# nexus\_pipeline\_v2.py

import json

import re

import hmac

import math

import hashlib

import logging

from dataclasses import dataclass, field

from typing import Any, Dict, List, Optional, Tuple, Callable

from statistics import mean, pstdev

# -------------- Logging --------------

logging.basicConfig(level=logging.INFO, format="%(message)s")

log = logging.getLogger("nexus.v2")

# -------------- Config (no magic numbers; overridable) --------------

@dataclass

class Config:

# Integrity

hmac\_key: bytes = b"CHANGE\_ME\_IN\_PROD" # set via env/secret manager in real deployments

enable\_ledger: bool = True

# Payload limits (explicit)

max\_tokens: int = 20000

max\_bytes: int = 200\_000

# Entropy threshold: derived from calibration or fallback.

# Rationale: English text char-entropy ≈ 3.5–4.5 bits/char; base64/random > ~5.5.

entropy\_threshold\_bits\_per\_char: float = 5.5

# Vector modulus rationale: bounding hashed-word integers improves numeric stability and reproducibility.

vector\_modulus: int = 1\_000\_003 # prime near 1e6; avoids some aliasing vs 10\*\*6

# Decision/routing: we removed randomness; deterministic DAG instead.

roles: Tuple[str, ...] = ("DATA\_ANALYST", "CREATIVE\_ENGINE", "ETHICAL\_GOVERNOR")

# Trust policy

allow\_domains: Tuple[str, ...] = ("noaa.gov", "nasa.gov", "who.int", "ipcc.ch", "epa.gov", "un.org")

deny\_domains: Tuple[str, ...] = ("clickbait.example", "fake-news.example")

CFG = Config()

# -------------- Utils --------------

\_URL\_RE = re.compile(r"\bhttps?://[^\s<>\"')]+", re.I)

\_WORD\_RE = re.compile(r"\w+", re.U) # unicode wordish

def stable\_hash\_int(text: str, bits: int = 64) -> int:

digest = hashlib.blake2b(text.encode("utf-8"), digest\_size=bits // 8).digest()

return int.from\_bytes(digest, "big", signed=False)

def tokenize(text: str) -> List[str]:

# Better than “strip all non-alnum”: Unicode-aware, preserves word boundaries.

return [w.lower() for w in \_WORD\_RE.findall(text)]

def shannon\_entropy(s: str) -> float:

if not s:

return 0.0

freq: Dict[str, int] = {}

for ch in s:

freq[ch] = freq.get(ch, 0) + 1

n = len(s)

return -sum((c/n) \* math.log2(c/n) for c in freq.values())

def hmac\_sha256(payload: Any, key: bytes) -> str:

data = json.dumps(payload, sort\_keys=True, separators=(",", ":")).encode("utf-8")

return hmac.new(key, data, hashlib.sha256).hexdigest()

def verify\_hmac(payload: Any, key: bytes, signature: str) -> bool:

expected = hmac\_sha256(payload, key)

return hmac.compare\_digest(expected, signature)

def extract\_domains(text: str) -> List[str]:

ds: List[str] = []

for url in \_URL\_RE.findall(text or ""):

host = url.split("://", 1)[-1].split("/", 1)[0].lower()

ds.append(host)

return ds

def size\_ok(text: str) -> bool:

return len((text or "").encode("utf-8")) <= CFG.max\_bytes

# -------------- Pluggable sentiment --------------

# Default tiny lexicon (deterministic, auditable). Swap with VADER/AFINN via SENTIMENT\_FN.

POS = {"progress","improve","benefit","clean","reduce","solution","hope","advance","protect","resilient",

"ethical","love","growth","efficient","renewable","stewardship"}

NEG = {"crisis","threat","risk","harm","pollution","drought","flood","wildfire","heatwave","fear","collapse",

"scarcity","loss","damage","catastrophe","disaster"}

def simple\_sentiment(text: str) -> float:

toks = tokenize(text)[:CFG.max\_tokens]

p = sum(t in POS for t in toks)

n = sum(t in NEG for t in toks)

return 0.0 if (p + n) == 0 else (p - n) / (p + n)

SENTIMENT\_FN: Callable[[str], float] = simple\_sentiment # inject better scorer as needed

# -------------- Nexus (consistent, pre+post gating) --------------

@dataclass

class NexusAnnotation:

ok: bool

notes: List[str] = field(default\_factory=list)

domains: List[str] = field(default\_factory=list)

entropy: float = 0.0

class Nexus:

absolutists = ("guarantee","proves","undeniable","certain","will happen","no doubt","without question")

sensitive = ("SSN","password","private key","mnemonic","seed phrase")

@staticmethod

def gate(kind: str, text: str) -> NexusAnnotation:

notes: List[str] = []

# Size / entropy

if not size\_ok(text):

notes.append("payload\_too\_large")

H = shannon\_entropy(text or "")

if H > CFG.entropy\_threshold\_bits\_per\_char:

notes.append(f"high\_entropy:{H:.2f}")

# Absolutism & sensitive

low = (text or "").lower()

if any(a in low for a in Nexus.absolutists):

notes.append("absolutist\_claims")

if any(s in (text or "") for s in Nexus.sensitive):

notes.append("sensitive\_marker")

# URLs & trust

doms = extract\_domains(text)

denied = [d for d in doms if any(d.endswith(x) for x in CFG.deny\_domains)]

allowed = [d for d in doms if any(d.endswith(x) for x in CFG.allow\_domains)]

if denied:

notes.append(f"deny\_domains:{denied}")

if allowed:

notes.append(f"allow\_domains:{allowed}")

ok = len([n for n in notes if not n.startswith("allow\_domains:")]) == 0

# “ok” means no blocking issues (we don’t block on allow\_domains presence)

return NexusAnnotation(ok=ok, notes=notes, domains=doms, entropy=H)

# -------------- Ledger (optional) --------------

@dataclass

class LedgerRec:

idx: int

stage: str

payload: Dict[str, Any]

sig: str # HMAC over payload

prev\_sig: str

class Ledger:

def \_\_init\_\_(self, key: bytes):

self.key = key

self.recs: List[LedgerRec] = []

def append(self, stage: str, payload: Dict[str, Any]) -> LedgerRec:

prev = self.recs[-1].sig if self.recs else "0"\*64

chained = {"stage": stage, "payload": payload, "prev": prev}

sig = hmac\_sha256(chained, self.key)

rec = LedgerRec(idx=len(self.recs), stage=stage, payload=payload, sig=sig, prev\_sig=prev)

self.recs.append(rec)

return rec

def verify(self) -> bool:

prev = "0"\*64

for rec in self.recs:

chained = {"stage": rec.stage, "payload": rec.payload, "prev": prev}

if not verify\_hmac(chained, self.key, rec.sig):

return False

prev = rec.sig

return True

# -------------- Agents (SRP: each does one job) --------------

class DataAnalyst:

def run(self, task\_text: str) -> Dict[str, Any]:

toks = tokenize(task\_text)[:CFG.max\_tokens]

# Deterministic integer series from word hashes

vals = [(stable\_hash\_int(t, 64) % CFG.vector\_modulus) for t in toks]

n = len(vals)

l2 = math.sqrt(sum(v\*v for v in vals)) if n else 0.0

avg = mean(vals) if n else 0.0

var = (pstdev(vals)\*\*2) if n > 1 else 0.0

sent = SENTIMENT\_FN(task\_text)

return {

"metrics": {"n\_tokens": n, "l2\_norm": l2, "mean": avg, "variance": var},

"sentiment": {"score": sent}

}

class CreativeEngine:

def run(self, analysis: Dict[str, Any], task\_text: str) -> Dict[str, Any]:

sent = float(analysis.get("sentiment", {}).get("score", 0.0))

n\_tokens = int(analysis.get("metrics", {}).get("n\_tokens", 0))

tone = "cautiously optimistic" if sent > 0.1 else ("concerned" if sent < -0.1 else "balanced")

draft = (

f"Creative summary: A {tone} synthesis of {n\_tokens} token(s). "

f"Signal suggests sentiment={sent:+.3f}. "

"Focus: clear benefits, documented risks, and actionable steps."

)

# Small deterministic artifact: 3×3 integer covariance-ish via simple recurrences

seed = stable\_hash\_int(json.dumps(analysis, sort\_keys=True), 64) % 10\_000\_000\_007

mat = []

x = seed % 1\_000\_000 + 1

for \_ in range(3):

row = []

for \_ in range(3):

x = (1103515245 \* x + 12345) % 2\*\*31

row.append(x % 1000)

mat.append(row)

# “Covariance” by mat \* mat^T in integers

cov = [[sum(mat[i][k]\*mat[j][k] for k in range(3)) for j in range(3)] for i in range(3)]

return {"draft": draft, "artifacts": {"covariance3x3\_int": cov}}

class EthicalGovernor:

def run(self, draft\_text: str) -> Dict[str, Any]:

issues: List[str] = []

low = draft\_text.lower()

# Absolutism

if any(a in low for a in Nexus.absolutists):

issues.append("Avoid absolute certainty; qualify claims with evidence.")

# “Actionable steps” require sources

if "actionable steps" in low and not \_URL\_RE.search(draft\_text):

issues.append("Cite sources for recommendations or mark as opinion.")

# Sensitive tokens

if any(s in draft\_text for s in Nexus.sensitive):

issues.append("Remove sensitive credential references.")

# Size

if not size\_ok(draft\_text):

issues.append("Draft too long; reduce or link out.")

decision = "APPROVED" if not issues else "REVISION\_REQUIRED"

return {"decision": decision, "issues": issues}

# -------------- Orchestrator (explicit DAG; fail-fast) --------------

class Pipeline:

def \_\_init\_\_(self, cfg: Config = CFG):

self.cfg = cfg

self.ledger = Ledger(cfg.hmac\_key) if cfg.enable\_ledger else None

self.analyst = DataAnalyst()

self.creative = CreativeEngine()

self.ethics = EthicalGovernor()

def \_gate(self, stage: str, text: str) -> NexusAnnotation:

ann = Nexus.gate(stage, text)

# Fail-fast on hard violations (size, entropy, sensitive markers, deny domains)

hard = [n for n in ann.notes if any(k in n for k in ("payload\_too\_large","high\_entropy","sensitive\_marker","deny\_domains"))]

if hard:

raise ValueError(f"Gating failed at {stage}: {hard}")

return ann

def run(self, task\_text: str) -> Dict[str, Any]:

if not isinstance(task\_text, str) or not task\_text.strip():

raise ValueError("Task must be a non-empty string.")

# TASK gate (pre)

task\_ann = self.\_gate("TASK", task\_text)

out: Dict[str, Any] = {"task\_gate": task\_ann.\_\_dict\_\_}

# ANALYST

analysis = self.analyst.run(task\_text)

if self.ledger: self.ledger.append("ANALYSIS", analysis)

# CREATIVE (pre/post gate)

pre\_cre = self.\_gate("PRE\_CREATIVE", task\_text) # gate on the same input context

draft\_pack = self.creative.run(analysis, task\_text)

post\_cre = Nexus.gate("DRAFT", draft\_pack.get("draft",""))

out["creative\_gate"] = {"pre": pre\_cre.\_\_dict\_\_, "post": post\_cre.\_\_dict\_\_}

if self.ledger: self.ledger.append("DRAFT", draft\_pack)

# ETHICS (pre/post)

pre\_eth = self.\_gate("PRE\_ETHICS", draft\_pack.get("draft",""))

ethics = self.ethics.run(draft\_pack.get("draft",""))

post\_eth = Nexus.gate("ETHICS", json.dumps(ethics, sort\_keys=True))

out["ethics\_gate"] = {"pre": pre\_eth.\_\_dict\_\_, "post": post\_eth.\_\_dict\_\_}

if self.ledger: self.ledger.append("ETHICS", ethics)

# Integrity (real: signed, verifiable)

out["analysis"] = analysis

out["draft"] = draft\_pack

out["ethics"] = ethics

if self.ledger:

out["ledger\_ok"] = self.ledger.verify()

out["ledger\_len"] = len(self.ledger.recs)

out["ledger\_head"] = self.ledger.recs[-1].sig if self.ledger.recs else "0"\*64

else:

out["ledger\_ok"] = False

out["ledger\_len"] = 0

out["ledger\_head"] = None

# Attach verifiable HMAC signature for the whole bundle

out["signature"] = hmac\_sha256({"analysis": analysis, "draft": draft\_pack, "ethics": ethics}, self.cfg.hmac\_key)

return out

# -------------- Main --------------

if \_\_name\_\_ == "\_\_main\_\_":

task = "Analyze public updates on coastal flooding and produce a clear summary with actionable steps referencing https://www.noaa.gov and https://www.epa.gov ."

p = Pipeline()

result = p.run(task)

print(json.dumps(result, indent=2))