needed).
    - Confidence-based filtering and optional class-specific thresholds.
    - Multi-frame voting: require repeated detections across NUM_FRAMES.
- **Outputs**
    - Final list of detected cards, counts, per-card confidence, bounding boxes.
    - Displayable string and optional JSON payload for downstream systems.

5) Configuration & Constants

- Located in demo_application/utils/constants.py:
    - MODEL_PATH, CLASS_NAMES, CONFIDENCE_THRESHOLD, AGGREGATION_THRESHOLD, NUM_FRAMES, IMAGE_SIZE, MODEL_INPUT_SIZE.
- **Best Practice**
    - Support environment-variable overrides and CLI flags.

    - Provide a config.yaml for deployment-time configuration.

6) Training & Dataset Pipeline

- **Dataset Types**
    - Synthetic (20k), Real (100), Augmented (1,000), Combined (1,100).
- **Scripts & Tools**
    - dataset_utils/augment_dataset.ipynb, combine_datasets.py, transform_labels_in_dateset.py.
    - Model training: model_utils/train.py (uses Ultralytics/PyTorch API).
- **Outputs**
    - Trained .pt weights in final_models/, logs in runs/.
- **CI / Repro**
    - Use requirements.txt, venv, lockfile, and Dockerfile for deterministic builds.

7) CLI & Debug Tools

- model_visualization.py — run inference locally with keyboard controls (Q quit, S toggle labels).
- Test scripts: test_connectivity.py, test_detection_debug.py, verify_setup.py.

Deployment & Ops Options

Local Desktop (current default)

- **Run:** streamlit run demo_application/main.py
- **Pros:** Fast to iterate; minimal infra.
- **Cons:** Single-user; limited scaling.

Containerized Service (recommended for reproducibility)

- **Dockerfile**
    - Base image: CUDA-enabled PyTorch image or Python slim for CPU.
    - Expose a REST API (Flask/FastAPI) wrapping CardGameDetector for programmatic access, and host Streamlit behind a reverse proxy if needed.
- **Advantages**
    - Easier dependency management, reproducible deployments, CI/CD integration.

Cloud Deployment (scalable, multi-user)

- **Options**
  - Host inference on GPU instances (AWS/GCP/Azure) and serve UI separately (S3 + Cloud Run).
  - Use a serverless or managed Kubernetes cluster for autoscaling.
- **Pattern**
  - Inference microservice (GPU-backed) ←→ Streamlit or web UI (lightweight).
  - Use message queue (RabbitMQ/Kafka) if ingesting many video streams concurrently.

Edge Deployment

- **For embedded / mobile**
  - Convert YOLOv8 model to ONNX → TensorRT / TFLite (quantize).
  - Run inference on Jetson / Coral / mobile GPU for low-latency local inference.

Performance Targets & Monitoring

Suggested Targets

- **Latency:** <50–120 ms per frame on GPU (depending on model size)
- **Throughput:** ≥10 snapshot inferences / second aggregated across frames
- **Accuracy:** Aim for >95% real-world card recognition after augmentation and combined training

Monitoring & Metrics

- Inference latency, model load times, GPU utilization, frame drop rate, accuracy per class, false positive rate.
- Save metrics/logs to runs/ and push critical metrics to Prometheus + Grafana for production.

Security, Privacy & Compliance

- **Camera Permissions:** Streamlit relies on browser permission prompts; ensure secure contexts (HTTPS) in deployment.
- **Data Handling:** If storing images or logs, anonymize or secure PII and comply with data regulations (GDPR) if applicable.
- **Model Integrity:** Protect model artifacts (signed checksums) and restrict write access to final_models/.

- **Surface for Abuse:** Limit API access via auth tokens/rate limits in cloud deployments.

Failure Modes & Mitigations

1. **No Card Detected**
   - Cause: poor lighting, occlusion, wrong camera index.
   - Mitigation: show user-friendly hints (lighting, distance), fallback to manual input.
2. **High False Positives**
   - Cause: low confidence threshold; noisy backgrounds.
   - Mitigation: allow per-class thresholds, increase aggregation threshold, or add background negative samples in training.
3. **Slow Inference / UI Freeze**
   - Cause: model loaded on UI thread or CPU-only inference.
   - Mitigation: move inference to worker thread/process; enable GPU; use smaller model (YOLOv8n) for low-power devices.
4. **Model File Not Found**
   - Mitigation: robust path resolution, absolute path option, and clear error messages from verify_setup.py.

Extensibility & Future Enhancements (architectural notes)

- **Multi-Camera Support**: abstract Capture interface to plug multiple sources; aggregate cross-camera detections.
- **REST API**: add FastAPI wrapper to expose JSON endpoints (/detect, /health, /model/info).
- **Streaming Inference**: implement sliding-window detection and event-driven snapshots.
- **Auto-Retraining Pipeline**: collect edge-case false detections and automate labeling + retraining (CI triggered).
- **Model Compression**: pipeline for ONNX → TensorRT → quantized runtime for edge.

Implementation Checklist (practical tasks)

- Ensure MODEL_PATH is configurable via env vars.
- Move inference to separate worker (thread/process) and keep model cached.
- Add health endpoints and metrics export (Prometheus).
- Provide Dockerfile with both CPU and GPU variants.
- Add automated test suite for CardGameDetector (unit + integration using sample test images).
- Add CI pipeline to validate verify_setup.py and run smoke tests.