PrivacyScrub V5: Combined SRS + TDD

Author: Greg Burns **Version:** 5.0 (Enterprise Release) **Date:** November 26, 2025
Architecture: Microservices / GPU-Accelerated / IaC

1. Introduction

1.1 Purpose

This document defines the Software Requirements (SRS) and Technical Design (TDD) for **PrivacyScrub V5**. V5 evolves the platform from a notebook-based prototype into a production-grade, **Infrastructure as Code (IaC)** deployed enterprise system.

1.2 Scope

V5 introduces:

- **Dedicated Microservices:** Decoupling ingestion, processing, and orchestration.
- **Smart Anonymization:** Persistent object tracking and identity linkage.
- **Enterprise MLOps:** Automated CI/CD gating based on model performance.
- **Infrastructure as Code:** Full GCP environment provisioning via Terraform.

2. System Architecture (Technical Design)

The V5 architecture abandons the monolithic worker in favor of specialized microservices deployed to Google Cloud Run and GKE, managed via Terraform.

2.1 Microservices Ecosystem

Service	Role	Tech Stack	Scaling Profile
API Gateway	Ingress & Routing	FastAPI / Cloud Run	High concurrency, Low CPU
Orchestrator	Job Lifecycle & I/O	Python / Cloud Run	I/O Bound (Stitching/Splitting)

GPU Worker	Inference & Tracking	PyTorch / Cloud Run (GPU)	Compute Bound (CUDA/TensorRT)
State Store	Job Metadata	Firestore	NoSQL Document Store
Media Store	Video/Image Blobs	Google Cloud Storage	Object Storage with Lifecycle

2.2 Infrastructure as Code (IaC) Strategy

All infrastructure is defined in Terraform modules:

- `modules/networking`: VPCs, Subnets, Firewalls.
- `modules/compute`: Cloud Run services, GKE clusters (if needed).
- `modules/storage`: GCS Buckets (with TTL policies), Firestore DB.
- `modules/security`: IAM Roles, Service Accounts, Secret Manager.

3. Functional Requirements (SRS) & Implementation (TDD)

3.1 Core Functionality: Smart Anonymization

FR-V5-01: Persistent Object Tracking (DeepSORT)

- **Requirement:** Redaction masks MUST persist and move smoothly across frames, maintaining ID consistency even if detection fails momentarily.
- **Implementation:** The **GPU Worker** integrates `DeepSORT` (or `ByteTrack`) alongside YOLOv8.
    - *Logic:* `detections = model(frame)`; `tracks = tracker.update(detections)`; `redact(tracks)`.

FR-V5-02: Identity Linkage (Re-ID)

- **Requirement:** The same individual MUST receive the same anonymization ID (e.g., "Person A") throughout the entire video timeline.

- **Implementation:** Generate lightweight embeddings (Re-ID vectors) for detected faces/bodies. Cluster these embeddings within the scope of a single job to unify identities before rendering masks.

FR-V5-03: Anonymization Quality Score

- **Requirement:** System MUST provide a confidence score (0-100) indicating the success of the redaction.
- **Implementation:** Post-redaction, a lightweight secondary model (e.g., a small ResNet binary classifier) scans the redacted regions to detect "leakage" (e.g., visible eyes or text).

3.2 Enterprise Operations

FR-V5-04: Webhook Notifications

- **Requirement:** Clients can register a `webhook_url`. The system MUST POST a payload to this URL upon job completion or failure.
- **Implementation:** The **Orchestrator** service, upon transitioning a job to `COMPLETED` or `FAILED`, reads the `webhook_url` from Firestore and dispatches an async HTTP POST event with the job payload.

FR-V5-05: Immediate Data Erasure (Right to Erasure)

- **Requirement:** `DELETE /v1/jobs/{id}` MUST trigger immediate hard deletion of all associated GCS objects.
- **Implementation:** The API Gateway triggers a background task in the **Orchestrator**. The Orchestrator iterates through `input/`, `chunks/`, and `output/` prefixes for that Job ID and issues `bucket.delete_blob()` calls.

4. MLOps & CI/CD Requirements

4.1 Automated Model Gating (CI/CD)

FR-V5-06: Deployment Gating

- **Requirement:** No model update can be deployed to production without passing a benchmark evaluation ($\text{F1} > 0.85$).
- **Implementation:**
    1. **GitHub Actions / Cloud Build:** Triggered on PR to `main`.
    2. **Eval Step:** Spins up a transient worker, runs the pipeline against a labeled `Golden Dataset` (stored in GCS).

    3. **Gate:** Helper script compares generated metrics against `thresholds.json`. If `current_f1 < target_f1`, the build fails.

4.2 Active Learning Loop

FR-V5-07: Feedback Ingestion

- **Requirement:** QA reviewers can flag "failed frames." These frames are automatically ingested for retraining.
- **Implementation:**
    - New Endpoint: `POST /v1/feedback`. Accepts `job_id`, `frame_index`, and `correction_data`.
    - Action: The Orchestrator copies the specific raw frame from the source video to a `training-backlog` bucket for data scientists to review.

5. API Specification (V5 Update)

5.1 Endpoints

- `POST /v1/video`: Accepts video + `webhook_url`. Returns `job_id`.
- `GET /v1/jobs/{id}`: Returns status, progress, quality score.
- `DELETE /v1/jobs/{id}`: Triggers Hard Delete (FR-V5-05).
- `POST /v1/feedback`: Submits QA data for Active Learning (FR-V5-07).

6. Deployment Guide (Local Workflow)

1. **Prerequisites:**
    - `gcloud` CLI installed and authenticated.
    - `terraform` installed.
    - `docker` installed.

Build & Push:
```
# Build GPU Worker
docker build -t gcr.io/$PROJECT/gpu-worker:v5 -f services/gpu-worker/Dockerfile .
docker push gcr.io/$PROJECT/gpu-worker:v5
```

2.

Provision Infrastructure:
```
cd terraform
terraform init
terraform apply -var="project_id=$PROJECT"
```