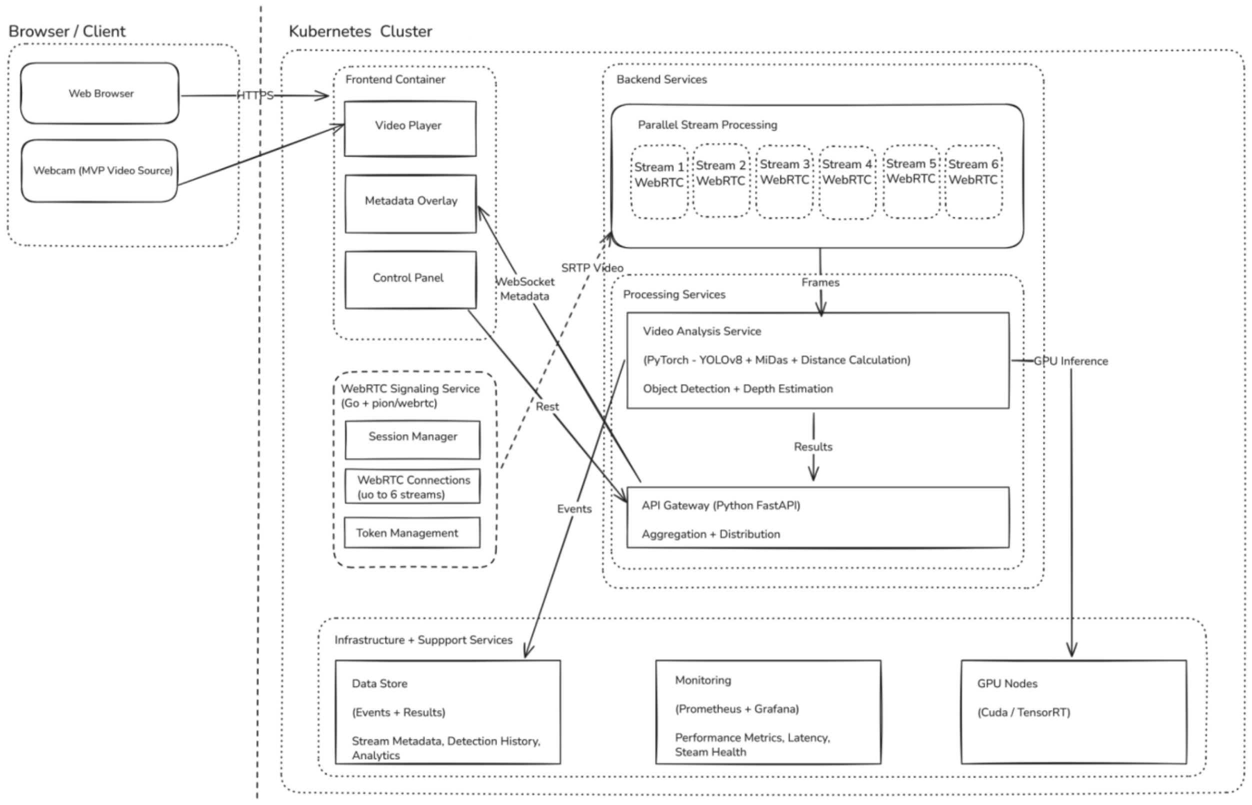
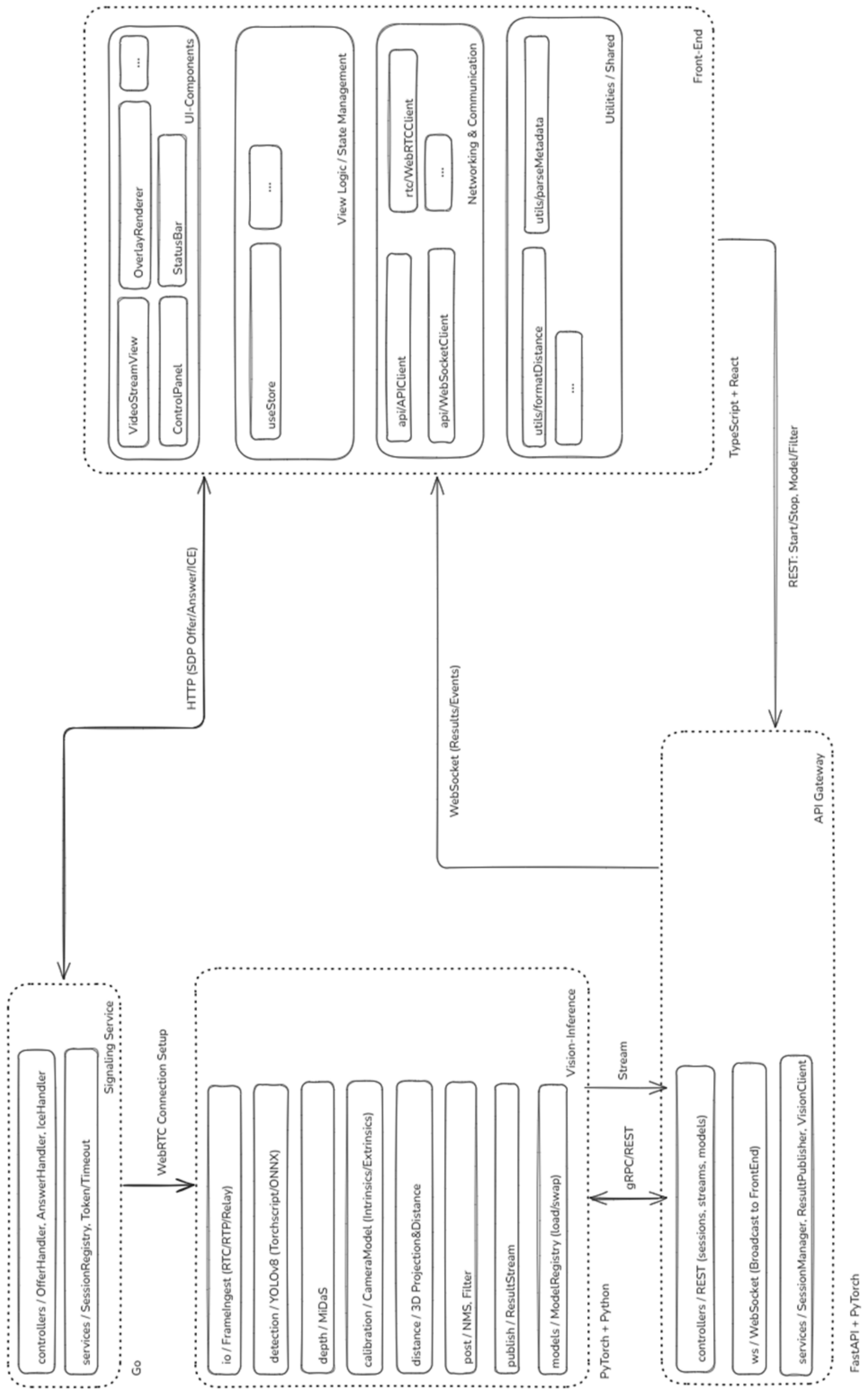
Page 1: Overview diagram of runtime components



Page 2: Overview diagram of code components


Front-End
TypeScript + React
UI-Components
VideoStreamView
ControlPanel
OverlayRenderer
StatusBar
...
View Logic / State Management
useStore
...
Networking & Communication
api/APIClient
api/WebSocketClient
rtc/WebRTCClient
...
Utilities / Shared
utils/formatDistance
...
utils/parseMetadata
HTTP (SDP Offer/Answer/ICE)
WebSocket (Results/Events)
REST: Start/Stop, Model/Filter
Signaling Service
Go
controllers / OfferHandler, AnswerHandler, IceHandler
services / SessionRegistry, Token/Timeout
WebRTC Connection Setup
Vision-Inference
PyTorch + Python
io / FrameIngest (RTC/RTP/Relay)
detection / YOLOv8 (Torchscript/ONNX)
depth / MiDaS
calibration / CameraModel (Intrinsics/Extrinsics)
distance / 3D Projection&Distance
post / NMS, Filter
publish / ResultStream
models / ModelRegistry (load/swap)
Stream
gRPC/REST
API Gateway
FastAPI + PyTorch
controllers / REST (sessions, streams, models)
ws / WebSocket (Broadcast to FrontEnd)
services / SessionManager, ResultPublisher, VisionClient

Page 3: Summary of the underlying technology stack

The *Robot Visual Perception* system is built on a modern, modular technology stack designed for real-time object detection, depth estimation, and distance calculation in video streams captured by mobile robots. Its goal is to analyze camera images in under 100 milliseconds and provide operators with precise spatial information about detected objects through a web-based interface.

The frontend is developed using React and TypeScript, leveraging WebRTC to display live video streams in the browser with minimal latency. It receives metadata such as object positions and distances via a WebSocket connection, rendering bounding boxes and distance labels as real-time overlays on the video stream using an HTML5 Canvas. A REST-based interface allows users to configure filters and control commands.

The backend acts as a communication layer between the frontend, streaming infrastructure, and AI analysis service. Implemented in Python with FastAPI, it manages signaling, stream routing, and aggregation of analysis results. A Go-based WebRTC signaling service built on *pion/webrtc* handles the low-latency video transmission. Processed metadata from the AI service is transmitted via gRPC or WebSocket, ensuring seamless real-time communication across components.

The core image analysis is handled by a dedicated Image Analysis Service, implemented in Python with PyTorch, ONNX Runtime, or TensorRT for GPU acceleration. It uses YOLOv8 for object detection and MiDaS for monocular depth estimation, combining both outputs to compute the spatial coordinates and distance of each detected object. The results are sent back to the backend as structured JSON messages for visualization.

All components are containerized using Docker and deployed in a Kubernetes environment for scalability and reliability. GPU-enabled nodes handle inference workloads, while tools such as Prometheus and Grafana provide monitoring and performance metrics. Data privacy is ensured through GDPR-compliant handling: no raw video data is stored, only transient frame processing is performed. Secure communication channels (TLS) and token-based authentication (JWT or OAuth 2.0) protect all data exchanges.

Together, this technology stack integrates modern web technologies, real-time streaming, and edge AI into a cloud-ready architecture that enables precise, low-latency object recognition and distance estimation for robotic perception systems.

Page 4: Textual explanation of the diagrams and choices

The *Robot Visual Perception* system follows a modular, containerized architecture designed for real-time object detection, depth estimation, and distance calculation in video streams. The code and runtime diagrams illustrate the separation between four main components: the frontend, backend, streaming service, and AI analysis module.

The frontend, built with React and TypeScript, connects via WebRTC to display low-latency video streams directly in the browser. It receives real-time detection results through WebSocket and overlays bounding boxes and distance labels on the video feed. WebRTC was chosen for its sub-100 ms latency and adaptive streaming, critical for teleoperation scenarios.

The backend, implemented in Python (FastAPI), manages communication between the frontend, streaming layer, and analysis service. It exposes REST and WebSocket APIs, aggregates inference results, and coordinates multiple video streams. FastAPI was selected for its modular design and support for scalable APIs.

The WebRTC signaling service, written in Go with the *pion/webrtc* library, handles session negotiation and real-time media transmission. Go's concurrency model enables efficient handling of multiple parallel streams. Separating signaling logic into its own service ensures low latency and fault isolation.

The image analysis service, developed in Python, performs the core AI processing using YOLOv8 for object detection and MiDaS for monocular depth estimation. Accelerated by PyTorch, ONNX Runtime, or TensorRT, it calculates 3D object positions and distances, returning structured JSON data to the backend. GPU-based inference and asynchronous processing guarantee responsiveness within the 100 ms target.

All components run in Docker containers orchestrated by Kubernetes, ensuring portability and scalability. Prometheus and Grafana monitor performance and latency, while ELK Stack provides centralized logging. Communication between modules uses WebRTC for video and gRPC/WebSocket for metadata, ensuring secure, low-latency data flow.

This architecture enables modular development, cloud scalability, and real-time visual perception for robotic applications – balancing performance, flexibility, and maintainability.