RustKernels

GPU-Accelerated Kernel Library

Financial Services, Analytics & Compliance Workloads

Rust • RingKernel Runtime • Batch & Ring Execution • REST/gRPC API

Executive Overview

Version 0.3.1

106 GPU-accelerated kernels across 14 domains with enterprise security, observability, resilience, and service APIs — built on the RustCompute (RingKernel) framework.

Built with Rust • 19 modular crates • Sub-microsecond Ring latency

Contact: michael.ivertowski@ch.ey.com

Contents

1 Executive Summary

RustKernels is a GPU-accelerated kernel library purpose-built for financial services, analytics, and compliance workloads. It is a Rust port of the DotCompute GPU kernel library, built on the RustCompute (RingKernel) framework, delivering 106 production-ready kernels across 14 business domains with both Batch and Ring execution modes.

Why RustKernels?

- **Dual execution modes** — Batch kernels for heavy periodic computation (10–50 μs launch) and Ring kernels as GPU-persistent actors with 100–500 ns message latency.
- **14 business domains** — comprehensive coverage from graph analytics and ML to compliance, risk, banking, treasury, and audit.
- **Enterprise-ready** — built-in security (JWT, RBAC, multi-tenancy), observability (Prometheus, OTLP tracing), and resilience (circuit breakers, retry, timeouts).
- **Service APIs** — deploy as a standalone service via Axum REST, Tonic gRPC, Tower middleware, or Actix actors.
- **Memory safety** — Rust's ownership model eliminates data races in GPU-host coordination without sacrificing performance.
- **Iterative & checkpointable** — multi-pass algorithms (PageRank, K-Means) with convergence tracking and graceful degradation under load.

1.1 At a Glance

106 GPU-accelerated kernels	**14** business domains	**19** modular Rust crates
<500 ns Ring message latency	**2** execution modes	**4** service integrations

2 Execution Modes

RustKernels provides two complementary execution modes, each optimized for different latency and throughput requirements.

2.1 Batch Kernels

Property	Details
Launch Overhead	10–50 μs per invocation
State Location	CPU memory, launched on-demand
Orchestration	CPU-orchestrated dispatch to GPU
Best For	Heavy periodic computation, analytics pipelines, report generation
Trait	`BatchKernel<I, O>` with `execute()` and `execute_with_context()`

2.2 Ring Kernels

Property	Details
Message Latency	100–500 ns per message
State Location	Permanently in GPU memory
Orchestration	GPU-persistent actors, always running
Best For	High-frequency operations, real-time streaming, order matching
Trait	`RingKernelHandler<M, R>` with `handle()` and `handle_secure()`

2.3 Specialized Traits

Beyond the two primary execution modes, kernels can implement additional traits for advanced behavior:

- **IterativeKernel** — multi-pass algorithms (PageRank, K-Means) with convergence tracking and configurable thresholds.
- **CheckpointableKernel** — serialize and restore kernel state for recovery after failures or planned restarts.
- **DegradableKernel** — graceful degradation under load with configurable degradation levels.

2.4 K2K Messaging

Kernel-to-Kernel (K2K) coordination enables complex multi-kernel workflows via four patterns: **IterativeState** (convergence tracking), **ScatterGather** (parallel workers), **FanOut** (broadcast), and **Pipeline** (multi-stage processing).

3 Domain Kernels

RustKernels organizes 106 kernels across 14 business domains, each in its own crate with dedicated message types, ring messages, and comprehensive tests.

3.1 Domain Overview

Domain	Kernels	Key Capabilities
Graph Analytics	28	PageRank, community detection, GNN inference, graph attention, centrality, pathfinding, connected components.
Statistical ML	17	Embedding generation, semantic similarity, SHAP values, feature importance, isolation forest, adaptive thresholds.
Compliance	11	Regulatory scanning, sanctions screening, transaction monitoring, audit trail validation.
Temporal Analysis	7	Time-series decomposition, trend detection, seasonality analysis, change-point detection.
Process Intelligence	7	Digital twin simulation, next-activity prediction, event log imputation, process conformance.
Behavioral Analytics	6	User behavior profiling, session analysis, anomaly detection, interaction patterns.
Risk Analytics	5	VaR calculation, stress testing, risk aggregation, exposure analysis.
Clearing	5	Netting, settlement, margin calculation, position reconciliation.
Treasury	5	Cash flow forecasting, liquidity management, FX exposure, investment analytics.
Accounting	9	Journal entry validation, trial balance, reconciliation, period close, consolidation.
Payments	2	Payment processing, fraud scoring.
Audit	2	Audit sampling, evidence evaluation.
Banking	1	Core banking transaction processing.
Order Matching	1	High-frequency order book matching engine.

3.2 Domain Crate Structure

Each domain crate follows a consistent, self-contained structure:

```
rustkernel-{domain}/
src/
lib.rs                          Module exports, register_all()
messages.rs                     Batch kernel input/output types
ring_messages.rs                Ring message types with #[derive(RingMessage)]
types.rs                        Common domain types
{feature}.rs                    Kernel implementations
```

Ring Message Type ID Ranges: Each domain has a reserved range to prevent collisions — **Graph** 200–299, **Compliance** 300–399, **Temporal** 400–499, **Risk** 600–699, **ML** 700–799.

4 Featured Kernels

4.1 Graph Analytics

Kernel	Description
PageRank	Iterative link analysis with configurable damping factor and convergence threshold. Implements `IterativeKernel`.
GNN Inference	Message-passing neural network inference for node classification and link prediction on transaction graphs.
Graph Attention	Graph Attention Network with multi-head attention for learning node representations with weighted neighbor aggregation.
Community Detection	Louvain and label propagation algorithms for discovering clusters in entity relationship graphs.

4.2 Machine Learning

Kernel	Description
SHAP Values	Kernel SHAP implementation for model-agnostic feature attribution and explainability on GPU.
Secure Aggregation	Federated learning aggregation with differential privacy for privacy-preserving model training.
Streaming Isolation Forest	Online anomaly detection with adaptive tree updates for real-time transaction monitoring.
Adaptive Threshold	Self-adjusting thresholds with concept drift detection for evolving data distributions.
Drug Interaction	Multi-drug interaction prediction using molecular fingerprints and interaction scoring.

4.3 Process Intelligence

Kernel	Description
Digital Twin	Monte Carlo process simulation for capacity planning, bottleneck identification, and what-if analysis.
Next Activity Prediction	Markov chain and N-gram models for predicting the next activity in a business process execution.
Event Log Imputation	Automated detection and repair of missing, out-of-order, and duplicate events in process logs.

5 Enterprise Modules

RustKernels v0.3.1 includes five enterprise modules providing production-grade infrastructure for deployment at scale.

5.1 Security

Feature	Details
Authentication	JWT and API key validation via `AuthConfig` with pluggable providers.
RBAC	Role-based access control with four permission levels: Execute, Configure, Monitor, Admin.
Multi-Tenancy	Tenant isolation with `TenantId`, resource quotas, and per-tenant kernel registries.
Secrets	`SecretStore` abstraction for credential management with environment and vault backends.

5.2 Observability

Feature	Details
Metrics	Prometheus-compatible metrics via `KernelMetrics` with histograms, counters, and gauges per kernel.
Tracing	Distributed tracing with OTLP export via `KernelTracing`, span propagation across K2K calls.
Logging	Structured logging with kernel context, tenant ID, and correlation IDs.
Alerting	SLO-based alerts with configurable `AlertRule` definitions and notification channels.

5.3 Resilience

Feature	Details
Circuit Breaker	Failure isolation with configurable thresholds, half-open probing, and automatic recovery.
Retry	Exponential backoff with jitter, configurable max attempts, and per-kernel retry policies.
Timeouts	Deadline propagation with `DeadlineContext` ensuring cascading timeout enforcement.
Health Checks	Liveness and readiness probes via `HealthProbe` with per-kernel health status reporting.

5.4 Runtime & Memory

Feature	Details
Lifecycle	State machine with Starting → Running → Draining → Stopped transitions and graceful shutdown.
Configuration	`RuntimeConfig` with presets: development, production, high-performance; file and environment sources.
Memory Pooling	Size-stratified memory pools via `KernelMemoryManager` with configurable pressure thresholds.
GPU Reductions	Multi-phase GPU reductions via `InterPhaseReduction` for efficient large-scale aggregation.
Analytics Contexts	Workload-specific buffer management with `AnalyticsContextManager`.

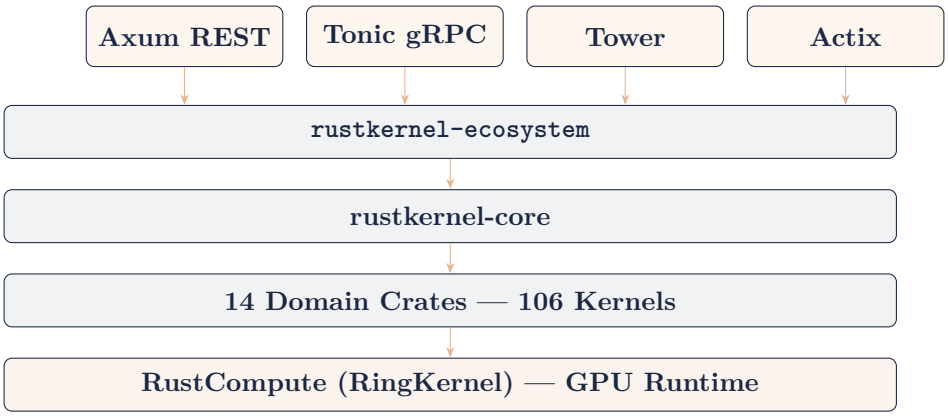
5.5 Licensing

Enterprise licensing with domain-based validation:

- **DevelopmentLicense** — all features enabled for local development and testing.
- **Domain-based gating** — feature validation at kernel registration and activation time via `LicenseValidator`.
- **Per-kernel control** — individual kernel enablement with license tier awareness.

6 Service Integrations

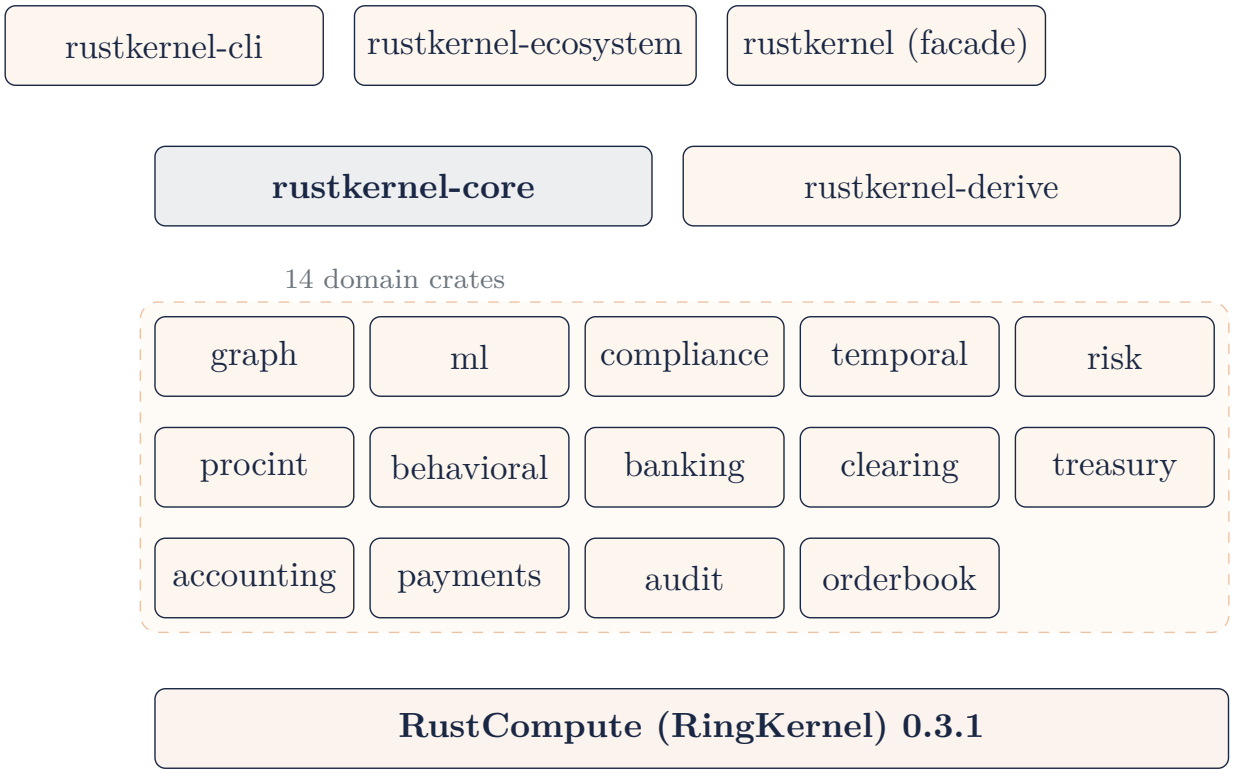
The `rustkernel-ecosystem` crate provides four service integration paths for deploying RustKernels as a standalone service.



Integration	Details
Axum REST	`KernelRouter` with endpoints: `/kernels`, `/execute`, `/health`, `/metrics`.
Tonic gRPC	`KernelGrpcServer` for high-performance RPC with protobuf serialization.
Tower Middleware	`TimeoutLayer`, `RateLimiterLayer`, and `KernelService` for composable request pipelines.
Actix Actors	`KernelActor` for actor-based concurrency and message-driven kernel execution.

7 Architecture

7.1 Workspace Structure



7.2 Core Trait Hierarchy

All kernels implement `GpuKernel` (metadata, validation, health). Execution traits branch into two paths:

BatchKernel<I, O> ⟶ IterativeKernel<S, I, O> ⟶ DegradableKernel
RingKernelHandler<M, R> ⟶ CheckpointableKernel ⟶ DegradableKernel

8 Use Cases

Primary Use Cases

- **Transaction Graph Analytics** — GPU-accelerated PageRank, community detection, and GNN inference on large-scale financial transaction networks.
- **Real-Time Compliance Screening** — Sub-microsecond sanctions screening and transaction monitoring via Ring kernels for continuous compliance.
- **Risk Analytics** — VaR calculation, stress testing, and exposure analysis with GPU-parallel Monte Carlo simulation.
- **ML Model Serving** — GPU-accelerated inference for fraud detection, anomaly scoring, and feature importance with SHAP explanations.
- **High-Frequency Order Matching** — Ring kernel-based order book with persistent GPU state for ultra-low-latency matching.
- **Process Intelligence** — Digital twin simulation, next-activity prediction, and event log quality analysis for operational optimization.
- **Treasury & Cash Management** — Cash flow forecasting, liquidity optimization, and FX exposure management with temporal analytics.
- **Federated Learning** — Privacy-preserving model training with secure aggregation and differential privacy across organizational boundaries.

9 Getting Started

9.1 Build & Test

```
# Build entire workspace
cargo build -workspace

# Run all tests
cargo test -workspace

# Build specific domain crate
cargo build -package rustkernel-graph

# Lint with all features
cargo clippy -all-targets -all-features - -D warnings
```

9.2 Deploy as REST Service

```
use rustkernel_ecosystem::axum::{KernelRouter, RouterConfig};

let router = KernelRouter::new(registry, RouterConfig::default());
let app = router.into_router();
// Serve with axum::Server on port 3000
```

9.3 Production Configuration

```
use rustkernel_core::config::{ProductionConfigBuilder};

let config = ProductionConfigBuilder::production()
.service_name("my-service")
.environment("staging")
.build()?;
```

Ecosystem Integration

RustCompute	GPU-native persistent actor model (RingKernel) providing the runtime foundation for all kernel execution.
RustGraph	GPU-native living graph database leveraging RustKernels for graph analytics and ML algorithms.
DataSynth	Enterprise synthetic data platform generating test data for kernel development and validation.
AssureTwin	Native audit intelligence platform consuming RustKernels for GPU-accelerated audit analytics.

Contact: michael.ivertowski@ch.ey.com