SmartPole — Unified Intelligent Camera Platform

Comprehensive design, blueprints, sample code, deployment artifacts, and documentation for a globally-adaptable, jurisdiction-aware intelligent camera platform intended to run on traffic poles and similar infrastructure. This document is modular — pick a section and use the included manifests/snippets to bootstrap real implementation.

Table of Contents

1. Executive summary
2. System architecture diagrams (text diagrams)
3. High-level module list and interfaces
4. Hardware Abstraction Layer (HAL) design + sample driver API
5. AI/ML pipeline architecture and sample code
6. Policy & Governance Engine + DSL examples
7. Data Fusion & Event Model (schemas + JSON Schema)
8. APIs, SDK, and OpenAPI spec (summary + sample endpoints)
9. Security & Privacy design (controls, key management, secure boot)
10. Fleet management & OTA / DMS
11. Deployment strategy (small / medium / large), containers, K8s
12. Data governance and compliance docs (sample retention matrix)
13. Test cases & simulation scenarios
14. Security audit plan & risk register
15. Open source vs proprietary breakdown
16. Modular component manifest (pluggable functions list)
17. Source tree structure and installation scripts
18. Model manifest and packaging guidance
19. Appendix: sample configuration files, Dockerfile, K8s skeletons, example Python snippets

---

1. Executive summary

SmartPole is a single-device, multi-purpose camera+compute platform built to perform real-time traffic & urban intelligence while enforcing privacy and jurisdictional policy. It is edge-first, cloud-supported, and designed for hot-swappable sensors, modular AI, and multi-agency access.

Design principles: modularity, policy-compliance, zero-trust security, observability, graceful degradation, and future expandability.

Target runtime: Linux-based edge OS (Ubuntu Core, Yocto), containers (containerd), optional NVIDIA Jetson or Coral accelerators, or x86 server-class edge nodes.

2. System architecture (text diagrams)

```
+-------------------------------+            +----------------------------------
+
|         Edge Device           |            |        Cloud/Regional Cluster
|
|  (SmartPole: camera + compute)|            |(Aggregation, long-term storage)
|
|                               |            |
|
|  +-------------------------+  |   TLS/Mutual | +----------------------------
+   |
|  | HAL / Sensor Manager    |<==============>| | Fleet Manager / DMS        |
|
|  +-------------------------+  |   Auth      | +----------------------------+
|
|  | AI Engine (ONNX/TFLite) |  |             | + Policy DB & Simulation   |   |
|  +-------------------------+  |             | + Model Registry            |
|
|  | Data Fusion & Event Buff|  |  Message    | + Event Bus (Kafka/Redis)  |   |
|  +-------------------------+  |  Broker     | + Long-term storage (S3)   |   |
|  | Policy Engine (local)   |  |  MQTT/AMQP  | + Analytics / OLAP         |   |
|  +-------------------------+  |             | + RBAC / IAM / Audit logs  |   |
|  | Secure Layer (TPM, VPN) |  |             | + Public/Open Data APIs    |   |
|  +-------------------------+  |             |                            |   |
+-------------------------------+            +----------------------------------
+
```

Message flows: - Telemetry & health -> DMS via MQTT or gRPC - Events and anonymized metrics -> regional event bus (Kafka) - Raw encrypted clips -> Cloud cold storage (S3, with retention & access controls)

3. High-level module list & interfaces

- hal/ : sensor detection, calibration, diagnostic APIs
- ai/ : model runner, pipeline orchestrator, model manager
- fusion/ : event correlator, time-sync, buffer
- policy/ : DSL interpreter, policy evaluator, simulation
- comms/ : secure networking, MQTT/gRPC/WS wrapper
- data/ : storage client, schema, uploader
- dms-client/ : device agent for fleet management
- ui/ : dashboard web backend + realtime WS
- api/ : REST + GraphQL gateway
- security/ : TPM util, secure boot checks, audit
- ops/ : scripts for deployment, k8s manifests, helm charts

Each module exposes crisp interfaces (gRPC + REST where applicable). See `api/openapi.yaml` for concrete endpoints.

4. Hardware Abstraction Layer (HAL)

Goals

- Auto-discover sensors
- Provide uniform interface for sampling, configuration, calibration
- Allow hot-swap and fallback

HAL interface (pseudo-Python)

```
class SensorBase:
    def probe(self) -> bool: ...
    def read(self) -> dict: ...
    def configure(self, cfg: dict) -> None: ...
    def calibrate(self) -> CalibrationResult: ...
    def health(self) -> HealthStatus: ...

# Example concrete sensor
class CameraSensor(SensorBase):
    def read(self):
        # returns frame + meta
        return {"frame": bytes, "timestamp": 1234567890}
```

Auto-detection

- Probe common buses (I2C, SPI, USB, CAN) with bus-specific adapters
- Keep plugin catalog by sensor UUIDs and vendor IDs

Calibration

- Calibration routines stored as scripts: `calibrate_camera.py`, `calibrate_lidar.py`
- Periodic calibration events scheduled with jitter

Diagnostics

- Expose /hal/health endpoint with JSON summary of uptime, drift, last_calibration

5. AI / ML Pipeline

Architecture

- Per-device multi-threaded orchestrator using multiprocessing + async IO
- Pipelines are chains of operators: source -> preproc -> model -> postproc -> event
- Models packaged as ONNX/TFLite/TorchScript; runtime picks the best backend

Runtime selection

- Backend chooser: if GPU present -> TensorRT/CUDA; if EdgeTPU -> Coral TFLite; else CPU ONNXRuntime

Example pipeline (LPR + classification)

```
Video capture -> ROI detector -> LPR model -> plate parser -> policy check ->
event
              \-> vehicle classifier -> speed estimator -> fused event
```

Inference logging

Each inference emits: {model_id, input_hash, timestamp, duration_ms, confidence, bbox}

Sample edge runner (Python - simplified)

```python
from concurrent.futures import ThreadPoolExecutor

class Pipeline:
    def __init__(self, stages):
        self.stages = stages
        self.exec = ThreadPoolExecutor(max_workers=4)

    def process(self, frame):
        # schedule stages concurrently where possible
        future = self.exec.submit(self._run, frame)
        return future

    def _run(self, frame):
        data = frame
        for s in self.stages:
            data = s.run(data)
        return data
```

~~~
## Model update flow
- Pull signed model bundles from model registry
- Verify signature, validate hash, stage to `/var/models` and switch via atomic
symlink
- Canary on subset of devices (1-5%) then roll out

# 6. Policy & Governance Engine

## Design
- Policies expressed in a compact DSL (YAML + expressions). The engine evaluates
event objects against applicable policy profiles (region/device level
precedence).
- Policy components: data_retention, redaction_rules, enforcement_mode,
alert_rules

## Sample DSL (YAML)

```yaml
policy_id: city_x_default_v1
jurisdiction: City X
data_retention:
  event_metadata_days: 365
  raw_video_days: 30
redaction:
  blur_faces: true
  blur_plates: false
  blur_threshold: 0.8
enforcement:
  red_light_violation:
    mode: enforcement
    evidence_required: [lpr, speed]
    severity_threshold: 0.9
  protest_mode:
    mode: monitoring
    roles_allowed: [city_admin]
~~~

Policy evaluation pipeline

1. Event created -> policy engine resolves jurisdiction chain (device -> site -> city -> state -> country)
2. Evaluate retention, obfuscation rules
3. Return `policy_outcome` attached to event

Simulation mode

- Run a policy through a set of recorded events to estimate outcomes and number of enforcement actions.

7. Data Fusion & Event Model

Event record (canonical JSON)

```
{
  "uuid": "...",
  "timestamp": "2025-09-13T22:00:00Z",
  "location": {"lat": 37.77, "lon": -122.41},
  "event_type": "red_light_violation",
  "severity": 0.92,
  "sensors": ["camera.front", "radar.left"],
  "ai": {
    "lpr": {"plate":"ABC123","confidence":0.96},
    "vehicle_class": {"type":"truck","confidence":0.88}
  },
  "policy_evaluation": {"allowed": false, "action":"issue_ticket"},
  "links": {"video":"s3://.../clip.mp4"}
}
```

JSON Schema (summary)

- `uuid`: string, `timestamp`: ISO8601, `location`: object, `event_type`: enum, `sensors`: array

(Full JSON Schema included in appendix of the repo)

Time sync

- Use GPS PPS where available and NTP/PTP as fallback
- All events stamped with monotonic clock + absolute timestamp

Cross-camera correlation

- Use hashed vehicle fingerprint (appearance hash + plate + trajectory) with TTL to correlate across cameras

8. APIs, SDK, and OpenAPI sample

Authentication

- OAuth2.0 JWT for user APIs
- mTLS for device-to-cloud

Example REST endpoints

- `POST /v1/events/query` - filter events
- `GET /v1/cameras/{id}/live` - wss URL for live stream (tokenized)
- `POST /v1/devices/{id}/commands` - send DMS command

OpenAPI sample (summary)

Include `openapi.yaml` with schema, examples, securitySchemes (bearerAuth + mutualTLS)

9. Security & Privacy

Zero Trust principles

- Authenticate + authorize every request
- Short-lived credentials and telemetry tokens
- Device attestation via TPM

Cryptography

- TLS 1.3 with mutual auth for device-cloud
- AES-256-GCM for local persistent storage
- Signed images and models (Ed25519)

Secure boot & firmware

- U-Boot + signed kernel + verified initramfs
- OTA images signed and validated by device

Tamper detection

- Sensors for chassis open and accelerometer
- Remote lockdown if tamper detected

Privacy-by-design

- Configurable blur/redaction

- Local-only mode: keep all raw video on-device with no upload
- Differential privacy for aggregated exports

10. Fleet management & OTA (DMS)

Features

- Device registry, lifecycle states
- OTA pipeline: firmware -> os packages -> model bundles -> policy bundles
- Blue/green updates with health probes

Health telemetry

- Heartbeat: every 30s publish device health (cpu, mem, temp, sensors)

11. Deployment strategy

Small (1-100 devices)

- Single regional cloud instance (VM), MQTT broker (managed or self-hosted), S3-compatible storage
- Use docker-compose for orchestration

Medium (100-10k)

- Kubernetes cluster per region
- Kafka or managed pubsub for events
- Multi-AZ storage, model registry

Large (10k+)

- Multi-region clusters, autoscaling, tenant isolation
- Policy sharding and hierarchical DMS

Air-gapped

- Local registry (Harbor), signed bundles on USB, manual approval workflows

12. Data governance

Retention matrix (example): - Event metadata: 365 days - Aggregated traffic metrics: 7 years - Raw video (standard): 30 days - Raw video (incident): 2 years (region dependent)

Export controls, audit logs, access review cadence, privacy impact assessment templates included

13. Test cases & simulation scenarios

Scenarios included: - Red-light violation: camera + LIDAR + LPR -> create evidence bundle - Crash detection: high-decel audio + video -> emergency alert - Protest: crowd density spike + protest mode (policy changes) -> monitoring-only - Model rollouts: canary failure -> rollback

Automated tests: - Unit tests for HAL adapters - Integration tests with simulated sensor streams (recorded sample data) - End-to-end tests in K8s using synthetic traffic generator

14. Security audit plan & risk register

Auditable items: - Build pipeline reproducibility - Secrets management - Model poisoning risks

Risk register highlights: - Risk: Unauthorized access to raw video. Mitigation: encryption at rest + RBAC + retention limits. - Risk: Model bias causing misclassification. Mitigation: dataset diversity, monitored metrics, human review.

15. Open source vs proprietary breakdown

Open source candidates: - HAL adapters, connectors, SDKs, example models (non-sensitive), dashboards Proprietary candidates: - Advanced analytics modules, high-value ML models trained on proprietary data, compliance adapters for specific vendors

16. Modular component manifest

Pluggable modules: - Sensor drivers (camera, lidar, radar, gas) - Model plugins (ONNX bundles) - Policy plugins (custom DSL evaluators) - Export plugins (S3, Azure Blob, on-prem NAS) - Notifier plugins (email, SMS, gov-specific webhooks)

17. Source tree (suggested)

```
smartpole/
├── edge/
│   ├── hal/
│   ├── ai/
│   ├── fusion/
│   ├── policy/
│   ├── comms/
│   └── docker/ (edge runtime images)
├── cloud/
│   ├── api/
│   ├── dms/
│   ├── model-registry/
│   └── analytics/
├── ops/
│   ├── k8s/
│   ├── helm/
│   └── infra-as-code/
├── docs/
├── samples/
│   ├── sample_data/
│   └── sample_models/
└── ci/
```

18. Model manifest (example)

```
models/
  lpr-onx-v1/
    model.onnx
    manifest.yaml
    signature.sig
  vehicle-classifier-v2/
    model.tflite
    manifest.yaml
```

manifest.yaml fields: model_id, version, input_shape, labels, quantized: true/false, backend_recommendation, signed_by

19. Appendix — sample files

Sample Dockerfile (edge)

```
FROM ubuntu:22.04
RUN apt-get update && apt-get install -y python3 python3-pip libssl-dev
WORKDIR /app
COPY edge/ /app
RUN pip install -r requirements.txt
CMD ["python3", "-m", "edge.agent"]
```

Sample docker-compose (dev)

```yaml
version: '3.8'
services:
  mqtt:
    image: eclipse-mosquitto
    ports: [1883:1883]
  edge-sim:
    build: ./edge
    environment:
      - MQTT_BROKER=mqtt:1883
    depends_on: [mqtt]
```

Sample K8s deployment (api)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: smartpole-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: smartpole-api
  template:
    metadata:
      labels:
        app: smartpole-api
    spec:
      containers:
      - name: api
        image: myregistry/smartpole-api:stable
        ports:
        - containerPort: 8080
```

Sample OpenAPI fragment

```yaml
openapi: 3.0.3
info:
  title: SmartPole API
  version: 1.0.0
paths:
  /v1/events/query:
    post:
      security:
        - bearerAuth: []
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/EventQuery'
      responses:
        '200':
          description: OK
components:
  schemas:
    EventQuery:
      type: object
      properties:
        time_from:
          type: string
          format: date-time
        event_types:
          type: array
          items:
            type: string
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

Sample policy files (see section 6 for full)

20. Next steps & priorities

- I included a full blueprint and code skeletons in this document. Pick which of the following you want implemented as working artifacts next (I can generate actual files you can download):
- Full edge runtime repo with HAL adapters and simulated sensors.
- Cloud DMS + model registry Docker images and Helm chart.
- Policy engine with DSL interpreter and simulator.
- End-to-end demo (single camera simulation -> event -> cloud ingestion).

(End of document)