got you. here’s a single, self-contained “save everything we built today” script. drop it in your repo root as save_all_aes.sh, run it, and it will write every file (policies, evolve runner, workflow with badge/history, validator, docs dashboard, signing tools, example request). nothing else required.
save_all_aes.sh # One-shot writer for AES v1.1 files + dashboard (copy/paste as-is, run from repo root). # Usage: # bash save_all_aes.sh # Afterwards (optional): # git add -A && git commit -m "Add AES v1.1 stack" && git push set -euo pipefail # --- layout --- mkdir -p policies xova .github/workflows tools docs docs/history examples # --- policies/default.json --- cat > policies/default.json <<'JSON' { "security": { "require_signatures": true }, "limits": { "time_s": 15 }, "selection": { "prefer": "highest_version" }, "thresholds": { "max": { "latency_ms": 600 }, "min": { "order_k2": 0.10, "order_k3": 0.05 } }, "post_chain": [{ "api":

"codex.analysis/order", "plugin": "codex.order_meter" }], "grading": { "penalties": { "bad-signature": 100, "timeout": 50, "exception": 40, "bad-metrics": 30 } } } JSON # --- xova/evolve.py --- cat > xova/evolve.py <<'PY' #!/usr/bin/env python3 import json, os, sys, time, hashlib from pathlib import Path ROOT = Path(__file__).resolve().parents[1] PLUGINS = ROOT / "plugins" OUTDIR = ROOT / "out" POLICY = ROOT / "policies" / "default.json" from registry import load_registry, choose_candidates from sandbox import run_plugin def load_policy(): with open(POLICY, "r", encoding="utf-8") as f: return json.load(f) def sha256_manifest_without_sig(man: dict) -> str: m2 = man.copy() m2.pop("signature", None) raw = json.dumps(m2, sort_keys=True, separators=(",",":")).encode() return hashlib.sha256(raw).hexdigest() def verify_manifest(man): want =

```
man.get("signature","") if not want: return False got =
sha256_manifest_without_sig(man) return got == want def passes_thresholds(metrics,
thr): if not thr: return True mx = thr.get("max", {}) mn = thr.get("min", {}) for k, v in
mx.items(): if k in metrics and metrics[k] > v: return False for k, v in mn.items(): if k in
metrics and metrics[k] < v: return False return True def rank_candidates(cands, prefer): if
prefer == "highest_version": def vparts(x): s = x["manifest"].get("version","0.0.0") return
tuple(int(p) if p.isdigit() else 0 for p in s.split(".")) return sorted(cands, key=vparts,
reverse=True) return cands def _write_summaries(chosen_name, man, metrics, api): """Persist run metadata for
dashboards/badges.""" OUTDIR.mkdir(exist_ok=True) summary = { "chosen": chosen_name, "version":
man.get("version", ""), "metrics": metrics, "timestamp": time.time(), "request_api": api,
```

"commit": os.environ.get("GITHUB_SHA", "") } (OUTDIR / "summary.json").write_text(json.dumps(summary, indent=2), encoding="utf-8") (OUTDIR / "metrics.json").write_text(json.dumps(metrics, separators=(",",":")), encoding="utf-8") def evolve(request_path: str): OUTDIR.mkdir(exist_ok=True) with open(request_path, "r", encoding="utf-8") as f: req = json.load(f) policy = load_policy() registry = load_registry(PLUGINS) api = req["api"] caps = req.get("capabilities", []) params = req.get("params", {}) cands = choose_candidates(registry, api, caps) cands = rank_candidates(cands, policy.get("selection", {}).get("prefer")) if not cands: print(json.dumps({"status": "failed", "errors": [f"No plugin matches api={api} caps={caps}"]}, indent=2)) return 1 errors = [] for cand in cands: man = cand["manifest"] if policy["security"].get("require_signatures",

```
False): if not verify_manifest(man): errors.append({"plugin": cand["name"], "error": "bad-signature"}) continue time_limit_s = float(man.get("time_limit_s", policy["limits"]["time_s"])) t0 = time.time() try: res = run_plugin(cand, params, OUTDIR, time_limit_s=time_limit_s) except Exception as e: errors.append({"plugin": cand["name"], "error": f"exception:{e}"}) continue latency_ms = (time.time() - t0) * 1000.0 metrics = res.get("metrics", {}) or {} metrics["latency_ms"] = latency_ms for link in policy.get("post_chain", []): dep = registry.get(link["plugin"]) if not dep: errors.append({"plugin": cand["name"], "warning": f"missing-post:{link['plugin']}"}) continue try: ar = run_plugin(dep, {"wav": str(OUTDIR / "sequence.wav"), "k_values": [2,3]}, OUTDIR, time_limit_s=time_limit_s) metrics.update(ar.get("metrics", {}) or {}) except Exception as e: errors.append({"plugin": cand["name"], "error":
```

f"post-exception:{e}"}) if not passes_thresholds(metrics, policy.get("thresholds", {})): errors.append({"plugin": cand["name"], "error": "bad-metrics", "metrics": metrics}) continue _write_summaries(cand["name"], man, metrics, api) print(json.dumps({ "status": "ok", "chosen": cand["name"], "version": man.get("version",""), "artifacts": res.get("artifacts", {}), "metrics": metrics, "tried": [x["name"] for x in cands] }, indent=2)) return 0 print(json.dumps({"status": "failed", "errors": errors}, indent=2)) return 1 if __name__ == "__main__": if len(sys.argv) < 2: print("Usage: python3 xova/evolve.py examples/request_nine.json") sys.exit(2) sys.exit(evolve(sys.argv[1])) PY chmod +x xova/evolve.py # --- .github/workflows/evolve.yml (guard + badge + history + docs sync) --- cat > .github/workflows/evolve.yml <<'YML' name: Evolve Build & Deploy on: push: branches: [main]

workflow_dispatch: permissions: contents: write concurrency: group: evolve-${{ github.ref }} cancel-in-progress: true jobs: evolve: runs-on: ubuntu-latest steps: - uses: actions/checkout@v4 - name: Guard id: guard run: | set -euo pipefail SKIP="false" MSG="$(jq -r '.head_commit.message // ""' "$GITHUB_EVENT_PATH" 2>/dev/null || true)" ACTOR="${GITHUB_ACTOR:-}" if echo "$MSG" | grep -qi '\[skip aes\]'; then SKIP="true"; fi if ["$ACTOR" = "github-actions[bot]"]; then SKIP="true"; fi echo "skip=$SKIP" >> "$GITHUB_OUTPUT" - name: Setup Python if: steps.guard.outputs.skip != 'true' uses: actions/setup-python@v5 with: python-version: '3.x' - name: Auto-sign plugin manifests (optional) if: steps.guard.outputs.skip != 'true' run: | python3 tools/auto_sign_manifests.py || true - name: Run Evolution if: steps.guard.outputs.skip != 'true' run: |

python3 xova/evolve.py examples/ request_nine.json || true ls -la out/ || true - name: Validate metrics (non-fatal) if: steps.guard.outputs.skip != 'true' id: validate run: | python3 tools/validate_metrics.py -- policy policies/default.json --metrics out/ metrics.json || true - name: Badge + history + sync docs if: steps.guard.outputs.skip != 'true' run: | set -euo pipefail mkdir -p docs docs/history python3 - <<'PY' import json, time, os, shutil, pathlib root = pathlib.Path("."); docs = root/"docs"; hist = docs/"history" policy = json.loads((root/"policies"/"default.json").read_text(encoding="utf-8")) if (root/"policies"/"default.json").exists() else {} metrics = json.loads((root/"out"/"metrics.json").read_text(encoding="utf-8")) if (root/"out"/"metrics.json").exists() else {} summary = json.loads((root/"out"/"summary.json").read_t

```
ext(encoding="utf-8")) if (root/"out"/"summary.json").exists() else {} # thresholds thr = policy.get("thresholds",{}); mx = thr.get("max",{}) or {}; mn = thr.get("min",{}) or {} def ge(k, d=0.0): try: return float(metrics.get(k, float('-inf'))) >= float(mn.get(k,d)) except: return False def le(k, d=1e9): try: return float(metrics.get(k, float('inf'))) <= float(mx.get(k,d)) except: return False overall = le("latency_ms",600) and ge("order_k2",0.10) and ge("order_k3",0.05) badge = {"schemaVersion":1,"label":"AES","message":"pass" if overall else "fail","color":"brightgreen" if overall else "red"} (docs/"badge.json").write_text(json.dumps(badge, separators=(",",":")), encoding="utf-8") # archive history ts = time.strftime("%Y%m%dT%H%M%SZ", time.gmtime()); dst = hist/ts dst.mkdir(parents=True, exist_ok=True) for name in
```

("metrics.json","summary.json","sequence.csv", "sequence.wav"): src = root/"out"/name if src.exists(): shutil.copy2(src, dst/src.name) # update history index idx_p = hist/"history_index.json" index = [] if idx_p.exists(): try: index = json.loads(idx_p.read_text(encoding="utf-8")) except: index = [] entry = {"timestamp": ts, "commit": os.environ.get("GITHUB_SHA",""), "chosen": summary.get("chosen",""), "version": summary.get("version",""), "metrics": metrics} index.append(entry) idx_p.write_text(json.dumps(index, indent=2), encoding="utf-8") PY [-f docs/ aes.html] || cp docs/index.html docs/ aes.html || true cp -f out/metrics.json docs/ 2>/dev/null || true cp -f out/summary.json docs/ 2>/dev/null || true cp -f out/ sequence.csv docs/ 2>/dev/null || true cp -f out/sequence.wav docs/ 2>/dev/null || true git config user.name "github-actions[bot]" git config user.email "41898282+github-

actions[bot]@users.noreply.github.com" git add docs git commit −m "Sync AES artifacts, badge, and history [skip ci] [skip aes]" || echo "No changes to commit" git push − name: Upload out/ artifact if: steps.guard.outputs.skip != 'true' uses: actions/upload−artifact@v4 with: name: evolve−output path: out/ retention−days: 7 YML # −−− tools/validate_metrics.py −−− cat > tools/validate_metrics.py <<'PY' #!/usr/bin/ env python3 import json, sys, argparse, pathlib def load_json(p): p = pathlib.Path(p) if not p.exists(): return {} try: return json.loads(p.read_text(encoding="utf−8")) except Exception: return {} def main(): ap = argparse.ArgumentParser() ap.add_argument("−−policy", required=True) ap.add_argument("−−metrics", required=True) ap.add_argument("−−json", action="store_true") args = ap.parse_args() policy = load_json(args.policy) metrics = load_json(args.metrics) thr =

policy.get("thresholds", {}) mx = thr.get("max", {}) if isinstance(thr.get("max", {}), dict) else {} mn = thr.get("min", {}) if isinstance(thr.get("min", {}), dict) else {} reasons, ok = [], True for k,v in mx.items(): if isinstance(metrics.get(k), (int,float)) and metrics[k] > v: ok = False; reasons.append(f"{k}>{v} (got {metrics[k]})") for k,v in mn.items(): if isinstance(metrics.get(k), (int,float)) and metrics[k] < v: ok = False; reasons.append(f"{k}<{v} (got {metrics[k]})") if args.json: print(json.dumps({"pass": ok, "reasons": reasons, "metrics": metrics}, separators=(",",":"))) else: print("AES metrics: " + ("PASS" if ok else "FAIL — " + "; ".join(reasons))) return 0 if __name__ == "__main__": sys.exit(main()) PY chmod +x tools/validate_metrics.py # --- tools/sign_manifest.py --- cat > tools/sign_manifest.py <<'PY' #!/usr/bin/env python3 import json, sys, hashlib, pathlib def

```
sign(path: str, in_place: bool): p = pathlib.Path(path) man = json.loads(p.read_text(encoding="utf-8")) man.pop("signature", None) raw = json.dumps(man, sort_keys=True, separators=(",",":")).encode() sig = hashlib.sha256(raw).hexdigest() if in_place: man["signature"] = sig p.write_text(json.dumps(man, indent=2), encoding="utf-8") print(f"Signed {p} -> signature={sig}") else: print(sig) if __name__ == "__main__": if len(sys.argv) < 2: print("Usage: python3 tools/sign_manifest.py <manifest.json> [--in-place]") sys.exit(2) sign(sys.argv[1], "--in-place" in sys.argv[2:]) PY chmod +x tools/sign_manifest.py # --- tools/auto_sign_manifests.py --- cat > tools/auto_sign_manifests.py <<'PY' #!/usr/bin/env python3 import json, hashlib, pathlib, sys PLUGINS = pathlib.Path("plugins") def sign_obj(obj): o2 = dict(obj)
```

o2.pop("signature", None) raw = json.dumps(o2, sort_keys=True, separators=(",",":")).encode() return hashlib.sha256(raw).hexdigest() def main(): if not PLUGINS.exists(): print("No plugins/ directory; skipping.") return 0 changed = 0 for man_path in PLUGINS.rglob("manifest.json"): data = json.loads(man_path.read_text(encoding="utf-8")) sig0 = data.get("signature") sig1 = sign_obj(data) if sig0 != sig1: data["signature"] = sig1 man_path.write_text(json.dumps(data, indent=2), encoding="utf-8") print(f"Signed/ updated: {man_path}") changed += 1 print(f"Done. {changed} manifest(s) updated.") return 0 if __name__ == "__main__": sys.exit(main()) PY chmod +x tools/auto_sign_manifests.py # --- docs/ aes.html (dashboard) --- cat > docs/ aes.html <<'HTML' <!DOCTYPE html> <meta charset="utf-8"> <title>AES Evolution

Dashboard</title> <style> body{font:14px/1.5 system-ui,Segoe UI,Roboto,Arial;margin:2rem;max-width:1000px} h1{margin:.2rem 0 .8rem} .row{display:flex;gap:1rem;flex-wrap:wrap} .card{border:1px solid #ddd;border-radius:8px;padding:12px;flex:1;min-width:280px} table{border-collapse:collapse;width:100%} th,td{border:1px solid #eee;padding:6px 8px;text-align:left} pre{background:#f7f7f7;padding:8px;border-radius:6px;overflow:auto;max-height:280px} .ok{color:#1b7f3a;font-weight:600} .bad{color:#b00020;font-weight:600} .mono{font-family:ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono", "Courier New", monospace} </style> <h1>AES Evolution Dashboard</h1> <p> Status badge: <img alt="AES badge" src="https://img.shields.io/endpoint?url=https://

raw.githubusercontent.com/wizardaax/recursive-field-math-pro/main/docs/badge.json">   <h2>Audio</h2> <audio controls src="sequence.wav"></audio> download WAV   <h2>Metrics</h2> <table id="mtab"><tbody></tbody></table> (summary.json not found)   <h2>sequence.csv (preview)</h2> <pre id="csv">Loading…</pre>   <h2>History</h2> <table id="hist"><thead><tr><th>Timestamp (UTC)</th><th>Commit</th><th>Chosen</th><th>Version</th><th>Latency</th><th>order_k2</th><th>order_k3</th></tr></thead><tbody></tbody></table> (history_index.json not found yet) <script> function

```
cls(key,val){ if(key.startsWith('order_')) return (+val>=0.1)?'ok':'bad'; if(key==='latency_ms') return (+val<=600)?'ok':'bad'; return ''; }
fetch('metrics.json').then(r=>r.ok?r.json():{}).then(m=>{ const tb=document.querySelector('#mtab tbody'); tb.innerHTML=''; const keys=Object.keys(m).sort(); if(!keys.length){ tb.innerHTML='<tr><td>(metrics.json not found yet)</td></tr>'; return; }
keys.forEach(k=>{ const v=m[k]; const tr=document.createElement('tr'); tr.innerHTML=`<th>${k}</th><td class="${cls(k,Number(v))}">${v}</td>`; tb.appendChild(tr); }); });
fetch('summary.json').then(r=>r.ok?r.json():{}).then(s=>{ if(!Object.keys(s||{}).length) return; const txt = `chosen=${s.chosen||''} version=${s.version||''} commit=${(s.commit||'').slice(0,7)} ts=${s.timestamp||''}`;
document.querySelector('#summary').textCo
```

```
ntent = txt; }); fetch('sequence.csv')
.then(r=>r.ok?r.text():'').then(t=>{ const
lines=(t||'').trim().split(/\r?\n/).slice(0,60);
document.getElementById('csv').textContent
= lines.length?lines.join('\n')+'\n'+
(lines.length>=60?'...
(truncated)':''):'(sequence.csv not found yet)';
}); fetch('history/history_index.json')
.then(r=>r.ok?r.json():[]).then(idx=>{ const
tb=document.querySelector('#hist tbody'); if(!
Array.isArray(idx) || !idx.length){
document.getElementById('histEmpty').style.
display='block'; return; }
idx.slice().reverse().forEach(e=>{ const
m=e.metrics||{}; const
row=document.createElement('tr'); const
sha=(e.commit||'').slice(0,7);
 row.innerHTML=`<td class="mono">$
{e.timestamp||''}</td> <td class="mono">$
{sha}</td> <td>${e.chosen||''}</td> <td>$
{e.version||''}</td> <td class="$
{cls('latency_ms',m.latency_ms)}">$
```

{m.latency_ms??"}</td> <td class="$
{cls('order_k2',m.order_k2)}">$
{m.order_k2??"}</td> <td class="$
{cls('order_k3',m.order_k3)}">$
{m.order_k3??"}</td>`; tb.appendChild(row);
}); }); </script> HTML # --- docs/.nojekyll
--- echo "# Disable Jekyll" > docs/.nojekyll
--- examples/request_nine.json --- cat >
examples/request_nine.json <<'JSON' { "api":
"codex.sequence/nine", "capabilities": [],
"params": { "steps": 27, "niner": true } } JSON
--- prepend README badge (safe if
README.md exists) --- if [-f README.md];
then BADGE='[![AES](https://img.shields.io/
endpoint?url=https://
raw.githubusercontent.com/wizardaax/
recursive-field-math-pro/main/docs/
badge.json)](https://wizardaax.github.io/
recursive-field-math-pro/aes.html)' if ! grep
-Fq "$BADGE" README.md 2>/dev/null; then
tmp="$(mktemp)"; printf '%s\n\n' "$BADGE" >
"$tmp"; cat README.md >> "$tmp"; mv

"$tmp" README.md fi fi echo "✅ Wrote AES v1.1 files. Next:" echo "1) Commit & push (optional now): git add -A && git commit -m 'Add AES v1.1 stack' && git push" echo "2) Enable Pages → Source: main /docs" echo "3) Run workflow (Actions → Evolve Build & Deploy → Run)" echo "4) Open: https://wizardaax.github.io/recursive-field-math-pro/aes.html"
if you also want the simpler **docs index** page (fallback) that previews the CSV and links the WAV, here it is; save as docs/index.html (optional—your AES dashboard lives at docs/aes.html):
<!DOCTYPE html> <meta charset="utf-8"> <title>AES Evolution Artifacts</title> <style>body{font:14px/1.4 system-ui;margin:2rem} pre{background:#f4f4f4;padding:8px}</style> <h1>AES Evolution Artifacts</h1> Synced from <code>out/</code> by the Evolve workflow. <a

href="aes.html">Open the full AES Dashboard sequence.wav • sequence.csv <pre id="csv">Loading sequence.csv preview…</pre> <script> fetch('sequence.csv').then(r=>r.text()).then(t=>{ const lines=t.trim().split(/\r?\n/).slice(0,40); document.getElementById('csv').textContent = lines.join('\n') + (lines.length>=40?'\n… (truncated)':''); }).catch(()=>{document.getElementById('csv').textContent='sequence.csv not found.'}); </script>
want me to bundle your ternary CPU / viewer files into a similar one-shot script too? say the word and I'll drop that pack as well.

got you. here's **everything in copy-paste form**, ready to drop straight into wizardaax/

recursive-field-math-pro. no chatter—just files.
name: Evolve Build & Deploy on: push: branches: [main] workflow_dispatch: permissions: contents: write concurrency: group: evolve-${{ github.ref }} cancel-in-progress: true jobs: evolve: runs-on: ubuntu-latest steps: - uses: actions/checkout@v4 - name: Guard id: guard run: | set -euo pipefail SKIP="false" MSG="$(jq -r '.head_commit.message // ""' "$GITHUB_EVENT_PATH" || true)" ACTOR="${GITHUB_ACTOR:-}" if echo "$MSG" | grep -qi '\[skip aes\]'; then SKIP="true"; fi if ["$ACTOR" = "github-actions[bot]"]; then SKIP="true"; fi echo "skip=$SKIP" >> "$GITHUB_OUTPUT" - name: Setup Python if: steps.guard.outputs.skip != 'true' uses: actions/setup-python@v5 with: python-version: '3.x' - name: Auto-sign plugin manifests (optional) if: steps.guard.outputs.skip != 'true' run: |

python3 tools/auto_sign_manifests.py || true − name: Run Evolution if: steps.guard.outputs.skip != 'true' run: | python3 xova/evolve.py examples/request_nine.json || true ls −la out/ || true − name: Validate metrics (non−fatal) if: steps.guard.outputs.skip != 'true' id: validate run: | python3 tools/validate_metrics.py −− policy policies/default.json −−metrics out/metrics.json || true − name: Badge + history + sync docs if: steps.guard.outputs.skip != 'true' run: | set −euo pipefail mkdir −p docs docs/history python3 − <<'PY' import json, time, os, shutil, pathlib root = pathlib.Path(".") docs = root/"docs"; hist = docs/"history" policy = json.loads((root/"policies"/"default.json").read_text(encoding="utf−8")) if (root/"policies"/"default.json").exists() else {} metrics = json.loads((root/"out"/"metrics.json").read_text(encoding="utf−8")) if

```
(root/"out"/"metrics.json").exists() else {}
summary =
json.loads((root/"out"/"summary.json").read_t
ext(encoding="utf-8")) if
(root/"out"/"summary.json").exists() else {}
thr = policy.get("thresholds",{}); mx =
thr.get("max",{}) or {}; mn = thr.get("min",{}) or
{} def ge(k, d=0.0): try: return
float(metrics.get(k, float('-inf'))) >=
float(mn.get(k,d)) except: return False def
le(k, d=1e9): try: return float(metrics.get(k,
float('inf'))) <= float(mx.get(k,d)) except:
return False overall = le("latency_ms",600)
and ge("order_k2",0.10) and
ge("order_k3",0.05) badge =
{"schemaVersion":1,"label":"AES","message":"p
ass" if overall else "fail","color":"brightgreen"
if overall else "red"}
(docs/"badge.json").write_text(json.dumps(b
adge, separators=(",",":")), encoding="utf-8")
ts = time.strftime("%Y%m%dT%H%M%SZ",
time.gmtime()); dst = hist/ts
```

dst.mkdir(parents=True, exist_ok=True) for name in ("metrics.json","summary.json","sequence.csv", "sequence.wav"): src = root/"out"/name if src.exists(): shutil.copy2(src, dst/src.name) idx_p = hist/"history_index.json" index = [] if idx_p.exists(): try: index = json.loads(idx_p.read_text(encoding="utf-8")) except: index = [] entry = {"timestamp": ts, "commit": os.environ.get("GITHUB_SHA",""), "chosen": summary.get("chosen",""), "version": summary.get("version",""), "metrics": metrics} index.append(entry) idx_p.write_text(json.dumps(index, indent=2), encoding="utf-8") PY [-f docs/ aes.html] || cp docs/index.html docs/ aes.html || true cp -f out/metrics.json docs/ 2>/dev/null || true cp -f out/summary.json docs/ 2>/dev/null || true cp -f out/ sequence.csv docs/ 2>/dev/null || true cp -f out/sequence.wav docs/ 2>/dev/null || true git config user.name "github-actions[bot]" git

config user.email "41898282+github-actions[bot]@users.noreply.github.com" git add docs git commit -m "Sync AES artifacts, badge, and history [skip ci] [skip aes]" || echo "No changes to commit" git push - name: Upload out/ artifact if: steps.guard.outputs.skip != 'true' uses: actions/upload-artifact@v4 with: name: evolve-output path: out/ retention-days: 7 { "security": { "require_signatures": true }, "limits": { "time_s": 15 }, "selection": { "prefer": "highest_version" }, "thresholds": { "max": { "latency_ms": 600 }, "min": { "order_k2": 0.10, "order_k3": 0.05 } }, "post_chain": [{ "api": "codex.analysis/order", "plugin": "codex.order_meter" }], "grading": { "penalties": { "bad-signature": 100, "timeout": 50, "exception": 40, "bad-metrics": 30 } } } #!/usr/bin/env python3 import json, os, sys, time, hashlib from pathlib import Path ROOT = Path(__file__).resolve().parents[1] PLUGINS = ROOT / "plugins" OUTDIR = ROOT

```
/ "out" POLICY = ROOT / "policies" / "default.json" from registry import load_registry, choose_candidates from sandbox import run_plugin def load_policy(): with open(POLICY, "r", encoding="utf-8") as f: return json.load(f) def sha256_manifest_without_sig(man: dict) -> str: m2 = man.copy() m2.pop("signature", None) raw = json.dumps(m2, sort_keys=True, separators=(",",":")).encode() return hashlib.sha256(raw).hexdigest() def verify_manifest(man): want = man.get("signature","") if not want: return False got = sha256_manifest_without_sig(man) return got == want def passes_thresholds(metrics, thr): if not thr: return True mx = thr.get("max", {}) or {} mn = thr.get("min", {}) or {} for k, v in mx.items(): if k in metrics and isinstance(metrics[k], (int,float)) and metrics[k] > v: return False for k, v in mn.items(): if k in metrics and
```

```
isinstance(metrics[k], (int,float)) and metrics[k] < v: return False return True def rank_candidates(cands, prefer): if prefer == "highest_version": def vparts(x): s = x["manifest"].get("version","0.0.0") return tuple(int(p) if p.isdigit() else 0 for p in s.split(".")) return sorted(cands, key=vparts, reverse=True) return cands def _write_summaries(chosen_name, man, metrics, api): OUTDIR.mkdir(exist_ok=True) summary = { "chosen": chosen_name, "version": man.get("version", ""), "metrics": metrics, "timestamp": time.time(), "request_api": api, "commit": os.environ.get("GITHUB_SHA", "") } (OUTDIR / "summary.json").write_text(json.dumps(summary, indent=2), encoding="utf-8") (OUTDIR / "metrics.json").write_text(json.dumps(metrics, separators=(",",":")), encoding="utf-8") def evolve(request_path: str):
```

OUTDIR.mkdir(exist_ok=True) with open(request_path, "r", encoding="utf-8") as f: req = json.load(f) policy = load_policy() registry = load_registry(PLUGINS) api = req["api"] caps = req.get("capabilities", []) params = req.get("params", {}) cands = choose_candidates(registry, api, caps) cands = rank_candidates(cands, policy.get("selection", {}).get("prefer")) if not cands: print(json.dumps({"status": "failed", "errors": [f"No plugin matches api={api} caps={caps}"]}, indent=2)) return 1 errors = [] for cand in cands: man = cand["manifest"] if policy["security"].get("require_signatures", False): if not verify_manifest(man): errors.append({"plugin": cand["name"], "error": "bad-signature"}) continue time_limit_s = float(man.get("time_limit_s", policy["limits"]["time_s"])) t0 = time.time() try: res = run_plugin(cand, params, OUTDIR, time_limit_s=time_limit_s) except Exception as e: errors.append({"plugin": cand["name"],

"error": f"exception:{e}"}) continue latency_ms = (time.time() − t0) * 1000.0 metrics = res.get("metrics", {}) or {} metrics["latency_ms"] = latency_ms for link in policy.get("post_chain", []): dep = registry.get(link["plugin"]) if not dep: errors.append({"plugin": cand["name"], "warning": f"missing−post:{link['plugin']}"}) continue try: ar = run_plugin(dep, {"wav": str(OUTDIR / "sequence.wav"), "k_values": [2,3]}, OUTDIR, time_limit_s=time_limit_s) metrics.update(ar.get("metrics", {}) or {}) except Exception as e: errors.append({"plugin": cand["name"], "error": f"post−exception:{e}"}) if not passes_thresholds(metrics, policy.get("thresholds", {})): errors.append({"plugin": cand["name"], "error": "bad−metrics", "metrics": metrics}) continue _write_summaries(cand["name"], man, metrics, api) print(json.dumps({ "status": "ok", "chosen": cand["name"], "version":

man.get("version",""), "artifacts": res.get("artifacts", {}), "metrics": metrics, "tried": [x["name"] for x in cands] }, indent=2)) return 0 print(json.dumps({"status": "failed", "errors": errors}, indent=2)) return 1 if __name__ == "__main__": if len(sys.argv) < 2: print("Usage: python3 xova/evolve.py examples/request_nine.json") sys.exit(2) sys.exit(evolve(sys.argv[1])) #!/usr/bin/env python3 import json, sys, argparse, pathlib def load_json(p): p = pathlib.Path(p) if not p.exists(): return {} try: return json.loads(p.read_text(encoding="utf-8")) except Exception: return {} def main(): ap = argparse.ArgumentParser() ap.add_argument("--policy", required=True) ap.add_argument("--metrics", required=True) ap.add_argument("--json", action="store_true") args = ap.parse_args() policy = load_json(args.policy) metrics = load_json(args.metrics) thr = policy.get("thresholds", {}) mx =

thr.get("max", {}) if isinstance(thr.get("max", {}), dict) else {} mn = thr.get("min", {}) if isinstance(thr.get("min", {}), dict) else {} reasons, ok = [], True for k,v in mx.items(): if isinstance(metrics.get(k), (int,float)) and metrics[k] > v: ok = False; reasons.append(f"{k}>{v} (got {metrics[k]})") for k,v in mn.items(): if isinstance(metrics.get(k), (int,float)) and metrics[k] < v: ok = False; reasons.append(f"{k}<{v} (got {metrics[k]})") if args.json: print(json.dumps({"pass": ok, "reasons": reasons, "metrics": metrics}, separators=(",",":"))) else: print("AES metrics: " + ("PASS" if ok else "FAIL — " + "; ".join(reasons))) return 0 if __name__ == "__main__": sys.exit(main()) #!/usr/bin/env python3 import json, sys, hashlib, pathlib def sign(path: str, in_place: bool): p = pathlib.Path(path) man = json.loads(p.read_text(encoding="utf-8")) man.pop("signature", None) raw =

```
json.dumps(man, sort_keys=True, separators=(",",":")).encode() sig = hashlib.sha256(raw).hexdigest() if in_place: man["signature"] = sig p.write_text(json.dumps(man, indent=2), encoding="utf-8") print(f"Signed {p} -> signature={sig}") else: print(sig) if __name__ == "__main__": if len(sys.argv) < 2: print("Usage: python3 tools/sign_manifest.py <manifest.json> [--in-place]") sys.exit(2) sign(sys.argv[1], "--in-place" in sys.argv[2:]) #!/usr/bin/env python3 import json, hashlib, pathlib, sys PLUGINS = pathlib.Path("plugins") def sign_obj(obj): o2 = dict(obj) o2.pop("signature", None) raw = json.dumps(o2, sort_keys=True, separators=(",",":")).encode() return hashlib.sha256(raw).hexdigest() def main(): if not PLUGINS.exists(): print("No plugins/ directory; skipping.") return 0 changed = 0 for man_path in PLUGINS.rglob("manifest.json"):
```

```
data = json.loads(man_path.read_text(encoding="utf-8")) sig0 = data.get("signature") sig1 = sign_obj(data) if sig0 != sig1: data["signature"] = sig1 man_path.write_text(json.dumps(data, indent=2), encoding="utf-8") print(f"Signed/updated: {man_path}") changed += 1 print(f"Done. {changed} manifest(s) updated.") return 0 if __name__ == "__main__": sys.exit(main()) { "api": "codex.sequence/nine", "capabilities": [], "params": { "steps": 27, "niner": true } } <!DOCTYPE html> <meta charset="utf-8"> <title>AES Evolution Dashboard</title> <style> body{font:14px/1.5 system-ui,Segoe UI,Roboto,Arial;margin:2rem;max-width:1000px} h1{margin:.2rem 0 .8rem} .row{display:flex;gap:1rem;flex-wrap:wrap} .card{border:1px solid #ddd;border-radius:8px;padding:12px;flex:1;min-width:280px} table{border-
```

collapse:collapse;width:100%}
th,td{border:1px solid #eee;padding:6px
8px;text-align:left}
pre{background:#f7f7f7;padding:8px;border-
radius:6px;overflow:auto;max-height:280px}
.ok{color:#1b7f3a;font-weight:600}
.bad{color:#b00020;font-weight:600}
.mono{font-family:ui-monospace, SFMono-
Regular, Menlo, Monaco, Consolas,
"Liberation Mono", "Courier New",
monospace} </style> <h1>AES Evolution
Dashboard</h1> Status badge:
<img alt="AES badge" src="https://
img.shields.io/endpoint?url=https://
raw.githubusercontent.com/wizardaax/
recursive-field-math-pro/main/docs/
badge.json">   <div
class="card"> <h2>Audio</h2> <audio
controls src="sequence.wav"></audio>
<a href="sequence.wav"
download>download WAV
 <h2>Metrics</h2> <table

id="mtab"><tbody></tbody></table> (summary.json not found)   <h2>sequence.csv (preview)</h2> <pre id="csv">Loading…</pre>   <h2>History</h2> <table id="hist"><thead><tr><th>Timestamp (UTC)</th><th>Commit</th><th>Chosen</th><th>Version</th><th>Latency</th><th>order_k2</th><th>order_k3</th></tr></thead><tbody></tbody></table> (history_index.json not found yet) <script> function cls(key,val){ if(key.startsWith('order_')) return (+val>=0.1)?'ok':'bad'; if(key==='latency_ms') return (+val<=600)?'ok':'bad'; return ''; } fetch('metrics.json').then(r=>r.ok?r.json():{}).then(m=>{ const tb=document.querySelector('#mtab tbody'); tb.innerHTML=''; const keys=Object.keys(m).sort(); if(!keys.length){

```
tb.innerHTML='<tr><td>(metrics.json not found yet)</td></tr>'; return; }
keys.forEach(k=>{ const v=m[k]; const tr=document.createElement('tr');
tr.innerHTML=`<th>${k}</th><td class="${cls(k,Number(v))}">${v}</td>`;
tb.appendChild(tr); }); });
fetch('summary.json').then(r=>r.ok?r.json():{}).then(s=>{ if(!Object.keys(s||{}).length) return; const txt = `chosen=${s.chosen||''} version=${s.version||''} commit=${(s.commit||'').slice(0,7)} ts=${s.timestamp||''}`;
document.querySelector('#summary').textContent = txt; });
fetch('sequence.csv').then(r=>r.ok?r.text():'').then(t=>{ const lines=(t||'').trim().split(/\r?\n/).slice(0,60);
document.getElementById('csv').textContent = lines.length?lines.join('\n')+'\n'+(lines.length>=60?'... (truncated)':''):'(sequence.csv not found yet)';
});
fetch('history/history_index.json')
```

.then(r=>r.ok?r.json():[]).then(idx=>{ const tb=document.querySelector('#hist tbody'); if(!Array.isArray(idx) || !idx.length){ document.getElementById('histEmpty').style.display='block'; return; } idx.slice().reverse().forEach(e=>{ const m=e.metrics||{}; const row=document.createElement('tr'); const sha=(e.commit||'').slice(0,7); row.innerHTML=`<td class="mono">${e.timestamp||''}</td> <td class="mono">${sha}</td> <td>${e.chosen||''}</td> <td>${e.version||''}</td> <td class="${cls('latency_ms',m.latency_ms)}">${m.latency_ms??''}</td> <td class="${cls('order_k2',m.order_k2)}">${m.order_k2??''}</td> <td class="${cls('order_k3',m.order_k3)}">${m.order_k3??''}</td>`; tb.appendChild(row); }); }); </script> # Disable Jekyll [![AES](https://img.shields.io/endpoint?url=https://raw.githubusercontent.com/wizardaax/

recursive-field-math-pro/main/docs/badge.json)](https://wizardaax.github.io/recursive-field-math-pro/aes.html) <!-- paste the line above at the very top of README.md (leave other content unchanged) -->

one-liner to stage them (optional)

git add .github/workflows/evolve.yml policies/default.json xova/evolve.py tools/validate_metrics.py tools/sign_manifest.py tools/auto_sign_manifests.py docs/aes.html docs/.nojekyll README.badge.patch.md examples/request_nine.json git commit -m "AES v1.1: workflow, runner, summaries, validator, dashboard, badge scaffolding" git push

that's it.

totally. here's a **clean, consolidated, copy-paste** script that recreates **everything we built** (policy, evolve runner with summaries,

workflow w/ guard+badge+history+docs sync, validator, signing tools, dashboard page, example request). save as save_all_aes.sh in your repo root and run it. #!/usr/bin/env bash # save_all_aes.sh — one-shot writer for AES v1.1 files + dashboard # Usage: # bash save_all_aes.sh # Then (optional): # git add -A && git commit -m "Add AES v1.1 stack" && git push set -euo pipefail # --- layout --- mkdir -p policies xova .github/workflows tools docs docs/history examples # --- policies/default.json --- cat > policies/default.json <<'JSON' { "security": { "require_signatures": true }, "limits": { "time_s": 15 }, "selection": { "prefer": "highest_version" }, "thresholds": { "max": { "latency_ms": 600 }, "min": { "order_k2": 0.10, "order_k3": 0.05 } }, "post_chain": [{ "api": "codex.analysis/order", "plugin": "codex.order_meter" }], "grading": { "penalties": { "bad-signature": 100, "timeout": 50, "exception": 40, "bad-metrics": 30 } } }

JSON # --- xova/evolve.py --- cat > xova/evolve.py <<'PY' #!/usr/bin/env python3 import json, os, sys, time, hashlib from pathlib import Path ROOT = Path(__file__).resolve().parents[1] PLUGINS = ROOT / "plugins" OUTDIR = ROOT / "out" POLICY = ROOT / "policies" / "default.json" from registry import load_registry, choose_candidates from sandbox import run_plugin def load_policy(): with open(POLICY, "r", encoding="utf-8") as f: return json.load(f) def sha256_manifest_without_sig(man: dict) -> str: m2 = man.copy() m2.pop("signature", None) raw = json.dumps(m2, sort_keys=True, separators=(",",":")).encode() return hashlib.sha256(raw).hexdigest() def verify_manifest(man): want = man.get("signature","") if not want: return False got = sha256_manifest_without_sig(man) return got == want def passes_thresholds(metrics,

```
thr): if not thr: return True mx = thr.get("max", {}) or {} mn = thr.get("min", {}) or {} for k, v in mx.items(): if k in metrics and isinstance(metrics[k], (int,float)) and metrics[k] > v: return False for k, v in mn.items(): if k in metrics and isinstance(metrics[k], (int,float)) and metrics[k] < v: return False return True def rank_candidates(cands, prefer): if prefer == "highest_version": def vparts(x): s = x["manifest"].get("version","0.0.0") return tuple(int(p) if p.isdigit() else 0 for p in s.split(".")) return sorted(cands, key=vparts, reverse=True) return cands def _write_summaries(chosen_name, man, metrics, api): OUTDIR.mkdir(exist_ok=True) summary = { "chosen": chosen_name, "version": man.get("version", ""), "metrics": metrics, "timestamp": time.time(), "request_api": api, "commit": os.environ.get("GITHUB_SHA", "") } (OUTDIR /
```

"summary.json").write_text(json.dumps(summary, indent=2), encoding="utf-8") (OUTDIR / "metrics.json").write_text(json.dumps(metrics, separators=(",",":")), encoding="utf-8") def evolve(request_path: str): OUTDIR.mkdir(exist_ok=True) with open(request_path, "r", encoding="utf-8") as f: req = json.load(f) policy = load_policy() registry = load_registry(PLUGINS) api = req["api"] caps = req.get("capabilities", []) params = req.get("params", {}) cands = choose_candidates(registry, api, caps) cands = rank_candidates(cands, policy.get("selection", {}).get("prefer")) if not cands: print(json.dumps({"status": "failed", "errors": [f"No plugin matches api={api} caps={caps}"]}, indent=2)) return 1 errors = [] for cand in cands: man = cand["manifest"] if policy["security"].get("require_signatures", False): if not verify_manifest(man): errors.append({"plugin": cand["name"], "error":

"bad-signature"}) continue time_limit_s = float(man.get("time_limit_s", policy["limits"]["time_s"])) t0 = time.time() try: res = run_plugin(cand, params, OUTDIR, time_limit_s=time_limit_s) except Exception as e: errors.append({"plugin": cand["name"], "error": f"exception:{e}"}) continue latency_ms = (time.time() - t0) * 1000.0 metrics = res.get("metrics", {}) or {} metrics["latency_ms"] = latency_ms for link in policy.get("post_chain", []): dep = registry.get(link["plugin"]) if not dep: errors.append({"plugin": cand["name"], "warning": f"missing-post:{link['plugin']}"}) continue try: ar = run_plugin(dep, {"wav": str(OUTDIR / "sequence.wav"), "k_values": [2,3]}, OUTDIR, time_limit_s=time_limit_s) metrics.update(ar.get("metrics", {}) or {}) except Exception as e: errors.append({"plugin": cand["name"], "error": f"post-exception:{e}"}) if not passes_thresholds(metrics,

```
policy.get("thresholds", {})):
errors.append({"plugin": cand["name"], "error":
"bad-metrics", "metrics": metrics}) continue
_write_summaries(cand["name"], man,
metrics, api) print(json.dumps({ "status": "ok",
"chosen": cand["name"], "version":
man.get("version",""), "artifacts":
res.get("artifacts", {}), "metrics": metrics,
"tried": [x["name"] for x in cands] }, indent=2))
return 0 print(json.dumps({"status": "failed",
"errors": errors}, indent=2)) return 1 if
__name__ == "__main__": if len(sys.argv) <
2: print("Usage: python3 xova/evolve.py
examples/request_nine.json") sys.exit(2)
sys.exit(evolve(sys.argv[1])) PY chmod +x
xova/evolve.py # --- .github/workflows/
evolve.yml (guard + badge + history + docs
sync) --- cat > .github/workflows/
evolve.yml <<'YML' name: Evolve Build &
Deploy on: push: branches: [ main ]
workflow_dispatch: permissions: contents:
write concurrency: group: evolve-${{
```

github.ref }} cancel-in-progress: true jobs: evolve: runs-on: ubuntu-latest steps: - uses: actions/checkout@v4 - name: Guard id: guard run: | set -euo pipefail SKIP="false" MSG="$(jq -r '.head_commit.message // ""' "$GITHUB_EVENT_PATH" 2>/dev/null || true)" ACTOR="${GITHUB_ACTOR:-}" if echo "$MSG" | grep -qi '\[skip aes\]'; then SKIP="true"; fi if ["$ACTOR" = "github-actions[bot]"]; then SKIP="true"; fi echo "skip=$SKIP" >> "$GITHUB_OUTPUT" - name: Setup Python if: steps.guard.outputs.skip != 'true' uses: actions/setup-python@v5 with: python-version: '3.x' - name: Auto-sign plugin manifests (optional) if: steps.guard.outputs.skip != 'true' run: | python3 tools/auto_sign_manifests.py || true - name: Run Evolution if: steps.guard.outputs.skip != 'true' run: | python3 xova/evolve.py examples/request_nine.json || true ls -la out/ || true -

name: Validate metrics (non-fatal) if: steps.guard.outputs.skip != 'true' id: validate run: | python3 tools/validate_metrics.py --policy policies/default.json --metrics out/metrics.json || true - name: Badge + history + sync docs if: steps.guard.outputs.skip != 'true' run: | set -euo pipefail mkdir -p docs docs/history python3 - <<'PY' import json, time, os, shutil, pathlib root = pathlib.Path("."); docs = root/"docs"; hist = docs/"history" policy = json.loads((root/"policies"/"default.json").read_text(encoding="utf-8")) if (root/"policies"/"default.json").exists() else {} metrics = json.loads((root/"out"/"metrics.json").read_text(encoding="utf-8")) if (root/"out"/"metrics.json").exists() else {} summary = json.loads((root/"out"/"summary.json").read_text(encoding="utf-8")) if (root/"out"/"summary.json").exists() else {}

```
thr = policy.get("thresholds",{}); mx = thr.get("max",{}) or {}; mn = thr.get("min",{}) or {} def ge(k, d=0.0): try: return float(metrics.get(k, float('-inf'))) >= float(mn.get(k,d)) except: return False def le(k, d=1e9): try: return float(metrics.get(k, float('inf'))) <= float(mx.get(k,d)) except: return False overall = le("latency_ms",600) and ge("order_k2",0.10) and ge("order_k3",0.05) badge = {"schemaVersion":1,"label":"AES","message":"pass" if overall else "fail","color":"brightgreen" if overall else "red"} (docs/"badge.json").write_text(json.dumps(badge, separators=(",",":")), encoding="utf-8") ts = time.strftime("%Y%m%dT%H%M%SZ", time.gmtime()); dst = hist/ts dst.mkdir(parents=True, exist_ok=True) for name in ("metrics.json","summary.json","sequence.csv", "sequence.wav"): src = root/"out"/name if src.exists(): shutil.copy2(src, dst/src.name)
```

idx_p = hist/"history_index.json" index = [] if idx_p.exists(): try: index = json.loads(idx_p.read_text(encoding="utf-8")) except: index = [] entry = { "timestamp": ts, "commit": os.environ.get("GITHUB_SHA",""), "chosen": summary.get("chosen",""), "version": summary.get("version",""), "metrics": metrics } index.append(entry) idx_p.write_text(json.dumps(index, indent=2), encoding="utf-8") PY [-f docs/aes.html] || cp docs/index.html docs/aes.html || true cp -f out/metrics.json docs/ 2>/dev/null || true cp -f out/summary.json docs/ 2>/dev/null || true cp -f out/sequence.csv docs/ 2>/dev/null || true cp -f out/sequence.wav docs/ 2>/dev/null || true git config user.name "github-actions[bot]" git config user.email "41898282+github-actions[bot]@users.noreply.github.com" git add docs git commit -m "Sync AES artifacts, badge, and history [skip ci] [skip aes]" || echo "No changes to commit" git push - name:

Upload out/ artifact if: steps.guard.outputs.skip != 'true' uses: actions/upload-artifact@v4 with: name: evolve-output path: out/ retention-days: 7 YML # --- tools/validate_metrics.py --- cat > tools/validate_metrics.py <<'PY' #!/usr/bin/env python3 import json, sys, argparse, pathlib def load_json(p): p = pathlib.Path(p) if not p.exists(): return {} try: return json.loads(p.read_text(encoding="utf-8")) except Exception: return {} def main(): ap = argparse.ArgumentParser() ap.add_argument("--policy", required=True) ap.add_argument("--metrics", required=True) ap.add_argument("--json", action="store_true") args = ap.parse_args() policy = load_json(args.policy) metrics = load_json(args.metrics) thr = policy.get("thresholds", {}) mx = thr.get("max", {}) if isinstance(thr.get("max", {}), dict) else {} mn = thr.get("min", {}) if isinstance(thr.get("min", {}), dict) else {}

reasons, ok = [], True for k,v in mx.items(): if isinstance(metrics.get(k), (int,float)) and metrics[k] > v: ok = False; reasons.append(f"{k}>{v} (got {metrics[k]})") for k,v in mn.items(): if isinstance(metrics.get(k), (int,float)) and metrics[k] < v: ok = False; reasons.append(f"{k}<{v} (got {metrics[k]})") if args.json: print(json.dumps({"pass": ok, "reasons": reasons, "metrics": metrics}, separators=(",",":"))) else: print("AES metrics: " + ("PASS" if ok else "FAIL — " + "; ".join(reasons))) return 0 if __name__ == "__main__": sys.exit(main()) PY chmod +x tools/validate_metrics.py # --- tools/sign_manifest.py --- cat > tools/sign_manifest.py <<'PY' #!/usr/bin/env python3 import json, sys, hashlib, pathlib def sign(path: str, in_place: bool): p = pathlib.Path(path) man = json.loads(p.read_text(encoding="utf-8")) man.pop("signature", None) raw =

```
json.dumps(man, sort_keys=True, separators=(",",":")).encode() sig = hashlib.sha256(raw).hexdigest() if in_place: man["signature"] = sig p.write_text(json.dumps(man, indent=2), encoding="utf-8") print(f"Signed {p} -> signature={sig}") else: print(sig) if __name__ == "__main__": if len(sys.argv) < 2: print("Usage: python3 tools/sign_manifest.py <manifest.json> [--in-place]") sys.exit(2) sign(sys.argv[1], "--in-place" in sys.argv[2:]) PY chmod +x tools/sign_manifest.py # --- tools/auto_sign_manifests.py --- cat > tools/auto_sign_manifests.py <<'PY' #!/usr/bin/env python3 import json, hashlib, pathlib, sys PLUGINS = pathlib.Path("plugins") def sign_obj(obj): o2 = dict(obj) o2.pop("signature", None) raw = json.dumps(o2, sort_keys=True, separators=(",",":")).encode() return hashlib.sha256(raw).hexdigest() def main():
```

if not PLUGINS.exists(): print("No plugins/ directory; skipping.") return 0 changed = 0 for man_path in PLUGINS.rglob("manifest.json"): data = json.loads(man_path.read_text(encoding="utf-8")) sig0 = data.get("signature") sig1 = sign_obj(data) if sig0 != sig1: data["signature"] = sig1 man_path.write_text(json.dumps(data, indent=2), encoding="utf-8") print(f"Signed/ updated: {man_path}") changed += 1 print(f"Done. {changed} manifest(s) updated.") return 0 if __name__ == "__main__": sys.exit(main()) PY chmod +x tools/auto_sign_manifests.py # --- docs/ aes.html (dashboard) --- cat > docs/ aes.html <<'HTML' <!DOCTYPE html> <meta charset="utf-8"> <title>AES Evolution Dashboard</title> <style> body{font:14px/ 1.5 system-ui,Segoe UI,Roboto,Arial;margin:2rem;max-width:1000px} h1{margin:.2rem 0 .8rem}

```
.row{display:flex;gap:1rem;flex-wrap:wrap}
.card{border:1px solid #ddd;border-radius:8px;padding:12px;flex:1;min-width:280px}
table{border-collapse:collapse;width:100%}
th,td{border:1px solid #eee;padding:6px 8px;text-align:left}
pre{background:#f7f7f7;padding:8px;border-radius:6px;overflow:auto;max-height:280px}
.ok{color:#1b7f3a;font-weight:600}
.bad{color:#b00020;font-weight:600}
.mono{font-family:ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono", "Courier New", monospace}
</style>
<h1>AES Evolution Dashboard</h1>
<p>
Status badge: 
<img alt="AES badge" src="https://img.shields.io/endpoint?url=https://raw.githubusercontent.com/wizardaax/recursive-field-math-pro/main/docs/badge.json">
</p>
<div class="row">
<div class="card">
<h2>Audio</h2>
<audio
```

controls src="sequence.wav"></audio>
<a href="sequence.wav"
download>download WAV
 <h2>Metrics</h2> <table
id="mtab"><tbody></tbody></table> <p
class="mono" id="summary">(summary.json
not found)   <div
class="card"> <h2>sequence.csv (preview)</
h2> <pre id="csv">Loading…</pre>
 <h2>History</h2> <table
id="hist"><thead><tr><th>Timestamp (UTC)</
th><th>Commit</th><th>Chosen</
th><th>Version</th><th>Latency</
th><th>order_k2</th><th>order_k3</th></
tr></thead><tbody></tbody></table> <p
id="histEmpty"
style="display:none">(history_index.json not
found yet) <script> function
cls(key,val){ if(key.startsWith('order_')) return
(+val>=0.1)?'ok':'bad'; if(key==='latency_ms')
return (+val<=600)?'ok':'bad'; return ''; }
fetch('metrics.json').then(r=>r.ok?r.json():

```
{}).then(m=>{ const
tb=document.querySelector('#mtab tbody');
tb.innerHTML=''; const
keys=Object.keys(m).sort(); if(!keys.length){
tb.innerHTML='<tr><td>(metrics.json not
found yet)</td></tr>'; return; }
keys.forEach(k=>{ const v=m[k]; const
tr=document.createElement('tr');
tr.innerHTML=`<th>${k}</th><td class="$
{cls(k,Number(v))}">${v}</td>`;
tb.appendChild(tr); }); });
fetch('summary.json').then(r=>r.ok?r.json():
{}).then(s=>{ if(!Object.keys(s||{}).length)
return; const txt = `chosen=${s.chosen||''}
version=${s.version||''} commit=$
{(s.commit||'').slice(0,7)} ts=${s.timestamp||''}
`;
document.querySelector('#summary').textCo
ntent = txt; }); fetch('sequence.csv')
.then(r=>r.ok?r.text():'').then(t=>{ const
lines=(t||'').trim().split(/\r?\n/).slice(0,60);
document.getElementById('csv').textContent
```

```
= lines.length?lines.join('\n')+'\n'+
(lines.length>=60?'...
(truncated)':''):'(sequence.csv not found yet)';
}); fetch('history/history_index.json')
.then(r=>r.ok?r.json():[]).then(idx=>{ const
tb=document.querySelector('#hist tbody'); if(!
Array.isArray(idx) || !idx.length){
document.getElementById('histEmpty').style.
display='block'; return; }
idx.slice().reverse().forEach(e=>{ const
m=e.metrics||{}; const
row=document.createElement('tr'); const
sha=(e.commit||'').slice(0,7);
row.innerHTML=`<td class="mono">$
{e.timestamp||''}</td> <td class="mono">$
{sha}</td> <td>${e.chosen||''}</td> <td>$
{e.version||''}</td> <td class="$
{cls('latency_ms',m.latency_ms)}">$
{m.latency_ms??''}</td> <td class="$
{cls('order_k2',m.order_k2)}">$
{m.order_k2??''}</td> <td class="$
{cls('order_k3',m.order_k3)}">$
```

{m.order_k3??"}</td>`; tb.appendChild(row); }); }); </script> HTML # --- docs/.nojekyll --- echo "# Disable Jekyll" > docs/.nojekyll # --- examples/request_nine.json --- cat > examples/request_nine.json <<'JSON' { "api": "codex.sequence/nine", "capabilities": [], "params": { "steps": 27, "niner": true } } JSON # --- optional lightweight docs/index.html linking to dashboard --- cat > docs/index.html <<'HTML' <!DOCTYPE html> <meta charset="utf-8"> <title>AES Evolution Artifacts</title> <style>body{font:14px/1.4 system-ui;margin:2rem} pre{background:#f4f4f4;padding:8px}</style> <h1>AES Evolution Artifacts</h1> Synced from <code>out/</code> by the Evolve workflow. Open the full AES Dashboard sequence.wav • sequence.csv <pre id="csv">Loading sequence.csv

preview…</pre> <script>
fetch('sequence.csv').then(r=>r.ok?
r.text():'').then(t=>{ const
lines=(t||'').trim().split(/\r?\n/).slice(0,40);
document.getElementById('csv').textContent
= lines.length?lines.join('\n')+
(lines.length>=40?'\n...
(truncated)':''):'(sequence.csv not found yet)';
}).catch(()=>{document.getElementById('csv')
.textContent='sequence.csv not found.'}); </
script> HTML echo "✅ Wrote AES v1.1 files."
echo "Next:" echo " 1) Commit & push: git
add -A && git commit -m 'Add AES v1.1
stack' && git push" echo " 2) Enable Pages →
Source: main /docs" echo " 3) Run workflow
(Actions → Evolve Build & Deploy → Run)"
echo " 4) Open: https://wizardaax.github.io/
recursive-field-math-pro/aes.html" ```欹0胖

Great question — and yes, **the math is there**,

but it's mostly embedded in the **plugin outputs and post-processing logic**, rather than being explicitly written out as formulas in the script.

Here's a breakdown of where the math lives and how it's used:

🔢 Where the math appears:

1. Thresholds in default.json

"thresholds": { "max": { "latency_ms": 600 }, "min": { "order_k2": 0.10, "order_k3": 0.05 } }

These are **numerical constraints**:

- latency_ms is a performance metric (likely time to process audio).
- order_k2 and order_k3 are likely **signal analysis metrics** — possibly derived from spectral or order-based transforms (e.g., autocorrelation, entropy, or custom plugin outputs).

2. Post-processing plugin: codex.order_meter

"post_chain": [{ "api": "codex.analysis/order",

"plugin": "codex.order_meter" }]
This plugin likely computes the **order metrics** (order_k2, order_k3) from the audio output. The math is inside that plugin — not shown here — but it's invoked and its results are validated.

3. Validator logic in validate_metrics.py

if metrics[k] > v: ... if metrics[k] < v: ...
This is basic **threshold comparison**, but it's enforcing mathematical constraints on plugin outputs.

4. Dashboard logic

if(key.startsWith('order_')) return (+val>=0.1)?'ok':'bad'; if(key==='latency_ms') return (+val<=600)?'ok':'bad';
This is **visual grading** based on math — turning metrics into pass/fail indicators.

🧠 What's missing (but implied):

- The actual **math behind order_k2 and order_k3** — you'd find that inside the

plugin codex.order_meter, which isn't included in this script.

- Any **signal processing**, **Fourier transforms**, or **statistical analysis** — again, likely inside the plugin or the sequence.wav generator.

✅ Conclusion:

Yes, the math is present in the form of:

- **Thresholds**
- **Validation logic**
- **Plugin outputs**
- **Dashboard grading**

But the **core mathematical computations** (e.g., how order_k2 is calculated) are abstracted away inside the plugin system. Would you like me to help **extract or reconstruct the math** behind those metrics (like order_k2)? Or maybe peek into the plugin logic if you have it?

got it — here's a **drop-in "order math"**

reference you can save alongside your stack. it makes the math explicit (Goertzel), computes order_k2 / order_k3 directly from a WAV, and outputs a small metrics JSON. #!/usr/bin/env python3 """ Order-math reference (stdlib only). - Reads a mono/stereo 16-bit PCM WAV. - Finds f0 via Goertzel scan. - Computes normalized power at subharmonics f0/2 and f0/3 (order_k2, order_k3). - Emits JSON (fs, f0_hz, f0_rel_power, order_k2, order_k3). Usage: python tools/order_math_reference.py --wav out/sequence.wav --json """ import wave, struct, math, json, argparse, pathlib def read_wav_mono(path, max_samples=None): p = pathlib.Path(path) with wave.open(str(p), 'rb') as w: nchan = w.getnchannels() sampwidth = w.getsampwidth() fs = w.getframerate() nframes = w.getnframes() frames = w.readframes(nframes) if sampwidth != 2:

raise ValueError("Only 16-bit PCM WAV supported") it = struct.iter_unpack('<h', frames) vals = [s[0] / 32768.0 for s in it] if nchan == 2: vals = [(vals[i] + vals[i+1]) * 0.5 for i in range(0, len(vals) - 1, 2)] if max_samples: vals = vals[:max_samples] return fs, vals def goertzel_power(samples, fs, freq): if freq <= 0.0 or freq >= fs * 0.5 or not samples: return 0.0 omega = 2.0 * math.pi * freq / fs coeff = 2.0 * math.cos(omega) s0 = s1 = s2 = 0.0 for x in samples: s0 = x + coeff * s1 - s2 s2 = s1 s1 = s0 # Goertzel bin power at `freq` return max(s1*s1 + s2*s2 - coeff*s1*s2, 0.0) def total_power(samples): return sum(x*x for x in samples) + 1e-12 def estimate_f0(samples, fs, fmin=80.0, fmax=2000.0, coarse_step=5.0, refine_span=30.0, refine_step=0.5): Ptot = total_power(samples) # coarse scan best_f, best_p = fmin, -1.0 f = fmin while f <= fmax: p = goertzel_power(samples, fs, f) if p >

best_p: best_p, best_f = p, f f += coarse_step # refine around best start = max(fmin, best_f − refine_span) end = min(fmax, best_f + refine_span) f = start while f <= end: p = goertzel_power(samples, fs, f) if p > best_p: best_p, best_f = p, f f += refine_step return best_f, best_p / Ptot def compute_order_metrics(wav_path, max_samples=65536, k_values=(2,3)): fs, data = read_wav_mono(wav_path, max_samples=max_samples) Ptot = total_power(data) f0, f0_rel = estimate_f0(data, fs) metrics = {"fs": fs, "f0_hz": f0, "f0_rel_power": f0_rel} for k in k_values: target = f0 / float(k) p = goertzel_power(data, fs, target) metrics[f"order_k{k}"] = p / Ptot return metrics def main(): ap = argparse.ArgumentParser() ap.add_argument("--wav", required=True) ap.add_argument("--json", action="store_true") ap.add_argument("--

max-samples", type=int, default=65536) args = ap.parse_args() m = compute_order_metrics(args.wav, max_samples=args.max_samples) if args.json: print(json.dumps(m, indent=2)) else: for k,v in m.items(): print(f"{k}: {v}") if __name__ == "__main__": main() # ORDER MATH (reference) ## Goertzel at a glance Given sample rate `fs`, target frequency `f`, and real samples `x[n]`, define: - `ω = 2π f / fs` - `c = 2 cos(ω)` - recurrence:
s[n] = x[n] + c * s[n-1] - s[n-2], with s[-1]=s[-2]=0
- power at `f`:
P(f) = s[N-1]^2 + s[N-2]^2 - c * s[N-1] * s[N-2]
Metrics 1. Estimate fundamental `f0` by scanning Goertzel power over a band (80–2000 Hz), coarse then refine. 2. Normalize by total signal power `Ptot = Σ x[n]^2`. 3. Subharmonic "order" powers: - `order_k2 = P(f0 / 2) / Ptot` - `order_k3 = P(f0 / 3) /

Ptot` ## Thresholds (from policy) – Pass if: `latency_ms ≤ 600`, `order_k2 ≥ 0.10`, `order_k3 ≥ 0.05`. This doc mirrors `tools/order_math_reference.py` so the math is explicit in-repo.

quick use

- save both files where shown.
- run: python tools/order_math_reference.py --wav out/sequence.wav --json
- you'll get: fs, f0_hz, f0_rel_power, order_k2, order_k3.

want me to also wire this into your evolve workflow as a fallback (in case codex.order_meter is absent)? I can drop a tiny patch that calls this script and writes out/metrics.json.