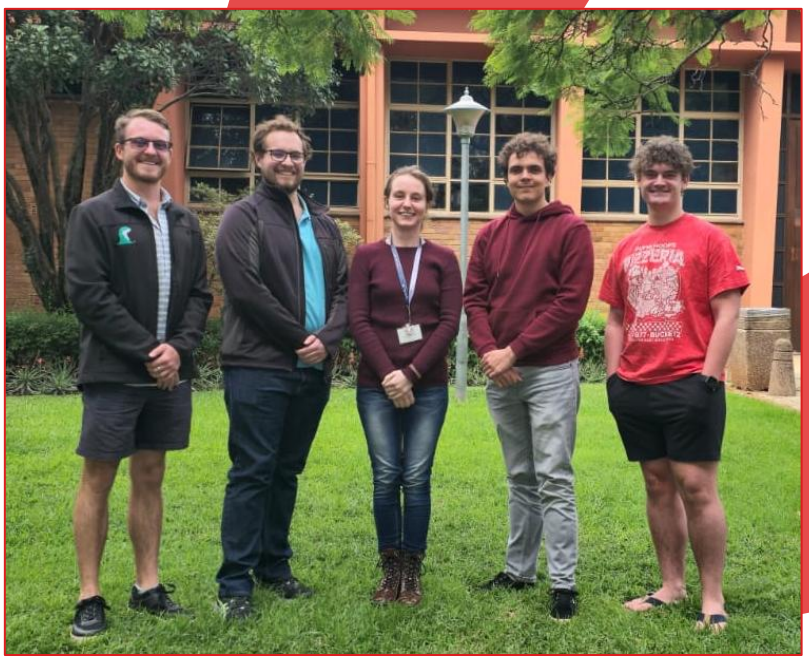
SuperLap Racing Line Optimization System

EPI-USE



Quintessential

Amber Ann Werner [u21457752]

Milan Kruger [u04948123]

Qwinton Knocklein [u21669849]

Sean van der Merwe [u22583387]

Simon van der Merwe [u04576617]

Architectural Requirements

High-Level Architectural Style

Requirement:

- **AR1.1:** The system shall follow a **microservices architecture** for modularity, with separate services for:
    - Image processing (OpenCV/Python)
    - Reinforcement Learning (RL) training (PyTorch/TensorFlow)
    - Visualization (Web-based frontend)
    - User management (Auth0/Custom JWT)
- **AR1.2: Event-driven communication** (e.g., Kafka/RabbitMQ) shall connect services to handle async tasks (e.g., RL training completion triggers visualization updates).

Justification:

- Decouples resource-intensive tasks (e.g., RL training) from user-facing components.
- Enables independent scaling of services.

Core Components & Interactions

Requirement:

- **AR2.1:** The system shall consist of:
    - **Track Processing Service:**
        - Input: Top-down track image (JPEG/PNG).
        - Output: Binary map + detected boundaries (stored in Redis for fast retrieval).
    - **RL Training Service:**

        - Input: Binary map + physics parameters (e.g., tire grip, bike specs).
        - Output: Optimized racing line (stored in PostgreSQL with versioning).
    - **Simulation Engine:**
        - Physics model (e.g., PyBullet/Custom) for realistic dynamics.
    - **API Gateway:**
        - REST/GraphQL endpoints for frontend communication.
    - **Frontend:**
        - Web-based (React/Three.js for 3D) + optional desktop (Electron).

AR2.2: Data flow shall adhere to:

User Upload → Track Processing → RL Training → Simulation → Visualization.

Data Management

Requirement:

- **AR3.1:** Track images and metadata shall be stored in **AWS S3/Blob Storage** (cost-effective for large files).
- **AR3.2:** Simulation results (racing lines, lap times) shall use **PostgreSQL** (structured queries) + **Redis** (caching).
- **AR3.3:** Training data from games/simulators shall be ingested via **parquet files** (columnar storage for efficiency).

Integration Requirements

Requirement:

- **AR4.1:** The system shall support APIs for:
    - **Racing Games** (e.g., Assetto Corsa via UDP/Telemetry APIs).
    - **Cloud GPU Providers** (e.g., AWS SageMaker for distributed RL training).

- **AR4.2:** Third-party auth (Google/OAuth) shall integrate via **Auth0** or **Firebase**.

Scalability & Performance

Requirement:

- **AR5.1:** RL training shall scale horizontally using **Kubernetes** (auto-scaling GPU nodes).
- **AR5.2:** Image processing shall offload to **AWS Lambda** during peak loads.
- **AR5.3:** Frontend shall use **CDN caching** (e.g., Cloudflare) for static assets.

Fault Tolerance & Recovery

Requirement:

- **AR6.1:** Training jobs shall checkpoint progress **every 15 minutes** (prevent data loss).
- **AR6.2:** Database failover shall be automated (PostgreSQL replica in standby mode).
- **AR6.3:** User uploads shall retry **3 times** before error reporting.

Security Architecture

Requirement:

- **AR7.1:** Zero-trust model:
    - **JWT tokens** for API auth.
    - **VPC isolation** for training workloads.
- **AR7.2:** Data encryption:
    - **At rest** (AES-256 for S3/PostgreSQL).
    - **In transit** (HTTPS/mTLS for microservices).

Deployment & DevOps

Requirement:

- **AR8.1:** Infrastructure-as-Code (IaC) via **Terraform/Ansible**.
- **AR8.2:** CI/CD pipeline (GitHub Actions/Jenkins) with:
    - **Testing:** Unit tests (PyTest), integration tests (Selenium).
    - **Rollback:** Automated if error rate >5% in canary deployments.

Cross-Cutting Concerns

Requirement:

- **AR9.1:** Observability:
    - **Logging:** ELK Stack (Elasticsearch, Logstash, Kibana).
    - **Monitoring:** Prometheus/Grafana for GPU usage, API latency.
- **AR9.2:** Compliance with **GDPR** for user data deletion requests.