Product Requirement Document (PRD): OpenCUDA-Lab

1. Overview & Objective

OpenCUDA-Lab is a lightweight, application-level C/CUDA runtime shared library (`libopencuda.so`) designed to eliminate unpredicted Out-Of-Memory (OOM) crashes, mitigate severe VRAM fragmentation, and expose fine-grained telemetry in resource-constrained CUDA environments.

By substituting default CUDA memory management with predictive estimation and custom pooled allocations, OpenCUDA-Lab provides deterministic memory execution and actionable batch-size recommendations for academic and research workloads.

2. Target Persona & Use Cases

- **HPC & AI Researchers:** Running custom CUDA kernels or deep learning primitives on shared or limited GPU infrastructure who need to maximize input batch sizes without manual trial-and-error OOM crashes.
- **Systems & DevOps Engineers:** Requiring application-level visibility into GPU memory dynamics (predictive vs. actual usage, allocation latencies, fragmentation) beyond driver-level metrics provided by tools like `nvidia-smi`.

3. Functional Requirements

FR-1: Memory Runtime Wrappers

- **API Interception:** Expose `ocl_malloc`, `ocl_realloc`, and `ocl_free` as drop-in standard alternatives to `cudaMalloc`, `cudaRealloc`, and `cudaFree`.
- **Zero-Driver-Overhead Execution:** Route standard memory requests to internal memory pools to bypass repeated driver-level allocations.

FR-2: Predictive Memory Estimator

- **Pre-Execution Analytical Pass:** Calculate projected peak VRAM requirements before kernel launch using kernel input shapes, data types, grid/block dimensions, and auxiliary buffer requirements.
- **Preemptive OOM Safeguard:** Evaluate projected memory against available VRAM prior to allocation. Intercept execution and trigger an `OCL_ERR_OOM_PREDICTED` error code if execution risks driver crash.

- **Batch Size Advisor:** Compute maximum safe batch size ($B_{recommended}$) using current pool availability and per-sample memory scaling factors.
-

FR-3: Fragmentation-Aware Pool Allocator

- **Contiguous Pool Pre-Allocation:** Reserve a configurable contiguous VRAM pool upon initialization (ocl_init) to minimize kernel execution context switching.
-
- **Reuse & Binning Policies:** Implement a segregated-fit slab/buddy allocation strategy to reuse memory blocks across varying kernel shapes.
-
- **Dynamic Pool Resizing (ocl_realloc):** Expand or rebalance internal memory pools dynamically using contiguous memory splits without triggering host-device synchronization bottlenecks.
-

FR-4: DevOps Observability & Telemetry

- **Structured Data Export:** Export real-time metrics in structured JSON and CSV formats.
-
- **Metrics Tracked:**
-
    - Predicted Peak VRAM vs. Actual Peak VRAM
    - External Memory Fragmentation Ratio (F)
    - Reallocation and pool-expansion count
    - Allocation/Deallocation latencies (microseconds)
    - Recommended vs. Actual Batch Sizes

4. System Interface & API Design

C

```
// Core Runtime Lifecycle & Allocation API
                ocl_init size_t                    uint32_t
                ocl_malloc void              size_t
                ocl_realloc void             size_t
                ocl_free void
                ocl_shutdown void

// Predictive Memory Estimation & Advice API
                        ocl_predict_peak_memory
    const
    size_t
```

```
size_t ocl_recommend_batch_size
    const
    size_t


// Telemetry Export API
                ocl_export_telemetry_json const char
                ocl_export_telemetry_csv const char
```

5. Non-Functional Requirements

- **Runtime Overhead:** Total memory management latency introduced by libopencuda.so must not exceed **5% to 10%** of total kernel execution time.
-
- **Portability:** Compatible with CUDA Toolkit 11.x and 12.x on Linux x86_64 and AArch64 architectures across NVIDIA Pascal through Hopper microarchitectures.
-
- **Thread Safety:** All pool allocation and telemetry operations must be thread-safe across multi-threaded host host-side callers launching asynchronous CUDA streams.
-

6. Workload Benchmark Suite

System performance and predictability will be evaluated using a two-tier workload suite:

1. **HPC System Kernels:** Vector addition/reduction, Dense Matrix Multiplication (GEMM), and parallel sorting algorithms (Radix Sort).
2.
3. **Dynamic AI Layers:** Convolutional layers ($2D/3D$) and Multi-Head Self-Attention (MHA) layers evaluated under varying sequence lengths and batch sizes.

7. Metrics, Hypotheses & Acceptance Criteria

Metric	Target / Baseline	Hypothesis & Acceptance Standard
Prediction Accuracy	Mean Absolute Percentage Error ($MAPE$)	**H1:** $\text{MAPE} < 15\%$ between analytical prediction and measured peak VRAM.

Effective Capacity	Max runnable batch size before OOM	**H2:** Pooled allocations enable larger maximum batch sizes vs. standard cudaMalloc.
Runtime Overhead	Execution Latency Difference (ΔT)	**H3:** Memory runtime management overhead remains $< 5\text{--}10\%$ of total execution time.
Operational Reliability	Job Failure Rate	**H4:** Elimination of unhandled OOM driver crashes via preemptive validation.
Fragmentation Index	$F = 1 - \frac{LargestFreeBlock}{TotalFreeMemory}$	F remains below **0.20** throughout dynamic reallocations.