DeFi Depeg Sentinel — Single-Notebook Game Plan (PDF-ready)

A dual-LLM, dual-pipeline, dual-MCP early-warning system for stablecoin & liquidity risk — implemented in one Jupyter notebook.

---

0) Purpose & Outcome

Goal: Detect early signs of stablecoin depegs and liquidity stress across AMM pools by fusing on-chain microstructure signals with off-chain narrative signals (governance/status posts), then produce an actionable analyst note with concrete mitigations.

One-Notebook Constraint: Everything (config, data pulls, feature engineering, dual LLM agents, MCP tools, evaluation, and export) lives in a single `.ipynb` to maximize reproducibility. External helpers are embedded as notebook cells.

2×2×2 Constraint:

- **2 LLMs (small):**
- *LLM#1 (IE/NLU):* Extracts structured risk events from unstructured text (governance, status, short news). Example: `phi-3-mini` or `gemma-2b` (local).
- *LLM#2 (Reasoner/Writer):* Fuses numeric features + events into a risk score, analyst note, and action plan. Example: `mistral-7b-instruct` (local).
- **2 Pipelines:**
- *Pipeline A (On-chain):* Pool state, oracle prices, reserve/TVL shifts → engineered features → anomaly score.
- *Pipeline B (Off-chain):* Governance/status posts → LLM#1 IE → structured risk events with severity.
- **2 MCP-style Tools:**
- *MCP#1* `onchain-data`: `get_oracle_price()`, `get_pool_price()`, `get_reserves()`, `get_virtual_price()`.
- *MCP#2* `offchain-intel`: `fetch_gov_posts()`, `fetch_status_updates()`, `fetch_market_meta()`.

Primary Outputs:

- Table of engineered features and anomaly scores per pool.
- Structured list of recent risk events with severities.
- **Analyst Note** (human-readable) with: (1) risk score 0–100, (2) 2–3 recommended actions, (3) rationale, (4) audit trail refs.
- Lightweight inline charts + file artifacts: `features.parquet`, `events.json`, `analyst_note.txt` (and optional HTML/PDF export).

---

1) Notebook Table of Contents (Cells)

1. **Preamble & Environment Checks**
2. Install/imports, version printouts, local LLM availability (Ollama), fail-safe mocks.
3. **Secrets & Configuration**
4. RPC URL, pool & oracle addresses, protocol list, model names, output paths.
5. **Utility Functions**
6. JSON helpers, retry wrappers, timestamp parsers, safe casting.
7. **MCP#1: On-Chain Data Tool (in-notebook)**
8. Web3 client, minimal ABIs, price/reserve/virtual_price queries.
9. **MCP#2: Off-Chain Intel Tool (in-notebook)**
10. HTTP fetchers/scrapers for governance & status posts, text normalization.
11. **Pipeline A: Feature Engineering (On-Chain)**
12. Pull prices/reserves → compute deviations, spreads, reserve deltas, rolling stats.
13. **Pipeline A: Anomaly Scoring**
14. IsolationForest (baseline), hooks for alternative detectors.
15. **Pipeline B: Docs to Events (Off-Chain)**
16. Concatenate recent posts → LLM#1 extraction → JSON events (type, severity, ts, summary).
17. **Fusion & Reasoning**
18. Combine numeric features + events → LLM#2 prompt → risk score + analyst note + actions.
19. **Visualization & Outputs**
20. Bar charts of anomaly scores, compact table view, write artifacts.
21. **Evaluation Harness (Backtest Stubs)**
22. Sliding window runner, heuristic labeling, metrics, ablation toggles.
23. **Packaging & Export**
24. Save notebook state, export note, (optional) convert to HTML/PDF.

---

2) Detailed Architecture & Data Flow

2.1 Entities & Data Sources

- **Pools:** Stable-stable AMM pairs (e.g., USDC/USDT Uniswap v3; DAI/USDC Curve).
- **Oracles:** Reference prices (e.g., Chainlink aggregators).
- **Governance/Status:** Protocol forums, status pages, and short updates.

2.2 Core Features (On-Chain)

- **DEX vs Oracle Deviation (**dev**)**: $dev = (p_{dex} - p_{oracle})/p_{oracle}$
- **Spread/Slippage Proxy:** Derived from tick/fee tier or Curve virtual price & imbalance vectors.

- **Reserve Delta:** $\Delta R = R_t - R_{t-1}$ per token + rolling gradient.
- **Liquidity Outflow Rate:** Rolling % change in TVL/proxy reserves.
- **Volatility Proxy:** Rolling std of deviations.

2.3 Event Schema (Off-Chain)

```
{
  "type": "oracle_update | pause_notice | governance_vote | incident |
parameter_change",
  "severity": 1-5,
  "ts": "ISO-8601",
  "summary": "Short human text",
  "source": "url or tag"
}
```

2.4 Fusion & Reasoning

- Input bundle to LLM#2:
- **Numerics:** Current `dev`, `anom_score`, `reserve_delta`, `vol_proxy` per pool.
- **Events:** Top-k recent events with severity and timestamps.
- Output:
- **risk_score:** 0–100 with 1–2 sentence justification referencing signals.
- **analyst_note:** <200 words, concise.
- **actions:** 2–3 concrete steps (hedge/unwind/routing/monitoring) each with a one-line rationale.

---

3) Implementation Blueprint (Cell-Level Spec)

Cell 1 — Preamble & Env Checks

- Imports: `web3`, `pandas`, `numpy`, `sklearn`, `matplotlib`, `requests`, `pydantic`, `tenacity`, `json`, `subprocess`, `textwrap`, `pathlib`, `datetime`.
- Detect Ollama (`which ollama`) and list local models; define `run_ollama(model, prompt)` with graceful fallback to mock outputs.

Cell 2 — Secrets & Configuration

- `Config` via `pydantic.BaseModel`: RPC, aggregator addresses, pool addresses/types, protocol list, time windows (lookback), model names, output directory.
- Sanity checks: non-empty strings, checksum addresses where applicable.

Cell 3 — Utilities

- `retry` decorator (`tenacity`) for flaky RPC/HTTP.
- `safe_json_extract()` to slice LLM outputs between first `[` and last `]`.
- `ts_utcnow()` helper.

Cell 4 — MCP#1: On-Chain Data Tool

- Web3 provider init; assert connectivity.
- Minimal ABIs:
- Chainlink `latestAnswer()`.
- Uniswap v3 `slot0()` and `liquidity()`; optional `observe()` for TWAP.
- Curve `get_dy()` / `virtual_price` (if available) — or fallback to price proxy from balances.
- Methods:
- `get_oracle_price(feed_addr) -> float`
- `get_univ3_price(pool_addr) -> float` (sqrtPriceX96 → price)
- `get_reserves(pool_addr) -> dict` (pair-specific ABIs)
- `get_virtual_price(curve_pool_addr) -> float | None`
- Return typed dicts; include `block_number` for audit.

Cell 5 — MCP#2: Off-Chain Intel Tool

- `fetch_gov_posts(protocol) -> list[dict]` (title, body, ts, url)
- `fetch_status_updates(protocol) -> list[dict]` (status, msg, ts, url)
- Normalize text: strip, collapse whitespace, truncate long bodies.
- Combine docs with source metadata.

Cell 6 — Pipeline A: Feature Engineering

- For each pool:
- Get DEX price p_{dex} and Oracle p_{oracle}.
- Compute `dev`, add `reserves`, `virtual_price` if present.
- Append to a `DataFrame` with `pool`, `timestamp`, `block`, signals.
- Optional rolling window features if running looped sampling.

Cell 7 — Pipeline A: Anomaly Scoring

- Baseline with `IsolationForest` on `[dev, vol_proxy, reserve_delta]` (fill NA → 0).
- Add field `anom_score = -decision_function(X)` so **higher = weirder**.
- Rank pools by `anom_score`.

Cell 8 — Pipeline B: Docs → Events via LLM#1

- Concatenate latest docs (`gov + status`) into a compact JSON in the prompt.
- Prompt LLM#1 to emit **strict JSON list** of events (schema above).
- Use `safe_json_extract()` to parse; on failure, fallback to low-severity mock.

Cell 9 — Fusion & Reasoning via LLM#2

- Build compact JSON context: top numeric signals per pool + top-k events.
- Prompt template:
- Role = DeFi risk analyst.
- Tasks = risk score (0–100) + short note + 3 actions.
- Style = concise, references signals; avoid hallucinated protocols.

- Parse the answer into sections with simple regex guards; also save raw text.

Cell 10 — Visualization & Outputs

- Bar chart of `anom_score` per pool; color by threshold bucket.
- Pretty print **Analyst Note** block.
- Write artifacts to `outputs/`: `features.parquet`, `events.json`, `analyst_note.txt`.

Cell 11 — Evaluation Harness (Backtest Stubs)

- Define date windows and pools; collect samples at fixed intervals.
- Heuristic label: event day if `|dev| > τ` sustained across N samples.
- Metrics: Precision\@k for alert days, AUC-PR on anomaly labels, lead/lag (Δ between first alert and threshold breach).
- Ablations: on-chain only vs off-chain only vs fused; LLM#2 vs template note.

Cell 12 — Packaging & Export

- Save a `RUN_META.json` (models, rpc, block range, pools).
- Export analyst note to HTML; optional conversion to PDF via `weasyprint` / `wkhtmltopdf` if available in environment.

---

4) Prompts (Ready-to-use Templates)

LLM#1 (IE/NLU) — Docs → Events

> **System**: You extract risk events from DeFi governance and status updates. Output strict JSON only.
>
> **User**: From the following JSON array of documents, extract risk events with keys `[type, severity, ts, summary, source]`. Valid `type` values: `oracle_update`, `pause_notice`, `governance_vote`, `incident`, `parameter_change`. `severity`: 1 (info) to 5 (critical). Keep summaries ≤ 140 chars. Documents: `{{DOCS_JSON}}`.

LLM#2 (Reasoner/Writer) — Fusion → Risk & Actions

> **System**: You are a DeFi risk analyst. Be concise and factual.
>
> **User**: Given `onchain_signals` and `events`, produce: (1) `risk_score` 0–100, (2) a <200-word `analyst_note`, (3) an array `actions` of 3 items `{title, rationale}`. Emphasize DEX-oracle deviation, anomaly score, reserve/virtual_price shifts, and high-severity events. Data: `{{FUSION_JSON}}`.

---

5) Thresholds & Heuristics (Tunable)

- **Deviation Alerts:** |dev| ≥ 0.003 (0.3%) for stable-stable pairs as soft flag; higher tiers at 0.5%, 1.0%.
- **Anomaly Score:** Top decile per session or absolute threshold (e.g., ≥ 0.6 on scaled score).
- **Event Weighting:** Severity 4–5 events increase risk score floor (e.g., min 40–60).
- **Fusion Rule-of-Thumb:**
- Base risk from normalized `anom_score`.
- Add penalty for sustained positive |dev|.
- Add event bump based on max(severity) and recency decay.

---

6) Security, Reliability & Ethics

- **RPC Hygiene:** Rate-limit with retries; backoff on errors; verify chain id.
- **Determinism:** Cache raw RPC responses with block numbers for auditability.
- **Safety:** Treat LLM outputs as advisory; never auto-execute trades.
- **PII/Secrets:** No user data; store keys via env vars; avoid logging secrets.
- **Model Risk:** Provide a template (non-LLM) fallback analyst note to avoid total failure when models are unavailable.

---

7) Extensibility Roadmap

- **Hedge Simulator:** Size routes via 0x/1inch quote APIs to estimate cost vs risk reduction.
- **Cross-Chain Watch:** Compare bridged vs native stables for divergence.
- **TWAP/Vol:** Add Uniswap `observe()` TWAP, realized vol.
- **Explainability:** SHAP on anomaly features for interpretability.
- **Alerts:** Slack/Discord webhooks with throttling and dedupe keys.

---

8) Deliverables Checklist

- Single notebook `Depeg_Sentinel.ipynb` with the 12 cells above.
- `outputs/` folder with example artifacts from a sample run.
- `RUN_META.json` with reproducibility metadata (models, pools, blocks).
- `README.md` (brief) explaining setup and how to execute.
- Optional `export/analyst_note.pdf` if PDF conversion tools are present.

---

9) Risks & Validation Notes

- **Data Quality:** Some pools lack clean ABIs or differ by decimals; include per-pool adapters.
- **Oracle Delays:** Chainlink heartbeat can lag; compare with TWAP to avoid false positives.
- **Event Spam:** Governance chatter is noisy; keep LLM#1 conservative on severity.

- **Backtest Bias:** Use fixed lookbacks and lock data at historical blocks to avoid look-ahead.

---

10) Minimal Code Skeleton (Inline-Ready)

This section mirrors the exact functions you'll drop into the notebook cells. Swap real addresses/URLs.

```
# --- Cell 1: Env & LLM runner ---
import json, subprocess, textwrap, os, datetime as dt
import pandas as pd, numpy as np
from tenacity import retry, stop_after_attempt, wait_exponential

def run_ollama(model: str, prompt: str) -> str:
    try:
        return subprocess.run(["ollama", "run", model], input=prompt.encode(),
capture_output=True, check=True).stdout.decode()
    except Exception:
        return f"[MOCK::{model}] {prompt[:200]}..."

# --- Cell 2: Config ---
from pydantic import BaseModel
class Config(BaseModel):
    eth_rpc: str
    chainlink_feeds: dict
    pools: dict
    models: dict
    outdir: str = "outputs"
CFG = Config(
    eth_rpc=os.getenv("ETH_RPC", "https://mainnet.infura.io/v3/KEY"),
    chainlink_feeds={"USDC/USD": "0xFEED...", "DAI/USD": "0xFEED..."},
    pools={"USDC/USDT_univ3": {"address": "0xPOOL...", "type": "uniswap_v3"}},
    models={"ie": "phi3:mini", "reason": "mistral:instruct"}
)

# --- Cell 3: Utils ---
@retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=0.5))
def safe_request(url: str):
    import requests; return requests.get(url, timeout=10)

def safe_json_extract(s: str):
    a, b = s.find('['), s.rfind(']')
    return json.loads(s[a:b+1]) if a!=-1 and b!=-1 else []

# --- Cell 4: On-Chain Tool ---
from web3 import Web3
```

```
class OnChainDataTool:
    def __init__(self, rpc):
        self.w3 = Web3(Web3.HTTPProvider(rpc)); assert self.w3.is_connected()
    def get_oracle_price(self, feed):
        abi=[{"inputs":[],"name":"latestAnswer","outputs":
[{"type":"int256"}],"stateMutability":"view","type":"function"}]
        c=self.w3.eth.contract(address=Web3.to_checksum_address(feed), abi=abi)
        return float(c.functions.latestAnswer().call())/1e8
    def get_univ3_price(self, pool):
        abi=[{"inputs":[],"name":"slot0","outputs":[{"type":"uint160"},
{"type":"int24"},{"type":"uint16"},{"type":"uint16"},{"type":"uint16"},
{"type":"uint8"},{"type":"bool"}],"stateMutability":"view","type":"function"}]
        c=self.w3.eth.contract(address=Web3.to_checksum_address(pool), abi=abi)
        sqrtp=c.functions.slot0().call()[0]; return float((sqrtp/(2**96))**2)

onchain = OnChainDataTool(CFG.eth_rpc)

# --- Cell 5: Off-Chain Tool ---
class OffChainIntelTool:
    def fetch_gov_posts(self, protocol: str):
        return [{"protocol": protocol, "title": "Oracle heartbeat update",
"body": "Might increase heartbeat to 60s.", "ts":
dt.datetime.utcnow().isoformat(), "source": "mock://gov"}]
    def fetch_status_updates(self, protocol: str):
        return [{"protocol": protocol, "status": "degraded", "msg": "High
volatility on stables.", "ts": dt.datetime.utcnow().isoformat(), "source":
"mock://status"}]

offchain = OffChainIntelTool()

# --- Cell 6-7: Features + Anomaly ---
from sklearn.ensemble import IsolationForest

def compute_features():
    rows=[]
    for name, meta in CFG.pools.items():
        dex_px = onchain.get_univ3_price(meta["address"]) if
meta["type"]=="uniswap_v3" else np.nan
        sym = "USDC/USD" if "USDC" in name else "DAI/USD"
        oracle_px = onchain.get_oracle_price(CFG.chainlink_feeds[sym])
        dev=(dex_px-oracle_px)/oracle_px

rows.append({"pool":name,"dex_px":dex_px,"oracle_px":oracle_px,"dev":dev,"ts":dt.datetime.utcnow(
    return pd.DataFrame(rows)

def anomaly_score(df):
    X=df[["dev"]].fillna(0.0).values; clf=IsolationForest(contamination=0.05,
```

```
random_state=0).fit(X)
    df["anom_score"] = -clf.decision_function(X); return df

feat = anomaly_score(compute_features())

# --- Cell 8: LLM#1 IE ---
import textwrap

def ie_extract_events(docs):
    prompt = textwrap.dedent(f"""
    Extract risk events from the following JSON array. Output JSON list with
keys: type, severity, ts, summary, source.
    Docs: {json.dumps(docs)[:3000]}
    """)
    raw = run_ollama(CFG.models["ie"], prompt)
    ev = safe_json_extract(raw)
    return ev or [{"type":"oracle_update","severity":2,"ts":docs[0]
["ts"],"summary":"mock","source":docs[0].get("source","mock") }]

# --- Cell 9: LLM#2 Reasoner ---

def reason_and_write(feats, events):
    context = {"onchain": feats.to_dict(orient="records"), "events": events}
    prompt = textwrap.dedent(f"""
    You are a DeFi risk analyst. Return three JSON keys: risk_score (0-100),
analyst_note (string < 200 words), actions (array of 3 items {{title,
rationale}}).
    Data: {json.dumps(context)[:4000]}
    """)
    return run_ollama(CFG.models["reason"], prompt)

# --- Cell 10: Viz & Outputs ---
import matplotlib.pyplot as plt

gov = offchain.fetch_gov_posts("curve"); sts =
offchain.fetch_status_updates("curve")
events = ie_extract_events(gov+sts)
report = reason_and_write(feat, events)

plt.figure(figsize=(6,3))
plt.bar(feat["pool"], feat["anom_score"], color="#2dd4bf"); plt.title("Anomaly
score by pool"); plt.xticks(rotation=20); plt.tight_layout(); plt.show()

os.makedirs(CFG.outdir, exist_ok=True)
feat.to_parquet(os.path.join(CFG.outdir, "features.parquet"))
open(os.path.join(CFG.outdir, "events.json"),"w").write(json.dumps(events,
indent=2))
open(os.path.join(CFG.outdir, "analyst_note.txt"),"w").write(report)
```

11) How to Run & Export

1. Open the notebook, set `ETH_RPC`, pool/oracle addresses, and model names.
2. Run cells in order; verify the environment cell confirms RPC connectivity and (optional) local models.
3. Inspect `feat` table and chart; read the **Analyst Note** printout.
4. Artifacts appear in `outputs/`.
5. (Optional) Use a final cell to convert the analyst note or the whole notebook to PDF if your environment supports it.

---

12) Acceptance Criteria (Definition of Done)

- Single notebook runs start-to-finish with mocks **even without** local models; with local models, emits structured events and a coherent analyst note.
- On-chain features include at least: `dex_px`, `oracle_px`, `dev`, `anom_score` (and block/ts metadata).
- Off-chain events include at least one parsed event with severity and timestamp.
- Fusion produces a numeric `risk_score` and 2–3 actionable recommendations.
- Artifacts are written to disk and are re-runnable for audit.

---

13) RAG-Enable Both Pipelines (2× RAG, still single notebook)

You want Retrieval-Augmented Generation driving the brains, not just vibes. We'll attach **two lightweight RAG indices**—one per pipeline—so each LLM agent retrieves grounded context before it speaks.

13.1 RAG Overview

- **RAG-A (On-Chain Knowledge Base):** Static/domain docs to help interpret numeric signals.
- *Content:* Uniswap v3 math quickref (ticks, sqrtPriceX96), Curve `virtual_price` mechanics, Chainlink heartbeat/round logic, stablecoin mechanics (USDC/USDT/DAI peg behavior), past incident briefs (Terra/UST, USDC depeg March 2023).
- *Purpose:* Give LLM#2 precise definitions and playbooks so its notes/actions are grounded.
- **RAG-B (Off-Chain Rolling Intel):** Fresh governance + status snippets with metadata.
- *Content:* The latest fetched posts (titles/bodies), status pings, short news blurbs; each chunk stamped with `ts`, `source`, `protocol`.
- *Purpose:* Let LLM#1 extract events with direct citations and reduce hallucinations.

Both indices live **in-notebook** using FAISS (or DuckDB+SQLite fallback). Embeddings via a small model (e.g., `sentence-transformers/all-MiniLM-L6-v2` or `intfloat/e5-small-v2`).

13.2 Document Schemas

```
// RAG-A docs (static)
{
  "kb": "onchain",
  "title": "Uniswap v3 Slot0 and Price",
  "text": "sqrtPriceX96… price = (sqrtP/2^96)^2 … tick spacing …",
  "tags": ["uniswap", "math", "price"],
  "version": "v1",
  "source": "internal://kb/univ3"
}

// RAG-B docs (dynamic)
{
  "kb": "offchain",
  "protocol": "curve",
  "ts": "2025-08-16T12:34:56Z",
  "title": "Oracle heartbeat update",
  "text": "Increasing heartbeat to 60s… temporary circuit breaker…",
  "source": "https://forum.curve.fi/..."
}
```

13.3 Chunking & Indexing

- **Chunk size:** 500–800 tokens for technical refs; 200–400 for short posts.
- **Overlap:** 50–100 tokens to preserve context.
- **Metadata in vector store:** `kb`, `protocol`, `ts`, `source`, `tags`, `title`.
- **Freshness:** For RAG-B, keep a 7–14 day window; apply recency boost at query time.

13.4 Code Cells (additions)

Cell 13a — Embeddings & Vector Store

```
# %pip install faiss-cpu sentence-transformers duckdb
from sentence_transformers import SentenceTransformer
import faiss, duckdb, numpy as np

EMB = SentenceTransformer("all-MiniLM-L6-v2")

class SimpleRAG:
    def __init__(self, dim=384):
        self.index = faiss.IndexFlatIP(dim)
        self.docs = []  # list of dicts with {id, text, meta}
        self.mat = None
    def add(self, docs):
        vecs = EMB.encode([d["text"] for d in docs], normalize_embeddings=True)
```

```
        if self.mat is None:
            self.mat = vecs
        else:
            self.mat = np.vstack([self.mat, vecs])
        start_id = len(self.docs)
        for i, d in enumerate(docs):
            d["id"] = start_id + i
            self.docs.append(d)
        self.index.reset(); self.index.add(self.mat.astype("float32"))
    def search(self, query, k=5, where=None, recency_boost=False):
        qv = EMB.encode([query], normalize_embeddings=True).astype("float32")
        sims, idx = self.index.search(qv, k)
        results = []
        for rank, j in enumerate(idx[0].tolist()):
            if j == -1: continue
            d = self.docs[j]
            if where and not all(d.get(k)==v for k,v in where.items()):
                continue
            score = float(sims[0][rank])
            if recency_boost and d.get("ts"):
                # simple time decay boost
                score += 0.05
            results.append({"text": d["text"], "meta": {k:v for k,v in d.items()
if k!="text"}, "score": score})
        return results[:k]

RAG_A = SimpleRAG()
RAG_B = SimpleRAG()
```

Cell 13b — Load KB (RAG-A) & Ingest Dynamic Posts (RAG-B)

```
# Static KB examples (replace with your curated snippets)
kb_docs = [
  {"kb":"onchain","title":"Univ3 price math","text":"In Uniswap v3,
price_token1_token0 = (sqrtPriceX96 / 2**96)^2 …", "source":"internal://kb/
univ3"},
  {"kb":"onchain","title":"Curve virtual_price","text":"virtual_price reflects
the current pool invariant relative to deposits …", "source":"internal://kb/
curve"},
  {"kb":"onchain","title":"Chainlink heartbeat","text":"Chainlink feeds update
on new rounds or heartbeat if deviation/heartbeat thresholds are met …",
"source":"internal://kb/chainlink"},
]
RAG_A.add(kb_docs)

# Dynamic posts from Cell 10 offchain fetchers
```

```
raw_docs = offchain.fetch_gov_posts("curve") +
offchain.fetch_status_updates("curve")
rag_b_docs =
[{"kb":"offchain","protocol":d.get("protocol","misc"),"ts":d.get("ts"),"title":d.get("title",
d.get("status","update")),"text":(d.get("body") or
d.get("msg")),"source":d.get("source","mock://") } for d in raw_docs]
RAG_B.add(rag_b_docs)
```

13.5 Wiring RAG into LLM Calls

LLM#1 (IE/NLU) now with context retrieval from RAG-B

```
def ie_extract_events_with_rag(docs, k=5):
    # Build a query from titles+first lines
    q = " ".join((d.get("title","")) for d in docs)[:500]
    ctx = RAG_B.search(q, k=k, where={"kb":"offchain"}, recency_boost=True)
    prompt = f"""
    Use the CONTEXT to extract risk events from DOCS. Output strict JSON list
with keys: type, severity, ts, summary, source.
    CONTEXT:
{json.dumps(ctx, indent=2)[:2000]}

    DOCS:
{json.dumps(docs)[:3000]}
    """
    raw = run_ollama(CFG.models["ie"], prompt)
    ev = safe_json_extract(raw)
    return ev or [{"type":"information","severity":
1,"ts":docs[0].get("ts"),"summary":"(fallback)","source":docs[0].get("source","mock") }]
```

LLM#2 (Reasoner/Writer) now with RAG-A (definitions/playbooks) + top recent events

```
def reason_and_write_with_rag(feats, events, k=5):
    # Query KB based on signals we see (e.g., large |dev|, curve pools etc.)
    q_terms = [f"deviation {abs(r['dev']):.4f}" for r in
feats.to_dict(orient="records")]
    q = "; ".join(q_terms) + "; stablecoin depeg; curve virtual_price; uniswap
v3"
    ctx = RAG_A.search(q, k=k, where={"kb":"onchain"})
    context = {"onchain": feats.to_dict(orient="records"), "events": events,
"kb": ctx}
    prompt = f"""
    You are a DeFi risk analyst. Use KB CONTEXT when explaining signals and
proposing actions. Return JSON with keys: risk_score (0-100), analyst_note (<200
words), actions (3 items {{title, rationale, source_ref}}). Cite relevant KB
```

```
`source` or event `source` in `source_ref`.
    DATA:
{json.dumps(context)[:4000]}
    """
    return run_ollama(CFG.models["reason"], prompt)
```

Replace previous calls to `ie_extract_events` and `reason_and_write` with the `*_with_rag` versions to activate RAG.

13.6 RAG-Specific Evaluation

- **Retrieval hit-rate\@k:** % of generated events/notes that include at least one citation from the correct protocol/topic.
- **Context relevance score:** Manual or heuristic cosine sim between query and retrieved chunks.
- **Ablation:** LLM with vs without RAG context; measure change in precision of event typing and correctness of definitions.

13.7 Caching & Freshness

- Cache embeddings to disk (`outputs/embeddings.npz` + `docs.json`).
- On each run, only re-embed new dynamic posts; rebuild FAISS index if doc count changed.
- Apply time decay in ranking for RAG-B to privilege recent posts.

13.8 Safety & Leakage

- Keep the KB concise and vetted; avoid including speculative blogspam as “ground truth”.
- Force JSON output and parse strictly; if invalid, downgrade severity or fall back to template note.

End of PDF-ready plan.