RaveMinds Series 1 Technical Playbook

Practical Applications of Local AI in Capital Markets and Payments

Karthikeyan Ganesan
RaveMinds Community
Capital Markets and Payments · Artificial Intelligence

Series 1 · 2026

www.raveminds.ai · contact@raveminds.ai

Market Sentinel	FinGuard	AskOps

3	$0	2 Months	100%
Projects	API Cost	Timeline	Local AI

Stack: Ollama · Mistral 7B · LanceDB · LangGraph · MCP · Sentence Transformers · DuckDB · FastAPI · Streamlit · Langfuse · n8n · Docker

github.com/raveminds-learn

Table of Contents

01 **Introduction to RaveMinds**
Philosophy, goals, and the local-first approach

02 **Common Architecture**
The shared pipeline diagram across all three systems

03 **AI Concepts Reference**
Embeddings, RAG, agents, LLMs, MCP, and more

04 **Market Sentinel**
Real-time market risk scoring

05 **FinGuard**
Behavioral fraud detection for payments

06 **AskOps**
Agentic infrastructure intelligence

07 **Architecture Patterns**
What ties all three systems together

08 **Local-First Stack Deep Dive**
Why every tool was chosen deliberately

01 Introduction to RaveMinds

RaveMinds is a community learning initiative built around one principle: learn AI by solving real problems in Capital Markets and Payments. Not generic use cases. Not toy demos. Series 1 is the first chapter — three production-grade AI systems, each solving a distinct domain problem, all built on a zero-cost local-first stack.

Philosophy

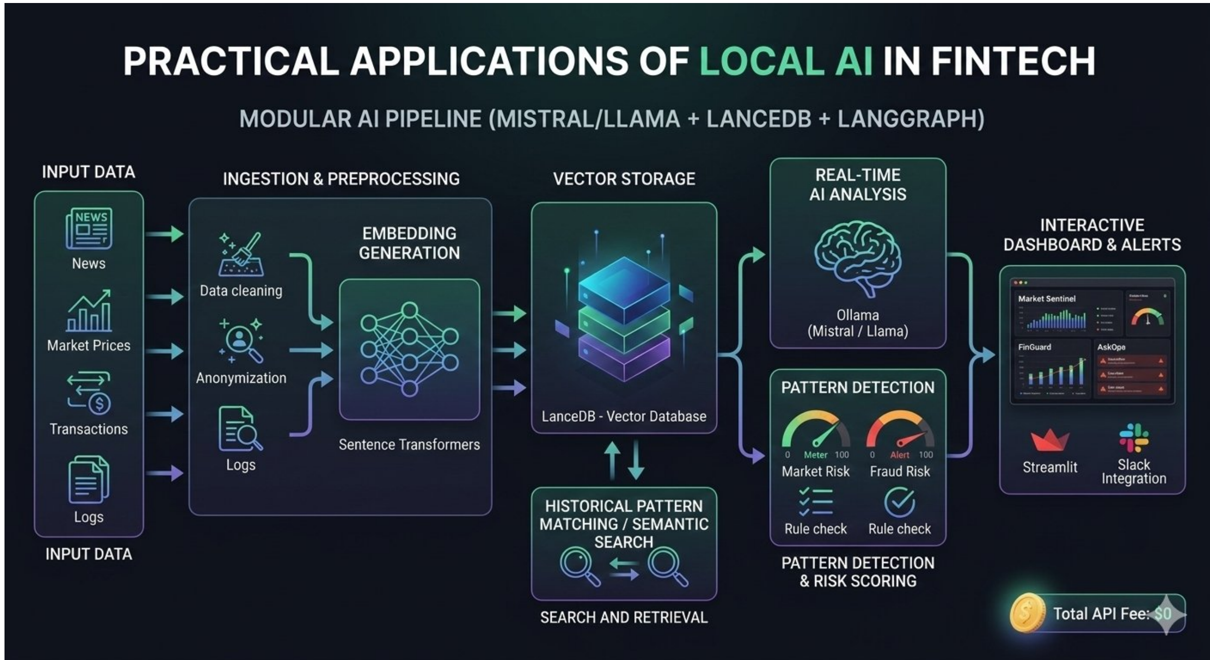
Domain-First	Every use case originates from real Capital Markets and Payments problems.
Local-First	All AI runs via Ollama. No cloud API keys, no usage costs, no data leaving your infrastructure.
Systems Thinking	The model is just one component. The real engineering is in the pipeline around it.
Modularity	Change the LLM, vector store, or embedding model without touching application logic.

Series 1 at a Glance

System	Domain Problem	Core AI Technique	Output
Market Sentinel	Slow manual risk assessment	LLM sentiment + vector RAG + DuckDB	Quant risk score
FinGuard	Fraud patterns rules miss	Behavioral embeddings + hybrid scoring	Risk score + alert
AskOps	Engineers as human query engines	Multi-agent LangGraph + MCP	Plain-English answer

02 Common Architecture

All three Series 1 systems share the same modular pipeline. News, market prices, transactions, and logs flow through identical stages — ingestion, anonymization, embedding, vector storage, AI analysis, and interactive delivery. The diagram below is the architectural blueprint that connects FinGuard, Market Sentinel, and AskOps.



Practical Applications of Local AI in Fintech · Modular AI Pipeline: Mistral/Llama + LanceDB + LangGraph · Total API Fee: $0

Pipeline Stage Breakdown

Stage	Description
Input Data	News headlines, market prices, financial transactions, and infrastructure logs. All data types feed the same ingestion layer.
Ingestion & Preprocessing	Data cleaning and PII anonymization (SHA-256 hashing, merchant categorization, amount bucketing, location generalization). Zero PII enters the vector store.
Embedding Generation	Sentence Transformers (all-MiniLM-L6-v2) convert cleaned data into 384-dimensional vectors capturing semantic meaning, not just surface keywords.
Vector Storage	LanceDB stores all embeddings with zero-copy columnar storage via Apache Arrow. Enables millisecond semantic similarity search across all historical patterns.
Historical Pattern Matching	Semantic search retrieves the most similar historical events, transactions, or incidents to ground the LLM in real precedent rather than training data alone.
Real-Time AI Analysis	Mistral 7B via Ollama reasons over the retrieved context. Returns structured JSON with scores, reasoning, and recommendations. All inference runs locally — $0 API cost.
Pattern Detection & Risk Scoring	Rule-based checks combined with LLM confidence scores produce the final output: fraud risk score (FinGuard), market risk score (Market Sentinel), or ops summary (AskOps).

Interactive Dashboard & Alerts	Streamlit dashboard (FinGuard, Market Sentinel) and Slack integration (AskOps) deliver the output to both technical and non-technical users.

03 AI Concepts Reference

Every AI concept used across Series 1, explained plainly with direct reference to where it appears in the systems.

Vector Embeddings

A way to represent text, transactions, or events as a list of numbers capturing semantic meaning. Two pieces of text with similar meaning will have mathematically close vectors, even if the exact words differ. FinGuard uses 384-dimensional embeddings (all-MiniLM-L6-v2).

Applied in RaveMinds: *FinGuard vectorizes transaction behavior. Market Sentinel embeds news events. AskOps embeds historical incidents. All stored in LanceDB.*

Semantic Similarity Search

Finding items that are conceptually similar by comparing vector representations using cosine distance — not keyword matching. Two transactions with different amounts, merchants, and dates can match if their behavioral fingerprints are similar.

Applied in RaveMinds: *FinGuard finds transactions matching known fraud patterns. Market Sentinel retrieves historically similar market events. AskOps surfaces past incidents.*

RAG — Retrieval Augmented Generation

Retrieved context is provided to the LLM alongside the query. This grounds the LLM in real data rather than training knowledge alone, preventing hallucination on factual details.

Applied in RaveMinds: *All three systems use RAG. The LLM receives retrieved historical context before generating analysis.*

LLM — Large Language Model

Neural network trained on large text corpora that understands and generates language. Used here for reasoning, forensic analysis, sentiment scoring, intent classification, and structured JSON output generation.

Applied in RaveMinds: *Mistral 7B via Ollama runs locally across all three systems. Zero API calls. Zero cost.*

Multi-Agent Orchestration

Multiple specialized AI agents each handle a distinct responsibility and pass state between each other in a defined graph workflow. Each agent is independently debuggable and testable.

Applied in RaveMinds: *AskOps: Supervisor classifies intent → Specialist queries MCP tools in parallel → Formatter renders Slack response.*

MCP — Model Context Protocol

Standardized protocol for connecting AI agents to external tools and data sources. Each tool implements the same interface so the agent communicates uniformly regardless of whether it is talking to Datadog, Oracle, Kafka, or EKS.

Applied in RaveMinds: *AskOps uses MCP servers for all four data sources. Adding a new source only requires implementing the MCP interface — no agent changes.*

Hybrid Scoring

Combining deterministic rule-based signals with probabilistic AI confidence scores. Rules provide stability and auditability. AI handles edge cases that rules have never encoded.

Applied in RaveMinds: *FinGuard: 40% rule-based + 60% LLM confidence. Pure AI hallucinates on edge cases. Pure rules miss novel patterns.*

PII Anonymization

Removing personally identifiable information before any AI component touches the data. SHA-256 hashed user IDs, categorized merchants, bucketed amounts, generalized locations — applied at ingestion before vectorization.

Applied in RaveMinds: *FinGuard ingestion.py applies full anonymization. Vector embeddings contain zero PII.*

LLM Observability

Tracing and logging every LLM call, agent decision, and tool invocation in a structured, queryable way. Non-negotiable for debugging multi-agent systems.

Applied in RaveMinds: *AskOps uses Langfuse (port 3000) to trace every agent transition and MCP tool call.*

Local LLM Deployment

Running LLM inference locally via Ollama rather than calling a cloud API. No API costs, no data privacy concerns, no rate limits, consistent latency.

Applied in RaveMinds: *All three systems: Mistral 7B via Ollama. Total API fee for the entire series: $0.*

04 Market Sentinel

AI-Powered Market Risk Assessment

Ollama + Mistral	Sentence Transformers	LanceDB	DuckDB	Streamlit	Plotly	Docker

The Problem

Financial markets react instantly to news, but assessing risk impact remains slow and subjective. Traditional analysis requires hours of manual research — checking historical precedents, analyzing market patterns, calculating potential impact. By the time analysis is complete, the market has already moved.

The Solution

Market Sentinel transforms a breaking news headline into a quantitative risk score (0–100) with explainable reasoning in real time. It combines Mistral LLM for contextual understanding, LanceDB for historical precedent retrieval, and DuckDB for actual price reaction analytics.

Analysis Pipeline

understanding/	Mistral LLM analyzes event type, sentiment, and potential impact. Returns structured JSON.
rag/	Sentence Transformers embed the event. LanceDB retrieves top similar historical events by cosine similarity.
analytics/	DuckDB queries historical OHLCV data for price return analysis, volatility, and market impact metrics.
scoring/	Multi-factor engine combines LLM sentiment + historical similarity + price reaction into 0–100 score.
ui/	Streamlit dashboard with risk gauge, Plotly charts, and actionable portfolio recommendations.

Example: Input → Output

```
INPUT title: "Apple Announces Major Product Recall Due to Safety Concerns"
ticker: AAPL | date: 2024-01-17
OUTPUT risk_score: 78 | risk_level: High
scoring: Sentiment 15pts · Impact 12pts · Similar events 8pts · Price reaction 18pts
actions: Reduce AAPL exposure 20–30% · Monitor price action 48–72 hours
```

Why Three Separate Databases?

LanceDB	Vector similarity search. Zero-copy columnar storage via Apache Arrow. Millisecond semantic search across thousands of historical events.

DuckDB	In-process analytical database. SQL aggregations over OHLCV time-series data without an external server. Much faster than pandas for this workload.
Ollama + Mistral	LLM reasoning about novel events with no direct historical match. Handles qualitative context that quantitative data cannot.

Risk Score Factors

Sentiment Analysis	15 points — LLM-assessed event sentiment
Impact Analysis	12 points — Estimated scope and severity
Similar Events	8 points — Historical precedent from vector search
Price Reaction	18 points — Actual price movement from DuckDB
Volatility	17 points — Historical volatility around similar events
Total	0–100 with Low / Medium / High / Critical thresholds

05 FinGuard

Behavioral Fraud Detection with Zero-Cost AI

Ollama + Mistral	Sentence Transformers	LanceDB	Streamlit	FastAPI	PyArrow

The Problem

Traditional fraud detection relies on point-in-time analysis — examining a single transaction and asking "Is this unusual?" Fraudsters bypass this with low-and-slow attacks that traditional rule-based systems cannot catch. Standard systems have memory for rules, but no memory for behavioral nuance across time, accounts, and contexts.

What standard systems miss:

Micro-transactions	Moving small amounts across thousands of accounts.
Time-dilated attacks	Actions performed days or weeks apart to avoid velocity alerts.
Social engineering	Mimicking legitimate user behavior with slight intent deviations.
Distributed patterns	Coordinated activities across multiple accounts simultaneously.

The Solution

FinGuard acts as a persistent behavioral memory layer. Every transaction is converted into a 384-dimensional embedding capturing behavioral context, stored in LanceDB, and queried semantically to find transactions matching known fraud patterns even when surface details are completely different.

Processing Pipeline

ingestion.py	Anonymize PII: SHA-256 hash user IDs, categorize merchants, bucket amounts, generalize locations.
vectorization.py	Sentence Transformers generate 384-dim behavioral embeddings from transaction context.
pattern_detection.py	LanceDB vector similarity search finds top-20 matching fraud pattern embeddings.
investigation.py	Mistral via Ollama performs forensic analysis. Returns structured JSON with reasoning.
risk_scoring.py	40% rule-based signals + 60% LLM confidence. Hybrid gives stability and novelty detection.
database.py	LanceDB management for persistent behavioral memory across all sessions.
dashboard/app.py	Streamlit UI: Overview, Transactions, Analytics, and System tabs.

Fraud Patterns Detected

Card Testing	Multiple small transactions in short windows to validate stolen card numbers.
Account Takeover	Behavioral shift in device, location, timing, and merchant.
Low-and-Slow	Small amounts over weeks — invisible to velocity rules.
Velocity Attacks	Rapid coordinated fraud across multiple accounts.
Merchant Hopping	Switching merchants to avoid single-merchant velocity triggers.
Device Switching	Multiple devices used for the same account in a short timeframe.

Example: Input → Output

```
INPUT transaction_id: TXN_98234 | user_id: USER_5021 | amount: $47.50
merchant: Online Electronics Store | timestamp: 2026-01-19T14:23:45Z

OUTPUT fraud_risk_score: 82 | risk_level: High
reasoning: Similar micro-transactions across 12 accounts over 8 days.
Vector similarity: 0.89 with Card-Testing fraud pattern.
actions: Flag USER_5021 · limit to $25 · extra auth x3 transactions
```

Performance

Embedding generation	~50ms per transaction
Vector search	~10ms for top-20 similar transactions
LLM investigation	~2–5 seconds (high-risk cases only)
End-to-end	~3–6 seconds per transaction
Infrastructure cost	$0

06 AskOps

Ask Your Infrastructure Anything

Mistral + Ollama	LangGraph	MCP Protocol	LanceDB	n8n	Langfuse	Slack Bolt	FastAPI

The Problem

Every day, business teams ping engineers: "What happened to trade 12345?" The engineer drops everything, opens 4 dashboards, queries the database, reads logs, and replies 10 minutes later. Multiply by 20 engineers, 10 times a day. Thousands of hours of lost engineering time every year. Infrastructure knowledge is locked in dashboards only technical teams can access efficiently.

The Solution

AskOps sits in Slack and answers infrastructure questions in plain English in under a minute. A three-agent LangGraph workflow handles the full pipeline. MCP provides standardized connectivity to every data source so agent logic never needs to know the implementation details of Datadog, Kafka, Oracle, or EKS.

Three-Agent Architecture

Supervisor Agent	Classifies user intent (trade query, incident lookup, system health). Extracts IDs and routes to Specialist with structured context.
Specialist Agent	Queries all data sources in parallel via MCP: datadog_mcp.py (alerts + logs), oracle_mcp.py (trade lifecycle), kafka_mcp.py (consumer lag), eks_mcp.py (pod status).
Formatter Agent	Generates three Slack views: Manager View (status, SLA, exposure), Technical Deep-Dive (root cause, fix), Full Report (complete analysis).

Full Workflow

```
Slack → FastAPI (:8001) → Supervisor → Specialist → Formatter → Slack Response

Specialist makes parallel MCP calls:
datadog_mcp.py → alerts + logs
oracle_mcp.py → trade lifecycle
kafka_mcp.py → consumer lag
eks_mcp.py → pod status

LanceDB vector store ← n8n data pipelines (:5678) ← historical incident feed
Langfuse traces (:3000) ← full LLM observability
```

Demo Trade Scenarios

Trade 12345	Settlement timeout + EKS pod OOMKilled + Kafka spike 45,000 msgs. Same pattern seen last Tuesday. State: FAILED.

Trade 22222	Duplicate of trade 22198 within 5-min dedup window. Dedup-service worked as designed. State: REJECTED.
Trade 33333	Full outage — all settlement pods in CrashLoopBackOff. State: PENDING.
Trade 44444	Settled in 8m 34s vs 2-minute SLA. State: SETTLED (slow).
Trade 55555	Notional $8.75M exceeds trader limit $5M. State: REJECTED.

Example Slack Response — Trade 12345

```
Trade 12345 — FAILED
09:47 AM Received and validated ✓
10:30 AM Kafka lag spiked to 45,000 messages
10:33 AM Settlement service timed out
10:34 AM EKS pod restarted twice (OOMKilled)
10:35 AM Trade moved to FAILED state

Root cause: Settlement pod instability + Kafka backlog.
Same pattern seen last Tuesday. Trade has NOT been reprocessed.
```

Running Services

FastAPI Gateway	http://localhost:8001 (API docs at /docs)
n8n Pipelines	http://localhost:5678 (visual workflow — historical context feed into LanceDB)
Langfuse Traces	http://localhost:3000 (complete LLM observability)
Slack Bot	Socket mode — no public endpoint required

07 Architecture Patterns

Three systems. Three different problems. The same architectural philosophy runs through all of them. These patterns were not planned upfront — they emerged from building Series 1.

Separation of Concerns	Ingestion, transformation, storage, retrieval, reasoning, and presentation are distinct modules. Changing the embedding model never requires touching the risk scoring logic.
Vector Storage as Behavioral Memory	LanceDB serves as a long-term memory layer enabling reasoning across time. FinGuard remembers behavioral patterns. Market Sentinel remembers how similar events played out. AskOps remembers past incidents — surfacing "same pattern seen last Tuesday."
LLM as Reasoning Layer Only	No system asks the LLM to retrieve or store data. The LLM receives pre-fetched, structured context and is asked only to reason about it. This prevents hallucination on factual financial data.
Graceful Degradation	All three systems degrade when the LLM is unavailable. FinGuard falls back to rule-based scoring. Market Sentinel returns quantitative metrics. AskOps returns raw data. Systems requiring perfect AI availability are not production-grade.
Privacy by Architecture	PII anonymization happens at ingestion before any AI component sees the data. Once PII is removed upstream, every downstream component is clean by default — not by policy.
Zero External Dependencies	No cloud APIs, no managed vector databases, no subscription services. Ollama for LLM, LanceDB for vectors, DuckDB for analytics — all local, embedded, free. This constraint forces more resilient system design.

08 Local-First Stack Deep Dive

Every tool was chosen deliberately. Here is the reasoning behind each decision, including what was evaluated and why it was not selected.

Ollama + Mistral 7B	Chosen over OpenAI GPT-4, Anthropic Claude. Local deployment eliminates API costs and data privacy concerns. Mistral's strong performance on financial text made it ideal. Running locally forces architectural discipline — you cannot rely on a powerful hosted model to paper over poor system design.
LanceDB	Chosen over Pinecone, Weaviate, ChromaDB. Serverless embedded database running in-process with zero-copy columnar storage (Apache Arrow). No server to manage, no network latency, no subscription cost. Used as the vector store in all three systems.
LangGraph	Chosen over simple LangChain chains and custom orchestration. AskOps requires branching logic, parallel MCP tool calls, and state management across agent transitions. LangGraph's stateful graph model provides all of this. Each agent is independently testable.
MCP Protocol	Chosen over direct API integrations and custom tool interfaces. Standardized connectivity decouples agent logic from data source implementations. Adding Prometheus or Grafana requires only implementing the MCP interface — zero changes to agent code.
Sentence Transformers (all-MiniLM-L6-v2)	Chosen over OpenAI text-embedding-ada-002 and Cohere. Local embedding generation eliminates per-call API cost and latency. 384-dim vectors are compact, fast, and performant for semantic similarity in financial text.
DuckDB	Chosen over PostgreSQL, ClickHouse, and pandas-only. In-process analytical database without an external server. Much faster than pandas for OHLCV aggregations and time-series analysis while remaining lightweight.
Langfuse	Chosen over custom logging and LangSmith. Every LLM prompt, completion, agent transition, and MCP tool call is captured and queryable. Non-negotiable for debugging AskOps multi-agent workflows.
n8n	Visual pipeline builder used in AskOps to continuously feed historical incident patterns into LanceDB. Creates an ever-growing knowledge base without manual curation — enabling "same pattern seen last Tuesday" responses.
Streamlit	Chosen over React and Vue for FinGuard and Market Sentinel dashboards. Interactive dashboards built entirely in Python — accessible to data scientists without frontend/backend divide.