Architectural Design Document: Computer Vision Service (Variation X3)

Version: 0.1 (First Draft)
Authors: Albert Aksenov
Date: April 15, 2025

1. Introduction

This document presents the first draft of the architectural design for a Computer Vision Service, as part of the course project requirements. The goal is to design a system capable of processing various video inputs (live streams, files, images) using configurable computer vision algorithms, specifically tailored for deployment on resource-constrained Single-Board Computers (SBCs) with challenging network conditions.

This draft outlines the project context, identifies key architectural drivers, discusses initial design considerations, and proposes a preliminary architecture based on the project description (Variation X3) and course guidelines. The target audience includes technical stakeholders such as the development team and technically proficient management.

2. Project Description (Variation X3)

(Copied from Course Project Variations document)

Goal description
You are to design a computer vision service. The purpose of this service is to process video streams, pictures, or video files with computer vision algorithms. Processing stages are different and should be easily configured for different kinds of products.

Users can submit RTSP streams to process them in real-time. In this case, the user should get a continuous video stream with a lag of no more than 1 second between source and processed streams. If users submit videos or pictures to be processed, they should get back processed results.

Service should be able to notify external services in case of detecting a specific condition.
Every processed frame should have detection/tracking markers (boxes, paths, etc.) drawn on top of the frame as an overlay.

We expect videos to be 1080p at a maximum and use an h.264 codec.

Context modifier
The system will be deployed on single-board computers (SBC) with a slow and unreliable internet connection. Single-board computers often lack GPUs. The marketing team expects high demand and shipping of 10-50 SBCs in the first month and 100-200 each month after that.

3. Project Context

The project involves designing a specialized Computer Vision (CV) service intended for edge deployment. The core business goal is to provide a flexible CV processing solution that can operate effectively on low-power SBCs, scale to hundreds or thousands of devices, and function reliably despite poor internet connectivity.

Domain: Edge Computing, Computer Vision, Video Processing, IoT (potentially).

Business Goals:

- Provide a configurable CV platform adaptable to different product needs.
- Enable real-time video analysis on edge devices.
- Support batch processing of images and video files.
- Achieve significant market penetration with rapid scaling of deployed SBCs (100-200/month).
- Ensure service operation despite network limitations inherent to edge deployments.

Primary Stakeholders:

- **End-Users:** Submit media (RTSP streams, files) for processing and receive results (processed streams, files). Expect low latency for real-time streams and timely results for batch jobs.
- **System Administrators/Operations:** Deploy, manage, monitor, and update the software across a large fleet of distributed SBCs. Require robust deployment mechanisms and effective monitoring tools.

- **Development Team:** Implement, test, and maintain the service. Require a clear, modular architecture that is easy to work with and extend.
- **Product/Marketing Team:** Define product configurations (CV pipelines) and rely on the system's scalability and reliability to meet market demand.
- **External Services:** Receive notifications based on specific detection events. Require a reliable notification mechanism.

4. Assumptions

Given the concise project description, several assumptions are necessary:

1. **SBC Specifications:** Assume typical SBCs like Raspberry Pi 4/5 or similar, implying ARM CPUs (4-8 cores), limited RAM (e.g., 4-8 GB), no dedicated GPU, and standard network interfaces (Ethernet, Wi-Fi). *Verification: Obtain target SBC hardware specifications.*
2. **CV Algorithm Complexity:** Assume the required CV algorithms (detection, tracking, etc.) are computationally feasible on the target SBC CPUs within the latency requirements, potentially requiring model optimization or lighter-weight algorithms. *Verification: Benchmark candidate CV algorithms on target SBCs.*
3. **Configuration Mechanism:** Assume configuration involves defining the sequence of CV algorithms, their parameters, and notification conditions/endpoints. This configuration needs to be updatable remotely. *Verification: Define configuration schema and update mechanism with stakeholders.*
4. **"Processed Results" Format:** Assume results for files/images include the processed media (with overlays) and potentially structured data (e.g., JSON list of detections). *Verification: Define output formats with stakeholders.*
5. **Notification Protocol:** Assume standard protocols like HTTP/HTTPS webhooks or MQTT are acceptable for notifications. The system must handle potential delivery failures due to network issues. *Verification: Define notification requirements (protocol, payload, reliability) with stakeholders integrating external services.*
6. **Network Characteristics:** Assume "slow and unreliable" means low bandwidth, high latency, and intermittent connectivity outages. *Verification: Characterize typical network conditions in deployment environments.*
7. **Acceptable Quality Trade-offs:** Assume some trade-offs between processing speed, accuracy, and resource consumption are acceptable (e.g., processing at a lower resolution or frame rate if needed to meet latency goals). *Verification: Define acceptable quality thresholds with product teams.*

8. **Security:** Assume standard security practices for edge devices (secure boot, encrypted communication, access control) are required, though not explicitly stated. *Verification: Define security requirements with stakeholders.*

5. Architectural Drivers

Based on the goals, context, and assumptions, we derive the following architectural drivers:

5.1. Key Functionality

- F1: Ingest RTSP video streams.
- F2: Ingest video files (H.264, <=1080p).
- F3: Ingest picture files.
- F4: Process inputs using a configurable pipeline of CV algorithms.
- F5: Output processed RTSP stream with overlays (<1s end-to-end lag).
- F6: Output processed video/picture files with overlays.
- F7: Draw detection/tracking markers (boxes, paths) as overlays.
- F8: Detect specific conditions based on CV results.
- F9: Notify external services upon condition detection.
- F10: Allow remote configuration/updates of the CV pipeline.

5.2. Quality Attributes (Prioritized)

Quality attribute scenarios help define measurable targets:

1. **Performance (Latency):**
    - *Scenario:* An RTSP stream is submitted under normal operating conditions on a target SBC. *Response:* The processed stream with overlays is available to the user with less than 1 second delay from the source frame capture time. (High Priority)
    - *Scenario:* A 1-minute, 1080p, H.264 video file is submitted for processing with a standard detection pipeline. *Response:* Processing completes within X minutes (TBD based on benchmarks). (Medium Priority)
2. **Reliability:**
    - *Scenario:* Network connection is lost for 5 minutes while processing and a notification condition is met. *Response:* The notification is queued locally and sent successfully once connectivity is restored. No data (frames/detections) relevant to the notification is lost. (High Priority)

    - *Scenario:* A CV processing module crashes unexpectedly. *Response:* The service attempts to restart the module automatically. If unsuccessful, the failure is logged, and the overall service continues operating if possible (e.g., processing other streams/files). (High Priority)
3. **Deployability / Manageability:**
    - *Scenario:* A new version of a CV algorithm needs to be deployed to 200 active SBCs. *Response:* The update process can be initiated remotely and completes within Y hours (TBD) with minimal manual intervention per device, providing status feedback. (High Priority, due to scale)
    - *Scenario:* An operator needs to check the health status (CPU, memory, service status) of a specific SBC. *Response:* Status information is available via a central monitoring system within Z minutes (TBD) of real-time. (High Priority)
4. **Modifiability / Configurability:**
    - *Scenario:* A new product requires a different sequence of CV algorithms (e.g., face detection followed by pose estimation instead of object tracking). *Response:* A developer can create and deploy this new configuration to target SBCs within hours, without code changes to the core framework. (High Priority)
5. **Resource Efficiency:**
    - *Scenario:* The CV service runs continuously on a target SBC. *Response:* CPU and RAM usage remain within the limits defined by the SBC specifications, allowing the OS and potentially other essential services to run. (High Priority, constraint-driven)
6. **Scalability:**
    - *Scenario:* The number of deployed SBCs increases from 50 to 500 over six months. *Response:* The deployment, management, and monitoring infrastructure handles the increased load without significant architectural changes or performance degradation. (Medium Priority - relates to central management, less to the SBC software itself)

5.3. Constraints

- C1: Deployment target is SBCs (limited CPU, RAM).
- C2: GPUs are generally unavailable.
- C3: Network connection is slow and unreliable.
- C4: Input video limited to H.264, <= 1080p.
- C5: High volume deployment target (100-200 SBCs/month).

- C6: Project deadlines (Draft April 15th, Final April 30th, Defense May 14th).

6. Approaches for Proving Response Measures

Verifying the quality attribute responses will require:

- **Benchmarking:** Running candidate CV algorithms, decoding/encoding processes, and full pipelines on target SBC hardware to measure latency, throughput, and resource usage.
- **Network Simulation:** Testing system behavior (especially notifications, updates) under simulated conditions of low bandwidth, high latency, and packet loss.
- **Stress Testing:** Simulating high load conditions (e.g., multiple concurrent streams/files if supported by the SBC) to identify bottlenecks.
- **Failure Injection:** Intentionally crashing components or simulating network outages to test recovery mechanisms (e.g., restart logic, notification queuing).
- **Deployment Testing:** Validating the remote deployment and configuration update process on a test set of SBCs.

7. Design Process

This architectural design primarily follows the principles of **Attribute-Driven Design (ADD)**. The process involves:

1. Identifying architectural drivers (functionality, quality attributes, constraints) - Done (Section 5).
2. Choosing architectural tactics and patterns to satisfy these drivers.
3. Structuring the system into components and defining their responsibilities and interactions (views).
4. Analyzing the proposed design against the drivers and refining it iteratively.

This draft represents the initial steps of this process.

8. Proposed Architecture (First Draft)

Given the constraints (SBC resources, network unreliability) and key drivers (performance, reliability, deployability, configurability), an **edge-centric architecture** is necessary. Most processing must occur directly on the SBC. A central management/monitoring component is likely needed for scalability but is secondary to

the core SBC application.

8.1. Architectural Style/Patterns

- **Pipe and Filter:** Suitable for the core video processing pipeline, where data flows through a series of independent processing stages (decode -> CV algorithm 1 -> ... -> overlay -> encode/display). This supports configurability (F4) and modifiability (QA4).
- **Microservices (on the Edge):** While perhaps too heavyweight for a single SBC, the *principles* apply: breaking down the SBC application into distinct, independently manageable components (e.g., Input Handler, Pipeline Manager, Notifier, Config Manager, Health Monitor). This aids modifiability and potentially reliability (isolating failures).
- **Layered Architecture:** Could structure the SBC application (e.g., Input/Output Layer, Processing Layer, Management Layer).

8.2. Key Components (Conceptual - On SBC)

1. **Input Manager:**
    - Responsibility: Handles incoming RTSP streams, file uploads (video/images). Validates input types.
    - Interfaces: Network sockets (RTSP), file system access.
2. **Video Decoder:**
    - Responsibility: Decodes H.264 video frames efficiently using CPU.
    - Interfaces: Input Manager, Processing Pipeline.
    - Tactics: Hardware-accelerated decoding if available (though unlikely based on C2), optimized software decoding libraries (e.g., FFmpeg, GStreamer).
3. **Processing Pipeline Executor:**
    - Responsibility: Manages the sequence of CV processing steps based on the current configuration. Executes CV Modules.
    - Interfaces: Video Decoder, CV Modules, Overlay Renderer, Configuration Manager.
    - Tactics: Dynamic loading of modules, efficient data passing between stages (e.g., shared memory or efficient serialization).
4. **CV Modules (Filters):**
    - Responsibility: Perform specific CV tasks (e.g., object detection, tracking). Must be CPU-optimized.
    - Interfaces: Processing Pipeline Executor.

    - Tactics: Use lightweight models (e.g., TensorFlow Lite, ONNX Runtime for CPU), model quantization, algorithm optimization. Examples: OpenCV functions, custom models.
5. **Overlay Renderer:**
    - Responsibility: Draws detection markers (boxes, paths) onto video frames.
    - Interfaces: Processing Pipeline Executor, Output Manager.
6. **Output Manager:**
    - Responsibility: Encodes processed frames (if needed for streaming) and handles output (streaming back via RTSP-like protocol or saving to file).
    - Interfaces: Overlay Renderer, Network sockets, File system.
7. **Condition Detector & Notifier:**
    - Responsibility: Evaluates CV results against configured conditions. If a condition is met, queues and sends notifications to external services.
    - Interfaces: Processing Pipeline Executor, Configuration Manager, Network Interface.
    - Tactics: Persistent queue for notifications (to handle network outages - QA2), retry mechanisms with backoff. MQTT could be suitable for unreliable networks.
8. **Configuration Manager:**
    - Responsibility: Stores and provides the current pipeline configuration. Handles updates received from the central management service (if present) or via a local mechanism.
    - Interfaces: Processing Pipeline Executor, Condition Detector, Management Agent.
9. **Management Agent / Health Monitor:**
    - Responsibility: Monitors SBC resource usage (CPU, RAM) and service health. Reports status and receives commands (e.g., updates, restarts) potentially from a central service.
    - Interfaces: OS APIs, Configuration Manager, Network Interface.
    - Tactics: Heartbeating, logging.

8.3. Preliminary Views (Conceptual Static View)

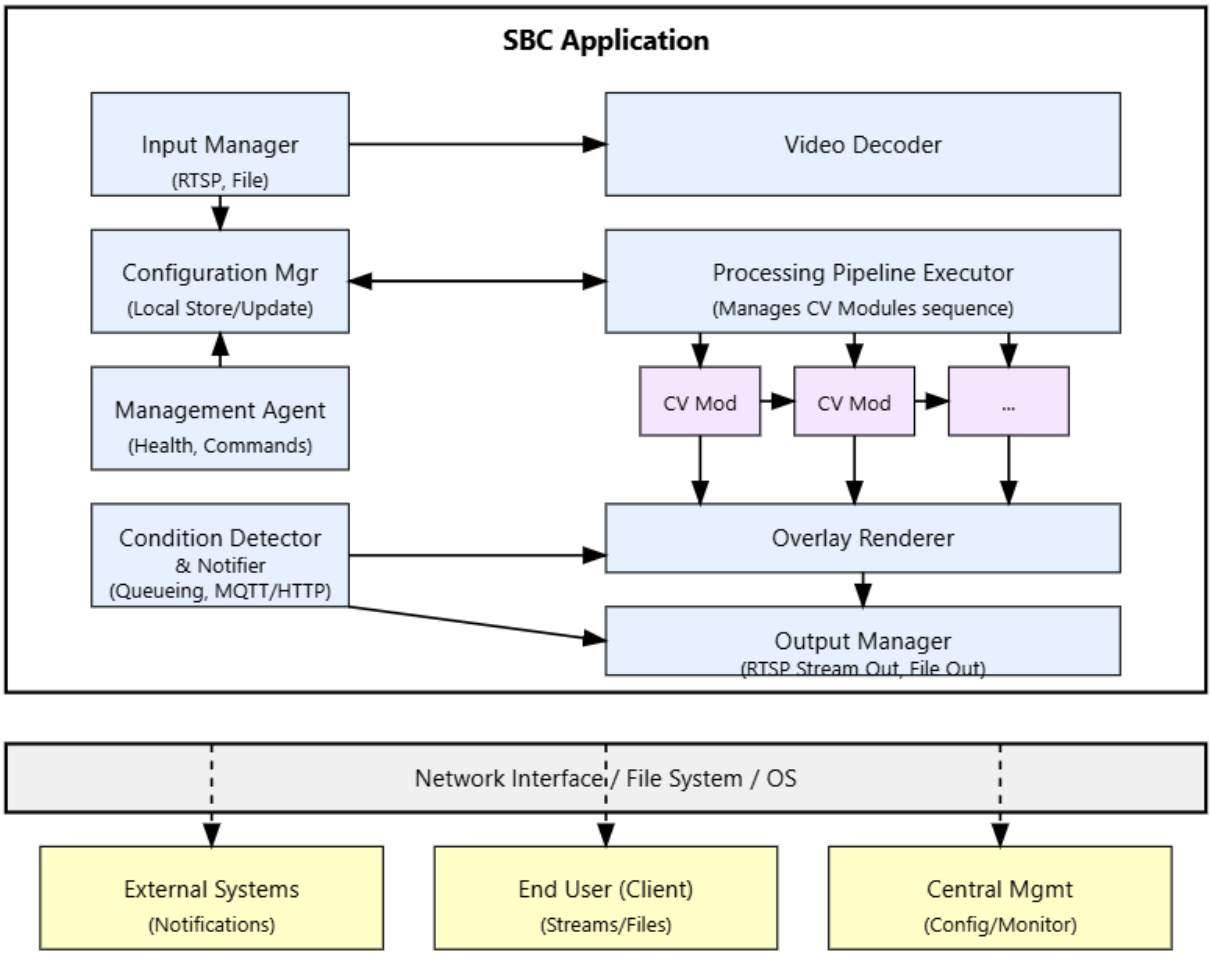

Diagram Key: Arrows indicate primary data/control flow. Dashed lines to external/central systems highlight reliance on the (unreliable) network.

8.4. Technology Considerations (Initial Ideas)

- **Language:** Python (strong CV/ML ecosystem, rapid development) or C++ (performance critical components).
- **CV Libraries:** OpenCV (general purpose), ONNX Runtime (optimized inference on CPU).
- **Streaming/Decoding:** GStreamer (flexible pipeline framework) or FFmpeg libraries.
- **Notifications:** MQTT (lightweight pub/sub for unreliable networks) or HTTPS (if endpoints support it).
- **Deployment:** Docker (containerization simplifies dependency management and deployment - addresses QA3).

- **Central Management (if built):** Could use IoT platforms, standard web frameworks, or custom solutions.

9. Architectural Analysis (First Draft)

This initial design attempts to address the key drivers:

- **Performance (Latency):** Processing is local to the SBC, minimizing network latency (QA1). CPU optimization of CV modules and efficient pipelines (Pipe and Filter) are crucial tactics. Potential challenge: achieving <1s on complex pipelines/1080p video with CPU limits. *Trade-off:* May need to reduce resolution, frame rate, or model complexity.
- **Reliability:** The Notifier's queue handles network outages (QA2). Component separation (Microservices principle) can help isolate failures. A robust Management Agent with restart capabilities is needed. **Trade-off:** Increased complexity for queueing and state management.
- **Deployability/Manageability:** Containerization (Docker) and a Management Agent reporting to a central system address scaling and remote updates (QA3, C5). **Trade-off:** Requires infrastructure for container orchestration/updates and central monitoring.
- **Modifiability/Configurability:** The Pipe and Filter pattern combined with a Configuration Manager directly supports dynamic pipelines (QA4, F4). **Trade-off:** Requires careful design of module interfaces and configuration schema.
- **Resource Efficiency:** Explicit focus on CPU optimization and lightweight technologies is required (QA5, C1, C2). **Trade-off:** Development effort for optimization; potential accuracy reduction.

10. Alternative Decisions / Future Considerations

Alternative Processing Framework: Instead of a custom Pipe and Filter executor, leverage GStreamer more heavily for the entire pipeline, including CV plugin integration.
Inter-Component Communication: On the SBC, explore options like DBus, gRPC, or shared memory for communication between components if implemented as separate processes instead of threads/libraries.
Central vs. Fully Decentralized: Could the system operate entirely without a central management server (relying on manual config updates)? Less scalable and manageable, but potentially more resilient to total network failure.
Language Choice: Python vs. C++ involves trade-offs between development speed/ecosystem and raw performance/resource control. A hybrid approach might be

feasible.

11. Conclusion (Draft)

This first draft outlines an edge-centric architecture for the Computer Vision Service. It prioritizes performance, reliability, and manageability under the significant constraints of SBC deployment and unreliable networking. The proposed design uses patterns like Pipe and Filter and leverages tactics such as local processing, notification queuing, containerization, and CPU optimization.

Further work is needed to refine component interactions, define specific APIs and data formats, select concrete technologies, perform feasibility analysis (benchmarking), and develop more detailed architectural views (dynamic, physical).