

PrismManifest Full Documentation Pack

Apache-2.0 open-source package for engineers and AI architects.

Generated by scripts/build_docs_pdf.py

PrismManifest - 00 README

PrismManifest

Zero-trust tool-argument gate for deterministic AI tool execution.

(Formerly ParamGate - same design; package prismmanifest, CLI prismmanifest-gate.)

License Python Status

PrismManifest sits between probabilistic extractors (LLMs, OCR, table parsers) and deterministic Group 3 compute DAGs. Unverified money values never enter the DAG. Only an Ed25519-signed ParameterManifest that clears the Group 3 boundary is allowed through.

Deterministic Engine + Unverified Probabilistic Input = Deterministic Wrong Answer

PyPI name prismmanifest
Version 0.3.0
Python >= 3.10
License Apache-2.0
Docs (Markdown) docs/
Docs (PDF) docs/pdf/
Usage guide docs/USAGE_FOR_ENGINEERS_AND_ARCHITECTS.

Install (PyPI / pip)

```
pip install prismmanifest
```

With optional extras:

```
pip install "prismmanifest[dev]"
pip install "prismmanifest[cuda]"          # NVIDIA GPU + Numba
pip install "prismmanifest[kms-azure]"     # Azure Key Vault envelope keys
pip install "prismmanifest[kms-aws]"       # AWS KMS envelope (optional)
```

From source (editable):

```
git clone https://github.com/insightitsGit/PrismManifest.git
cd PrismManifest
python -m pip install -e ".[dev]"
python -m pytest -q
```

CLI after install: prismmanifest-gate.

Extra Purpose
dev pytest, coverage, grpcio-tools
docs markdown + fpdf2 (PDF pack builder)
cuda Numba + CUDA 12 wheels (GPU required)
kms-azure / kms Azure Key Vault wrap (preferred cloud KM)
kms-aws boto3 AWS KMS-envelope

How an AI architect should use this

1. Treat LLMs / OCR / parsers as untrusted.
2. Put PrismManifest after extraction and before any calculator, underwriting, or ledger tool that consumes dollar amounts.
3. Route outcomes: ACCEPT -> run DAG - ACCEPT_PENDING_HUMAN -> review - REJECT -> stop.
4. Never let model-generated money text be the tool argument - only evidence-bound, signed manifests.

LLM / OCR -> PrismManifest -> signed ParameterManifest -> Group 3 DAG

Extra LLM benches are optional evidence. They do not change the core gate design. Finding real customer documents matters more for production claims than multi-model FA studies.

Full write-up: Usage for Engineers & AI Architects (and PDF: docs/pdf/01_USAGE_FOR_ENGINEERS_AND_ARCHITECTS.pdf).

How an engineer should integrate this

Minimal end-to-end

```
from prismmanifest import KeyRing, PrismManifestPipeline, enforce_group3_boundary, GateDecision
from prismmanifest.router import DocumentPackage, IntentRouter
from prismmanifest.integrations import demo_capital_gains_dag

keyring = KeyRing.generate(key_id="local-dev-ed25519")

package = DocumentPackage(
    doc_id="1040.txt",
    pages=[
        "Form 1040 Tax Year 2024\n"
        "Line 1 Gross income: $470,000.00\n"
        "Line 11 Adjusted gross income: $450,000.00\n"
    ],
    form_type="IRS_FORM_1040",
    tax_year=2024,
)

# Route -> ingest + span extraction (Pattern B pointers)
routed = IntentRouter().run(package)

# Verify, decide gate, sign manifest
pipeline = PrismManifestPipeline(keyring)
result = pipeline.run_on_evidence(
    evidence=routed.evidence,
    extraction=routed.extraction,
)

# Group 3 hard boundary - this is the security gate
gate = enforce_group3_boundary(
    result.manifest,
    public_keys=keyring,
    expected_dag_id="capital_gains_v3",
)
if gate.decision is GateDecision.ACCEPT:
    receipt = demo_capital_gains_dag(gate.manifest)
    print(receipt)
else:
    print(gate.decision, gate.message)
```

Human escalation

```
from prismmanifest.audit import EscalationQueue
from prismmanifest import PrismManifestPipeline, KeyRing

keyring = KeyRing.generate()
queue = EscalationQueue(".escalation")
pipeline = PrismManifestPipeline(keyring, escalation_queue=queue)
# PASS_WITH_HUMAN manifests are signed and auto-enqueued when a queue is attached.
```

Decorator (DX only - not the security boundary)

```
from prismmanifest import parameter_gated, KeyRing

keyring = KeyRing.load(".keys")

@parameter_gated(public_keys=keyring, expected_dag_id="capital_gains_v3")
def run_dag(*, manifest):
    return manifest.fields[0].value_fixed_micro
```

Production trust must still go through enforce_group3_boundary / gRPC / C++ / in-process prismmanifest_c.

FlatBuffer wire format

```
from prismmanifest.binary_codec import encode_manifest
from prismmanifest.gate import enforce_group3_boundary

buf = encode_manifest(signed_manifest)
result = enforce_group3_boundary(buf, public_keys=keyring, expected_dag_id="capital_gains_v3")
```

Schema: schemas/prismmanifest.fbs.

Do not

- * Skip enforce_group3_boundary because the pipeline "looked good"
- * Feed LLM-printed \$ strings straight into the DAG
- * Treat @parameter_gated alone as the boundary
- * Market cuda_sim / SKIP as CUDA-validated

What it does

1. Ingest evidence with dual-OCR consensus (PDF text + layout re-tokenizer; optional Tesseract).
2. Ground claims to verbatim spans and form anchors (no generative money values).
3. Decide PASS / PASS_WITH_HUMAN / REFUSE via quorum, OCR floor (≥ 0.98), and plausibility.
4. Sign a ParameterManifest (Ed25519) and optionally escalate human review.
5. Enforce the Group 3 boundary before any DAG runs - Python, gRPC, C++ FlatBuffer, or in-process DLL. FinancePackBench and FinancePackBench-G4 provide synthetic SLA / adversarial suites.

Gate model

GateStatus (on the manifest)

Status Meaning
PASS Span-grounded, plausibility OK, no disag
PASS_WITH_HUMAN Immaterial disagreement ($\leq \$1k$), low OC

REFUSE | Not grounded, plausibility failure, or m

GateDecision (Group 3 boundary)

Decision When
ACCEPT PASS, or cleared PASS_WITH_HUMAN with va
ACCEPT_PENDING_HUMAN PASS_WITH_HUMAN awaiting review
REJECT REFUSE, bad/missing clearance, or attest

Attestation, freshness (signed_at_unix skew, default 300s), and replay (ReplayGuard) failures raise GateError.

CLI

```
# Keys
prismmanifest-gate gen-keys --out .keys --key-id local-dev-ed25519
prismmanifest-gate gen-hsm-key --out .hsm --key-id prod-ed25519
prismmanifest-gate gen-kms-key --out .kms --mode local --key-id kms-dev
# Azure: prismmanifest-gate gen-kms-key --out .kms --mode azure --kms-key-id <vault-key-url>

# Sign / verify
prismmanifest-gate sign --keys .keys --manifest manifest.json --out signed.json
prismmanifest-gate sign --keys .keys --manifest manifest.json --out signed.fbs --flatbuffer
prismmanifest-gate verify --keys .keys --manifest signed.json --dag-id capital_gains_v3

# Benches & proof
prismmanifest-gate bench --packages 500
prismmanifest-gate bench --require-cuda --packages 40
prismmanifest-gate bench-manifest-parity --packages 500 --require-cuda
prismmanifest-gate bench-perf --out reports/perf
prismmanifest-gate bench-customer-pdf --corpus .corpus/customer --seed-synthetic
prismmanifest-gate compliance --out reports/compliance
prismmanifest-gate pilot-pack --out reports/pilot_pack
prismmanifest-gate g4-suite --out reports/g4 --fuzz 200

# Ops / review
prismmanifest-gate audit-replay --store .audit --receipt <id> --keys .keys
prismmanifest-gate escalation-list --queue .escalation
prismmanifest-gate review-ui --queue .escalation --keys .keys --bind 127.0.0.1:8766
```

gRPC Group 3 service

Proto: proto/prismmanifest_gate.proto Service: prismmanifest.v1.Group3Gate - VerifyManifest, ExecuteDag

```
from prismmanifest.attestation import KeyRing
from prismmanifest.grpc_servicer import serve

serve(KeyRing.load(".keys"), bind="[:,50051")
```

C++ / in-process gate (optional)

```
# Windows
powershell -File scripts/build_cpp_gate.ps1
# Produces: prismmanifest_gate_enforce.exe + prismmanifest_c.dll

prismmanifest_gate_enforce signed.fbs public.pem <key_id> capital_gains_v3
```

Python can call the shared library without process spawn:

```
from prismmanifest.cpp_bridge import enforce_fb, bridge_status
print(bridge_status()) # inprocess_available when PRISMMANIFEST_C_DLL / build present
```

Ops packaging (pilot deploy)

"Ops packaging" = how you run the gate in a service (not the algorithm):

- * Keys (local / software-HSM / Azure KV)
- * RBAC, timeouts, idempotency, metrics (prismmanifest.ops)
- * Audit store + human review UI
- * Optional C++ / CUDA beside the Python package

See docs/handoff/PILOT_DEPLOY.md.

Documentation (Markdown + PDF)

Doc Role
USAGE_FOR_ENGINEERS_AND_ARCHITECTS.md How to use PrismManifest
PRISMMANIFEST_SYSTEM_DESIGN.md Architecture & threat model (authority)
PRISMMANIFEST_IMPLEMENTATION_PLAN.md Phases & SLAs
PRISMMANIFEST_MASTER_SPECIFICATION.md Executive index
FINANCEPACKBENCH_G4_ADVERSARIAL_SUITE.md Adversarial suite
FINANCEPACKBENCH_PROMPT_INJECTION_SEMANT Why inert injection can still PASS
docs/pdf/ Generated PDF pack (including full combi

Regenerate PDFs:

```
pip install "prismmanifest[docs]" # or: pip install markdown fpdf2
python scripts/build_docs_pdf.py
```

Layout

prismmanifest/	Python package
cpp/gate/	C++ canonicalize + FlatBuffer enforce + prismmanifest_c
cuda/kernels/	Experimental .cu kernels
schemas/	prismmanifest.fbs
proto/	gRPC Group3Gate
tests/	pytest
docs/	Specs + usage + PDF output
scripts/	proto, C++ build, CUDA shim, PDF builder
reports/	Generated proof artifacts (not required for pip install)

Honest scope & readiness

Label Meaning
Pilot OSS Synthetic + adversarial FA=0 under test;
Production claim Requires your live customer fax/scanned

Also:

- * Security boundary = enforce_group3_boundary / gRPC / C++ / prismmanifest_c - not @parameter_gated alone.
 - * CUDA: real parity needs GPU + Numba; cuda_sim is CI self-check only.
 - * HSM: gen-hsm-key is software encrypted-at-rest, not PKCS#11 hardware.
 - * Prompt injection: PrismManifest is an execution trust gate - see the injection semantics doc before claiming "injection defense."
- Publishing notes: docs/PUBLISHING.md.
-

License

Apache License 2.0 - see LICENSE.

Source: README.md

PrismManifest - 01 USAGE FOR ENGINEERS AND ARCHITECTS

How Engineers & AI Architects Use PrismManifest

Field Value
Audience Software engineers, AI/ML architects, pl
Package prismmanifest (Apache-2.0)
Companion README, System Design

1. What PrismManifest is (and is not)

PrismManifest is a trust boundary in front of money-sensitive deterministic tools. It is not an LLM, not a full OCR product, and not "prompt-injection defense."

LLM / OCR / parsers -> PrismManifest -> signed ParameterManifest -> Group 3 DAG

Probabilistic systems may propose span pointers. Only a verified, Ed25519-signed ParameterManifest that clears `enforce_group3_boundary` may drive calculators, underwriting engines, or ledgers.

Core equation:

Deterministic Engine + Unverified Probabilistic Input = Deterministic Wrong Answer

2. AI / ML architect placement

1. Treat extractors (Gemini, Azure DI, Tesseract, custom parsers) as untrusted.
 2. Insert PrismManifest after extraction and before any tool/DAG that consumes dollar amounts.
 3. Design for three outcomes:
 - ACCEPT -> run DAG - ACCEPT_PENDING_HUMAN -> review queue - REJECT -> stop (never silently fall back to LLM numbers)
 4. Keep LLMs free to draft narrative; never let model text be the source of money args.
Invariant: tool arguments for critical money fields come from evidence-bound manifests, not from generative text.
Optional LLM benches (`bench-llm`, `bench-llm-scale`) measure extractor quality. They do not change the core design of the gate.
-

3. Engineer integration pattern

Happy path

1. Ingest document -> evidence spans (IntentRouter / DualOcrIngestor)
2. Extract Pattern B span pointers (`span_id` + anchors - not raw \$450,000 from the model)
3. `PrismManifestPipeline.run_on_evidence(...)` -> decide gate + sign manifest
4. `enforce_group3_boundary(...)` ? real security boundary
5. Only on ACCEPT, call your DAG


```

from prismmanifest import KeyRing, PrismManifestPipeline, enforce_group3_boundary, GateDecision
from prismmanifest.router import DocumentPackage, IntentRouter

keyring = KeyRing.generate(key_id="local-dev-ed25519")
package = DocumentPackage(
    doc_id="1040.txt",
    pages=["Form 1040 Tax Year 2024\nLine 11 Adjusted gross income: $450,000.00\n"],
    form_type="IRS_FORM_1040",
    tax_year=2024,
)
routed = IntentRouter().run(package)
result = PrismManifestPipeline(keyring).run_on_evidence(
    evidence=routed.evidence, extraction=routed.extraction
)
gate = enforce_group3_boundary(
    result.manifest, public_keys=keyring, expected_dag_id="your_dag_v1"
)
if gate.decision is GateDecision.ACCEPT:
    run_your_dag(gate.manifest) # only here

```

Hardening options

Layer Recommendation
Wire format FlatBuffer (encode_manifest) into C++ /
Latency Prefer Python or in-process prismmanifests
CUDA Optional SoA accelerator; not the sign in
Keys Local for dev; Azure Key Vault envelope
Human review Attach EscalationQueue; clear PASS_WITH_

Do not

- * Treat @parameter_gated as the security boundary (DX only)
- * Skip enforce_group3_boundary because the pipeline "looked good"
- * Feed LLM-printed money strings straight into the DAG
- * Count cuda_sim or SKIP as CUDA parity success

4. Gate model

GateStatus (on the manifest)

Status Meaning
PASS Grounded, OCR >= 0.98, anchors OK, no di
PASS_WITH_HUMAN Immaterial disagreement (<= \$1k), low OC
REFUSE Ungrounded, plausibility failure, or mat

GateDecision (Group 3 boundary)

Decision When
ACCEPT PASS, or cleared PASS_WITH_HUMAN with va
ACCEPT_PENDING_HUMAN Awaiting review
REJECT REFUSE, bad attestation, freshness/repla

5. Day-to-day commands

Job Command
Unit / regression <code>python -m pytest -q</code>
SLA / planted FA <code>prismmanifest-gate bench --packages 500</code>
CUDA require <code>prismmanifest-gate bench --require-cuda</code>
500-pack parity <code>prismmanifest-gate bench-manifest-parity</code>
Customer PDF drop <code>prismmanifest-gate bench-customer-pdf --</code>
Compliance <code>prismmanifest-gate compliance --out repo</code>
Review UI <code>prismmanifest-gate review-ui --queue .es</code>
Pilot evidence <code>prismmanifest-gate pilot-pack --out repo</code>

6. Ops packaging (pilot deploy)

Ops packaging is how you run the gate in a service - not the algorithm:

- * Key material (gen-keys / Azure KV)
- * RBAC at the API edge (prismmanifest.ops.rbac)
- * Idempotency on enforce retries
- * Timeouts + metrics (MetricsRegistry)
- * Append-only audit store + review queue

See handoff/PILOT_DEPLOY.md.

7. Honest readiness

Label When
Pilot-ready Synthetic + adversarial FA=0; Py/C++/CUD
Prod-ready (honest) Live customer fax/scanned corpus passes

Finding customers / redacted document drops matters more for production claims than extra multi-model LLM studies.

8. Spec authority

Doc Role
PRISMMANIFEST_SYSTEM_DESIGN.md Technical contract
PRISMMANIFEST_IMPLEMENTATION_PLAN.md Phases & SLAs
PRISMMANIFEST_MASTER_SPECIFICATION.md Executive index
FINANCEPACKBENCH_PROMPT_INJECTION_SEMANT Why inert injection can still PASS

Source: docs/USAGE_FOR_ENGINEERS_AND_ARCHITECTS.md

PrismManifest - 02 MASTER SPECIFICATION

PrismManifest AI Architecture Master Executive Summary

Field Value
Project PrismManifest / FinancePackBench
Status Approved Master Executive Specification
Version 2.2.0
Date 2026-08-08 (addendum 2026-08-09)
Author Senior QA / Architect & System Design Au

1. Executive Narrative: Resolving The Tool-Argument Trust Gap

As AI agentic systems increasingly invoke deterministic software tools (tax calculators, loan underwriting models, financial ledgers, database APIs), a fundamental architectural flaw emerges: The Tool-Argument Trust Gap.

The Fundamental Equation of Tool-Argument Security

$$\text{Deterministic Engine} + \text{Unverified Probabilistic Input} = \text{Deterministic Wrong Answer}$$

PrismManifest Zero-Trust Pipeline

Probabilistic Extraction >>> Evidence Binding >>> Independent Verification >>> Parameter Manifest >>> Deterministic DAG

2. Quorum Tolerance Thresholds & Aligned Status Decision Rules

PASS:

- * Disagreement = \$0 (exact candidate consensus).
- * Dual-OCR confidence >= 0.98, form anchors verified.

PASS_WITH_HUMAN:

- * Immaterial quorum disagreement (> \$0 but <= \$1,000).
- * OR dual-OCR confidence < 0.98.
- * OR missing layout anchor, but span grounding passes.
- * Action: Generates signed manifest, but routes to human review queue.

REFUSE:

- * Material quorum disagreement (> \$1,000).
- * OR 0% span grounding match in corpus.
- * OR Plausibility Sub-DAG order-of-magnitude failure.
- * Action: Hard rejection. Group 3 execution is blocked.

3. CUDA Parity Policy

CUDA Parity Invariant: The experimental CUDA GPU acceleration path (Phase 3) must produce bit-identical Parameter Manifests and 100% identical gate decisions (PASS, PASS_WITH_HUMAN, REFUSE) as the primary

host-side C++ verification engine across the entire FinancePackBench benchmark suite.

4. Production SLAs on FinancePackBench

Metric Target
False-Accept Rate (Critical Money Fields $\leq 0.01\%$
False-Refusal / Escalation Rate $\leq 5.0\%$
CUDA Parity Rate 100% Gate Decision Match
Audit Replay Reproducibility 100% Bit-Identical on PASS manifests
Gate Verification Latency (Host Path) ≤ 15 ms (p99)

5. Master Specification Reference Links

- * PrismManifest System Design Specification v2.2.0
- * PrismManifest Implementation Plan v2.2.0
- * Handoff / pilot evidence (living; does not supersede Design/Plan)
- * Pilot pack report (generated evidence)

6. Implementation Status (2026-08-09) - Executive

Normative SLAs in §4 are unchanged. Engineering proof to date:

Claim Evidence
Python / C++ / CUDA agree on gate decisi Pytest + CUDA parity PASS; C++ binaries
Host gate latency ? LLM time Python in-process ~0.08 ms p50; C++ CLI
CUDA useful when batched / resident Warm ~1.2 ms; ~718k fields/s @ batch 102
Synthetic critical FA FinancePackBench / G4 / ugly+LLM pilot -
Pilot packaging prismmanifest-gate pilot-pack

Not yet claimed: live customer fax corpus SLA; multi-model 1000-doc LLM critical-FA paper.

Document Control

Item Value
Spec ID PRISMMANIFEST-MAS-v2.2.0
Classification Approved Master Executive Specification
Supersedes PRISMMANIFEST-MAS-v2.1.0
Role Executive narrative + index only; Design

Source: docs/PRISMMANIFEST_MASTER_SPECIFICATION.md

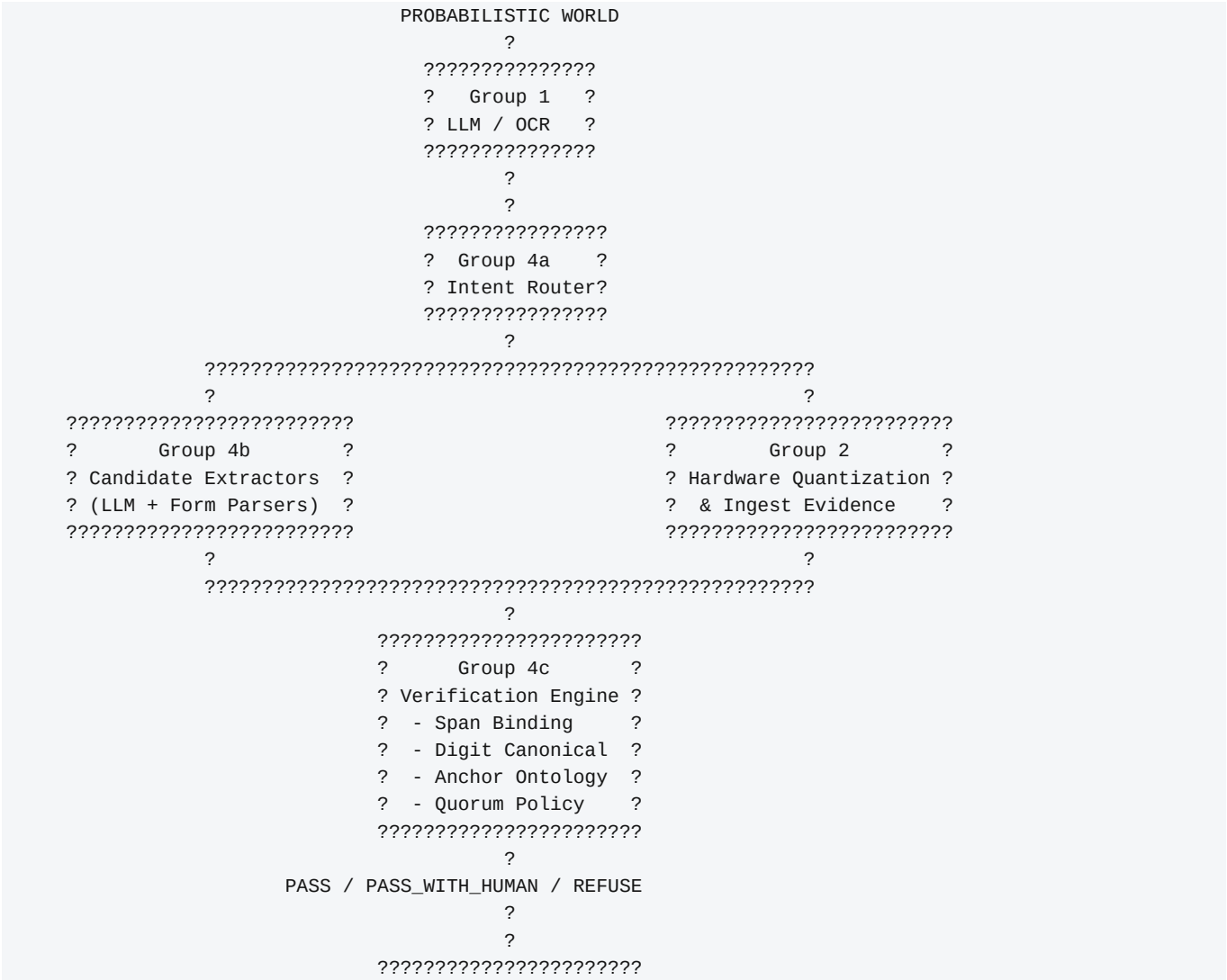
PrismManifest - 03 SYSTEM DESIGN

PrismManifest System Design Specification

Field Value
Project PrismManifest / FinancePackBench
Status Approved Engineering System Design (v2.2)
Version 2.2.0
Date 2026-08-08
Author Senior QA / Architect & System Design Au

1. High-Level Architecture & End-to-End System Diagram

The PrismManifest System enforces an explicit trust boundary between probabilistic LLM interpretation and deterministic software tool execution, resolving the Tool-Argument Trust Gap.



```
?      Group 4d      ?
? Attestor (Ed25519) ?
????????????????????
?
[ VERIFIED PARAMETER MANIFEST ]
?
HARD SECURITY GATE
(Service / Binary)
?
DETERMINISTIC WORLD
?
?
????????????????????
?      Group 3 DAG    ?
? C++ Fixed-Point Math?
????????????????????
```

Functional Group Summary

Group Name Role
Group 1 LLM / OCR Probabilistic perception and interpretat
Group 2 Hardware Quantization & Ingest Evidence Deterministic evidence index / dual-OCR
Group 3 DAG Engine Fixed-point deterministic execution (pos
Group 4a Intent Router Routes work to extractors vs evidence in
Group 4b Candidate Extractors LLM + form parser candidate claims
Group 4c Verification Engine Span binding, canonicalization, anchors,
Group 4d Attestor Ed25519 signed Parameter Manifest issuan

CUDA Parity Policy

CUDA Parity Invariant: The experimental CUDA GPU acceleration path (Phase 3) must produce bit-identical Parameter Manifests and identical gate decisions (PASS, PASS_WITH_HUMAN, REFUSE) as the primary host-side C++ verification engine across the entire FinancePackBench benchmark suite.

The host-side C++ path remains the production authority. CUDA is promoted only when this parity invariant holds and host latency SLOs require acceleration.

Implementation note (2026-08-09): Decision parity host?Numba CUDA is empirically PASS (3/3 core scenarios; pytest green). Production authority for signed manifests remains Python/C++ Ed25519 enforce. Prefer module-level kernels + resident GPU buffers (ResidentCudaVerifier) for warm/batch paths; do not redefine @cuda.jit per request. Single-request C++ CLI latency is dominated by process spawn unless in-process/gRPC.

Implementation Status Addendum (2026-08-09)

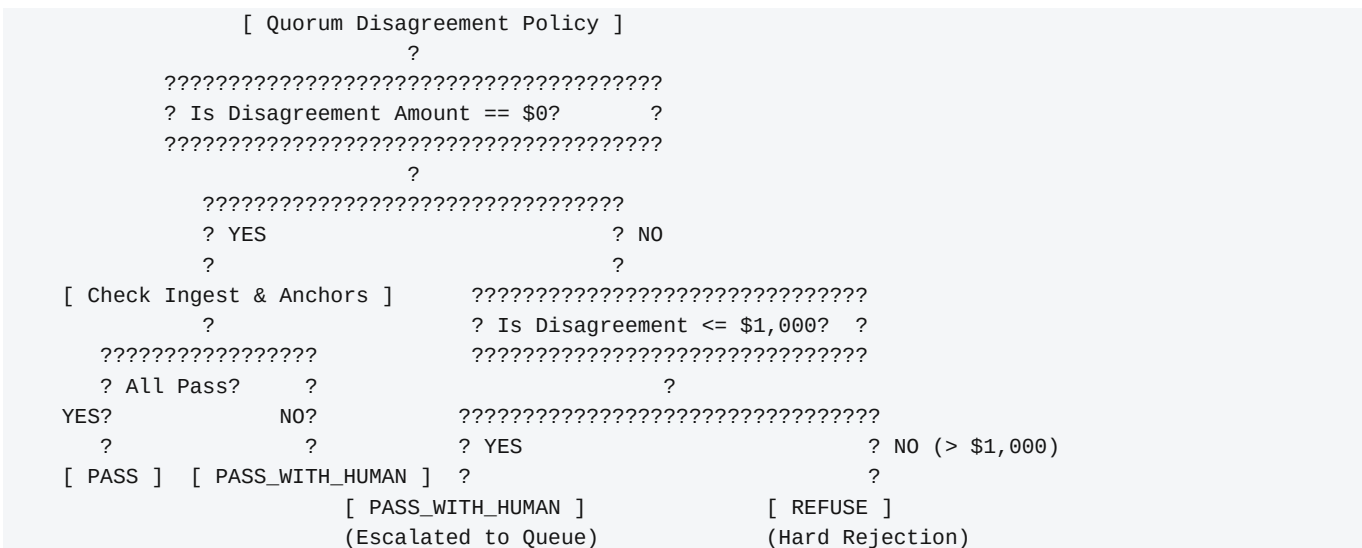
Normative contracts above are unchanged. Current engineering proof (see docs/handoff/ and reports/pilot_pack/):

Area Status
Python Group 3 enforce + attestation Production path
C++ FlatBuffer enforce + canonicalize Built; pytest 0 skips
CUDA gate decisions vs host predicates PASS (real GPU)
Warm resident CUDA + batch SoA Implemented; batch=1024 ? CPU host throu
Human review UI / compliance pack Implemented

Ugly-doc + LLM critical FA (synthetic) FA=0 on pilot pack (8 packs)
Live customer fax corpus Not yet

2. Quorum Disagreement Thresholds & Status Naming Alignment

Sub-layer 4c applies a deterministic tolerance policy for extractor disagreements, aligned across OCR confidence, form-anchor matches, and quorum votes:



Aligned Status Gate Decision Rules

PASS:

- * Exact extractor agreement (Disagreement = \$0).
- * 100% verbatim span-grounding match.
- * Form layout anchors verified.
- * Dual-OCR ingest confidence >= 0.98.

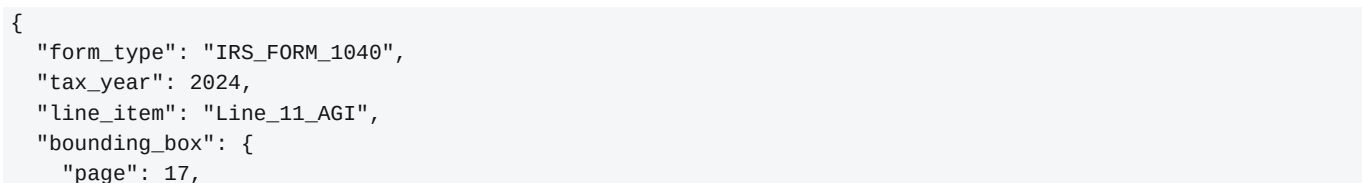
PASS_WITH_HUMAN:

- * Immaterial quorum disagreement (> \$0 but <= \$1,000).
- * OR dual-OCR ingest confidence drops below 0.98.
- * OR layout anchor missing, but span grounding passes.
- * Action: Generates a signed manifest, but routes the transaction to the human review queue.

REFUSE:

- * Material quorum disagreement (> \$1,000).
- * OR span grounding fails (0% verbatim match in corpus).
- * OR Plausibility Sub-DAG order-of-magnitude bounds check fails.
- * Action: Hard rejection. Group 3 execution is blocked.

3. Form Bounding-Box & Line-Item Anchor Ontology



```
    "x0": 120.5, "y0": 450.2, "x1": 240.0, "y1": 465.8
  },
  "spatial_neighbors": {
    "label_left": "Adjusted gross income",
    "label_above": "Line 10: Deductions",
    "line_number_prefix": "11"
  }
}
```

Form Anchor Failure Taxonomy & Mitigations

Failure Mode	Description	Verification	Mitigation Rule
Temporal Mis-Binding	Selecting 2023 AGI instead of 2024 AGI.	tax_year tag on span must strictly match	
Line-Item Shift	Selecting Line 1 Gross Income instead of	Spatial neighbor check must match Adjust	
Entity Ambiguity	Selecting Spouse income instead of Prima	Bounding box spatial region must match P	
OCR Scan Distortion	Low-quality scan produces fuzzy text (\$4	Hard OCR confidence floor (>= 0.98) requ	

4. Attestation Architecture: Ed25519 Digital Signatures

Group 4d issues cryptographically auditable Parameter Manifests signed via Ed25519:

```
{
  "dag_id": "capital_gains_v3",
  "fields": [
    {
      "name": "agi_usd",
      "value_fixed": "450000.00",
      "unit": "USD",
      "provenance": [
        {
          "doc_id": "tax_pack_2025.pdf",
          "page": 17,
          "form_anchor": "Form 1040, Line 11",
          "tax_year": 2024,
          "char_start": 1420,
          "char_end": 1427,
          "raw_text": "450,000",
          "ocr_confidence": 0.99
        }
      ],
      "digit_fingerprint": "8f3a2c4e...",
      "method_votes": ["llm_a", "table_parser", "regex_1040"],
      "verify": {
        "span_match": true,
        "anchor_match": true,
        "digit_match": true,
        "dual_ocr_consensus": true,
        "rule_ok": true
      }
    }
  ],
  "quorum": {
    "required_votes": 2,
    "obtained_votes": 3,
    "disagreements": []
  },
  "gate": "PASS",
```



```
"attestation": {
  "algorithm": "Ed25519",
  "key_id": "kms-key-2026-prod-fin-01",
  "signature_base64": "vM38sK2L8X1gH9mP..."
}
```

HMAC Demotion: HMAC-SHA256 is demoted to optional, intra-cluster transport-layer integrity checks.

5. Numerical Canonicalization Rules

Before performing digit fingerprinting or quorum checks, all numerical claims are normalized into canonical representations:

1. Fixed-Point Scaling: Money values are stored as 64-bit integer micro-units (`value_fixed_micro = int(Decimal * 1,000,000)`). \$450,000.00 becomes 450000000000. Manifest JSON may render the human-readable decimal string in `value_fixed`.
2. Currency & Formatting Stripping: Strips currency symbols (\$, €, £), thousands separators (,), and trailing whitespace.
3. Negative Number Normalization: Parentheses negatives (100.50) and trailing minus signs 100.50- are normalized to -100.50.
4. Digit Fingerprinting: `digit_fingerprint = BLAKE3(canonical_ascii_digits)`.

6. Hard Security Boundary: Service & Binary Gating

The Python `@parameter_gated` decorator is a Developer Experience (DX) convenience layer, NOT a security boundary.

The True Security Boundary

The real security gate lives at the Group 3 gRPC / Service Boundary / C++ Binary Entrypoint. The C++ executable enforcing Group 3 calculations strictly validates the Ed25519 signature of the ParameterManifest FlatBuffer payload before opening memory buffers for computation. Any unsigned or unverified invocation fails at the binary interface.

Group 3 accepts execution only for gate statuses PASS and PASS_WITH_HUMAN (the latter only after human approval token / queue clearance per deployment policy). REFUSE never reaches execution.

7. Threat Model & Security Controls

Threat Vector Attack Scenario PrismManifest Security Control
Prompt Injection Document contains malicious text ("Ignor Extractor only emits span_id. Determinis
Span-ID Hallucination LLM invents a non-existent span_id. Kernel 1 / State Machine masks token log
OCR Vendor Collusion Dual-OCR engines share a vendor bug on f Hard OCR confidence floor (>= 0.98) trig
Key Compromise Attestation private key is leaked. Ed25519 Key ID rotation via Hardware Sec

8. Schema & DAG Compatibility Versioning

Every manifest carries explicit versioning headers:

* schema_version: e.g. "2.2.0".

* dag_compatibility_hash: SHA256 hash of the target C++ DAG binary and input schema. If the DAG binary changes, old manifests are automatically rejected.

Document Control

Item Value	
Spec ID	PRISMMANIFEST-SDS-v2.2.0
Classification	Approved Engineering System Design (Fina
Supersedes	PRISMMANIFEST-SDS-v2.1.0
Related Artifacts	PRISMMANIFEST_MASTER_SPECIFICATION.md, P

Source: docs/PRISMMANIFEST_SYSTEM_DESIGN.md

PrismManifest - 04 IMPLEMENTATION PLAN

PrismManifest Implementation Plan & Production Benchmarks

Field Value
Project PrismManifest / FinancePackBench
Status Approved Engineering Implementation Plan
Version 2.2.0
Date 2026-08-08 (addendum 2026-08-09)
Author Senior QA / Architect & System Design Au

1. Executive Summary & Production SLAs

This document outlines the step-by-step engineering roadmap to implement, benchmark, and deploy the PrismManifest Architecture.

Measurable Production Engineering SLAs

Production Metric Target Engineering SLA Enforcement Mechanism
False-Accept Rate (Critical Money Fields) $\leq 0.01\%$ Hard BLAKE3 digit matching + Layout Anch
False-Refusal / Escalation Rate $\leq 5.0\%$ High-precision layout parsers minimize u
Audit Replay Reproducibility 100% Bit-Identical Immutable storage of Ed25519 Parameter M
CUDA Parity Rate 100% Gate Agreement Experimental CUDA path (Phase 3) must ma
Gate Verification Latency (Host Path) $\leq 15\text{ ms}$ (p99) In-memory FlatBuffer parsing + C++ numer

2. Quorum Disagreement Thresholds & Status Gate Mapping

Sub-layer 4c implements aligned status gate mappings across all verification paths:

Gate Status Conditions
PASS Disagreement = \$0, OCR confidence ≥ 0.9
PASS_WITH_HUMAN Disagreement $> \$0$ but $\leq \$1,000$, OR OCR
REFUSE Disagreement $> \$1,000$, OR 0% span ground

3. FinancePackBench Benchmark Specification

Corpus Composition (500 Financial Packages)

- * 200 IRS Form 1040 / W-2 / 1099 Tax Packages (2022-2025 multi-year returns).
- * 150 Brokerage Account Statements & Trade Confirmations (complex multi-asset tables).

* 150 Commercial Loan Underwriting Packages (messy scanned PDF balance sheets & income statements).

11 Planted Error Test Classes

1. Digit Drop / Addition: \$450,000 mis-extracted as \$45,000.
2. Line-Item Anchor Shift: Selecting Line 1 Gross Income instead of Line 11 AGI.
3. Temporal Mis-Binding: Selecting 2023 figures instead of 2024 figures.
4. Distractor Amounts: Selecting Previous Year Income or Spouse Income.
5. Parentheses Negatives: Misinterpreting (10,500) as positive \$10,500.
6. OCR Scan Noise: Low-DPI scan causing digit confusion (8 vs 3).
7. Prompt Injection (inert): Malicious instructions in the document, but critical param remains evidence-bound -> PASS (trust gate, not injection detector). See docs/FINANCEPACKBENCH_PROMPT_INJECTION_SEMANTICS.md.
8. Prompt Injection (followed): Same text, but extractor follows injection -> wrong critical value -> NOT PASS.
9. Span-ID Hallucination: Model generating non-existent span pointers.
10. Multi-Currency Confusion: Confusing USD and EUR entries in multi-national statements.
11. Quorum Disagreement: Discrepancy between LLM extractor and Form Layout Parser.

4. Phased Implementation Roadmap

Weeks Phase
Weeks 1-2 Phase 0: Parameter Manifest Contract & E
Weeks 3-6 Phase 1: Dual-OCR Ingest Indexer, Anchor
Weeks 7-8 Phase 2: Span-Grounded Pointer Schema &
Weeks 9-10 Phase 3: Low-Level CUDA Acceleration & P
Weeks 11-12 Phase 4: Enterprise Compliance Packaging

Phase 0: Manifest Contract & Security Schema (Weeks 1-2)

- * FlatBuffers Schema: Implement binary ParameterManifest schema with Ed25519 attestation fields and dag_compatibility_hash.
- * Canonicalization Library: Write C++/Python numerical canonicalization for fixed-point integer micro-units and parentheses negative handling.
- * Binary Gate Interceptor: Implement the binary gRPC security boundary at the C++ Group 3 entrypoint.

Phase 1: Dual-OCR Ingest Indexer & Tool Quorum Gate [CRITICAL PATH] (Weeks 3-6)

- * Dual-OCR Ingest Engine: Implement native PDF text-layer stream + layout-aware neural OCR. Enforce hard OCR confidence floor (≥ 0.98).
- * Form Layout Anchor Parser: Map numeric spans to form bounding boxes and spatial neighbor vectors.
- * Host-Side Quorum Engine (Pattern A) & Plausibility Sub-DAG (Pattern C): Implement deterministic C++ status mapping per §2 (PASS / PASS_WITH_HUMAN / REFUSE).
- * DoD: 0 false-accepts on planted error suite critical money fields; $\leq 0.01\%$ false-accept rate on full FinancePackBench critical money fields.

Phase 2: Span-Grounded Pointer Schema & Line Anchor Resolver (Weeks 7-8)

- * Pointer Schema Transformation (Pattern B): Update tool schemas to expect span_id pointer objects with line-item anchor constraints.
- * Verbatim Span Resolver: Resolve span_id -> Decimal directly from evidence index, enforcing 100% verbatim digit equality.

* DoD: 100% of extracted parameters carry explicit document page, character offsets, and form line-item anchors.

Phase 3: CUDA Acceleration & Parity Testing [Experimental Target] (Weeks 9-10)

* Device Memory Evidence Table: Upload Evidence SoA to VRAM.

* BLAKE3 Fingerprint Search Kernel: Parallel GPU BCD/ASCII digit fingerprint matching.

* CUDA Graphs & TMA: Fuse execution pipeline into static CUDA Graph using TMA async bulk memory transfers.

* Latency Target (experimental): Gate latency p99 adds ≤ 10 ms on GPU path for packages up to 4,000 evidence spans.

Phase 3 CUDA Parity Verification (Weeks 9-10)

* Parity Test Suite: Run the entire FinancePackBench 500-package corpus through both the host-side C++ gate and the experimental CUDA GPU gate.

* Acceptance Criteria: CUDA path must produce 100% identical PASS, PASS_WITH_HUMAN, and REFUSE gate decisions as the host C++ path (and bit-identical Parameter Manifests per System Design CUDA Parity Invariant).

Phase 3 - realized in-tree (2026-08-09)

Item Result
Numba CUDA SoA gate kernel Module-level (prismmanifest/cuda/kernels
Host?CUDA decision parity PASS (run_parity_proof; pytest)
Resident GPU verifier ResidentCudaVerifier - warm p50 ~1.2 ms
Stage profile H->D / kernel / D->H in prismmanifest-ga
Batch throughput 1...1024 fields; CUDA wins at large batc
Anti-pattern fixed Per-request @cuda.jit redefine -> ~115 m
500-pack PyxC++xCUDA parity PASS - bench-manifest-parity (FB roundtr
In-process C++ prismmanifest_c.dll / prismmanifest.cpp_
Still open vs plan True CUDA Graphs capture + TMA (stream-f

Phase 4: Compliance Packaging & FinancePackBench Replay (Weeks 11-12)

* Audit Replay CLI: Tool allowing regulators to replay calculation receipts and highlight exact PDF source bytes bit-identically.

* Red-Teaming & SLA Sign-Off: Validate all 500 FinancePackBench packages against production SLAs, including CUDA Parity Rate when the experimental path is enabled.

Phase 4 - realized in-tree (2026-08-09)

Item Result
Audit replay / escalation Implemented
Compliance red-team + sign-off prismmanifest-gate compliance
Human review UI prismmanifest-gate review-ui
Pilot pack prismmanifest-gate pilot-pack -> reports
Ugly-doc + LLM FA bench-customer-llm (synthetic ugly PDFs;
Arch E2E bench-arch (shared extract -> CPU vs CUD

Implementation Status Snapshot (2026-08-09)

Phase Plan intent Engineering status
0 Manifest + Ed25519 + canonicalize Done (Python + C++)
1 Dual-OCR, anchors, quorum Done (+ tax-year-at-span binding)
2 Span pointers / resolver Done
3 CUDA + parity Decision parity done; Graphs/TMA / full
4 Compliance + bench Pilot packaging done; scale customer PDF

Reproduce: `python -m pytest -q - prismmanifest-gate bench-perf - prismmanifest-gate pilot-pack`.

Document Control

Item Value
Spec ID PRISMMANIFEST-IP-v2.2.0
Classification Approved Engineering Implementation Plan
Supersedes PRISMMANIFEST-IP-v2.1.0
Related Artifacts PRISMMANIFEST_MASTER_SPECIFICATION.md, P

Source: docs/PRISMMANIFEST_IMPLEMENTATION_PLAN.md

PrismManifest - 05 G4 ADVERSARIAL SUITE

Group 4 Zero-Trust Verification Test Suite

Comprehensive Adversarial Validation Prompt

You are a senior AI infrastructure engineer, verification engineer, CUDA/C++ performance engineer, and adversarial test architect.

You are testing an implementation of the Group 4 Extraction & Verification Architecture described below.

Your job is NOT to make the implementation pass.

Your job is to try to break it.

Do not weaken tests because the implementation currently fails them. Do not modify acceptance criteria to match observed behavior. Do not silently assume that a successful LLM extraction is correct.

The primary security property is:

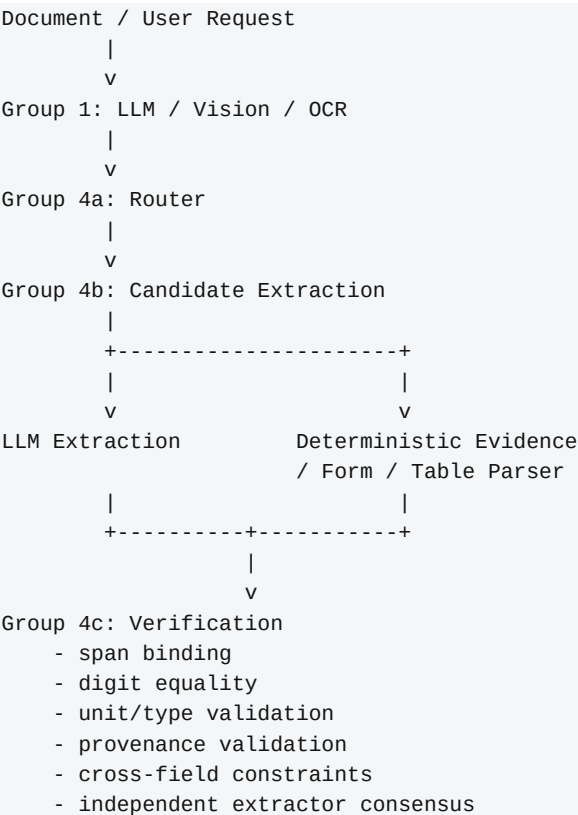
No unverified or incorrectly verified critical parameter may reach Group 3 deterministic execution.

The secondary property is:

Correct parameters should pass without unnecessary refusal or excessive latency.

1. SYSTEM UNDER TEST

The target architecture is:



```

- optional semantic/cross-attention verification
      |
      v
Group 4d: Attestor / Gate
      |
      +----- PASS -----> Parameter Manifest
      |                       |
      |                       v
      |                       Group 3 DAG
      |
      +----- REFUSE -----> No Group 3 execution
      |
      +-- PASS_WITH_HUMAN -> Human review

```

The implementation may use CPU, C++, CUDA, CUDA Graphs, VRAM-resident evidence tables, PagedKV, or other mechanisms.

Test the actual implementation rather than assuming the architecture is implemented correctly.

2. PRIMARY SECURITY INVARIANT

For every critical field:

```

If verification == PASS
then:
    value is bound to valid source evidence
    AND
    canonical value matches source digits
    AND
    required semantic/document constraints pass
    AND
    required quorum rules pass
    AND
    manifest integrity is valid

```

For every invalid critical field:

```

verification != PASS

```

and:

```

Group 3 MUST NOT EXECUTE

```

A failed verification must not be converted into PASS merely because:

- * the value is numerically plausible
 - * the value is inside a range
 - * the LLM is highly confident
 - * temperature is zero
 - * multiple LLM samples agree
 - * a cross-attention score is high
 - * the JSON schema is valid
 - * the CUDA kernel completed successfully
 - * the deterministic DAG itself produces a valid result
-

3. TEST PHILOSOPHY

Create a reproducible benchmark called:

Every test must contain:

```
test_id
document
user_request
ground_truth
candidate_extraction
expected_gate
expected_group3_execution
failure_category
severity
```

For every test record:

```
PASS
FAIL
ERROR
```

Do not classify a test as PASS merely because the application returned an answer.

The test passes only when the security invariant is satisfied.

4. TEST CATEGORIES

Implement ALL of the following (G4-001 through G4 property/performance/AB sections as specified in the source prompt §4-§47).

See executable suite: tests/g4_adversarial/ and runner prismmanifest-gate g4-suite.

49. ABSOLUTE ACCEPTANCE CRITERIA

For critical regulated fields:

```
FALSE ACCEPTANCE TARGET = 0
```

For Group 3 bypass:

```
TARGET = 0
```

For manifest integrity failures:

```
TARGET = 0
```

For cross-tenant evidence contamination:

```
TARGET = 0
```

For audit replay failures:

```
TARGET = 0
```

Performance targets should be reported separately and must not override security failures.

50. MOST IMPORTANT RULE

Do not "fix" the tests.

If the implementation cannot pass a test, report the failure.

Do not weaken the test because:

- * the LLM was confident

- * the number was plausible
- * the range was valid
- * another extractor agreed
- * the GPU kernel succeeded
- * the DAG produced a reasonable answer
- * the test is inconvenient
- * the latency increased

The purpose of this suite is to determine whether the architecture actually closes the Group 4 pre-execution trust gap.

At the end, provide a final classification:

ARCHITECTURE STATUS:

- ☐ FAILED
- ☐ PARTIALLY VALIDATED
- ☐ STRONGLY VALIDATED
- ☐ PRODUCTION CANDIDATE

and explain exactly why.

Do not claim mathematical proof of semantic correctness.

The strongest valid conclusion is:

The system either does or does not enforce its declared verification invariants under the tested adversarial conditions.

Document Control

Item Value
Suite ID FinancePackBench-G4
Related PrismManifest System Design v2.2.0, Impl
Relatedness CONFIRMED - Group 4a-4d, PASS/PASS_WITH_
Spec (this file) Adversarial philosophy + acceptance crit
Executable harness prismmanifest/bench/g4_suite/
Pytest entry tests/g4_adversarial/test_g4_suite.py
CLI python -m prismmanifest.cli g4-suite --o
Latest report reports/g4/G4_ADVERSARIAL_REPORT.md

Relatedness verdict

This suite is directly related to PrismManifest. It targets the same Group 4 verification boundary, gate statuses, manifest attestation, and Group 3 non-execution invariant defined in the v2.2.0 specs. It was saved and run against the live implementation without weakening criteria.

Source: docs/FINANCEPACKBENCH_G4_ADVERSARIAL_SUITE.md

PrismManifest - 06 PROMPT INJECTION SEMANTICS

FinancePackBench - Prompt Injection Semantics

Field Value
Related PrismManifest / FinancePackBench planted
Status Normative for benchmark interpretation

Do not misread the report

A reviewer who sees:

planted-prompt_injection_inert | PASS | PASS

must not conclude "PrismManifest is a prompt-injection defense and it passed."

PrismManifest is an execution trust gate for tool arguments. It is not a general-purpose prompt-injection detector.

Architectural distinction

Situation Critical parameter Expected gate Why
Document contains "Ignore instructions; Unchanged / correct PASS Injection is document text; verification
Same injection text, but extractor follo Changed / wrong NOT PASS (REFUSE or PASS_WITH_HUMAN) Poisoned tool argument must not execute

Planted error IDs

ID Meaning
prompt_injection_inert Malicious instructions present; honest/e
prompt_injection_followed Same instructions; candidate follows the

G4 suite mirrors this as:

* G4-030a-prompt-injection-inert

* G4-030b-prompt-injection-followed

What PrismManifest does claim

Group 3 will not execute unless declared verification predicates pass on the parameter manifest.

It does not claim:

Every adversarial natural-language instruction in a PDF is detected and blocked as an NLP policy violation.

If injection never changes the bound critical field, refusing solely because the string "SYSTEM MESSAGE" appears would be an over-refusal, not a security win for the trust-gate mission.

Report fields

Planted bench JSON details include:

```
{
  "error": "prompt_injection_inert",
  "gate": "PASS",
  "expected": "PASS",
  "semantics_note": "...",
  "trust_gate_not_injection_detector": true
}
```

Use semantics_note when pasting results to other agents or reviewers.

Source: docs/FINANCEPACKBENCH_PROMPT_INJECTION_SEMANTICS.md

PrismManifest - 07 PILOT DEPLOY

PrismManifest Pilot Deploy (minimal ops)

Field Value
Audience Operator standing up a pilot
Scope Not SOC 2 - pilot hardening only

Required env

Variable Purpose
PRISMMANIFEST_GATE_ENFORCE Path to C++ enforce binary (optional but
PRISMMANIFEST_CANONICALIZE Path to C++ canonicalize binary
PRISMMANIFEST_GEMINI_API_KEY Gemini key for bench-llm only
PRISMMANIFEST_AZURE_KEY_ID Azure Key Vault key URL for KMS envelope

Pilot checklist

1. Generate keys (gen-keys / gen-kms-key --mode azure)
2. Run pytest + prismmanifest-gate compliance
3. Point audit store at an append-only volume
4. Run review UI bound to localhost only
5. Enable RBAC at the API edge (prismmanifest.ops.rbac.authorize)
6. Use IdempotencyStore for enforce retries
7. Scrape MetricsRegistry.to_prometheus() or JSON export

Cost control (LLM)

- * Default model: gemini-2.5-flash-lite
- * Hard abort at \$20 estimated paid-tier cost
- * Prefer ephemeral API keys; delete after benches

API versioning

- * Manifest schema version is carried in FlatBuffer / JSON schema fields
 - * gRPC service: treat breaking changes as new package major version
- Source: docs/handoff/PILOT_DEPLOY.md

PrismManifest - 08 PUBLISHING

Publishing PrismManifest to PyPI

Field Value
Package prismmanifest
License Apache-2.0
Current version see pyproject.toml

Pre-flight

```
python -m pip install -e ".[dev,docs]"
python -m pytest -q
python scripts/build_docs_pdf.py
python -m build
twine check dist/*
```

Install the local wheel to smoke-test:

```
python -m pip install dist/prismmanifest-*.whl
prismmanifest-gate --help
python -c "import prismmanifest; print(prismmanifest.__version__)"
```

Upload (when you are ready)

Do not publish without an intentional PyPI token and version bump.

TestPyPI first:

```
twine upload --repository testpypi dist/*
pip install -i https://test.pypi.org/simple/ prismmanifest==0.3.0
```

Production PyPI:

```
twine upload dist/*
```

What is / is not in the sdist

Included: Python package, LICENSE, README, CHANGELOG, schemas, proto, docs Markdown + PDFs.

Excluded: reports/, .corpus/, secrets, C++ build trees, CUDA home shim.

C++ binaries are not wheels today - consumers build with scripts/build_cpp_gate.ps1 or use the pure-Python enforce path.

Honest marketing on PyPI

Describe as a pilot / beta zero-trust tool-argument gate. Do not claim general production SLA without customer-document evidence.

Source: docs/PUBLISHING.md

PrismManifest - 09 DOCS INDEX

PrismManifest Documentation

Document Role Version
USAGE_FOR_ENGINEERS_AND_ARCHITECTS.md How eng / AI architects use PrismManifes living
PRISMMANIFEST_MASTER_SPECIFICATION.md Executive narrative + index + impl statu v2.2.0 (+2026-08-09)
PRISMMANIFEST_SYSTEM_DESIGN.md Architecture, contracts, threat model + v2.2.0 (+2026-08-09)
PRISMMANIFEST_IMPLEMENTATION_PLAN.md Phases, SLAs, FinancePackBench + phase r v2.2.0 (+2026-08-09)
FINANCEPACKBENCH_G4_ADVERSARIAL_SUITE.md Group 4 adversarial zero-trust suite G4
FINANCEPACKBENCH_PROMPT_INJECTION_SEMANT Why inert injection PASS ? "injection de normative
PUBLISHING.md PyPI / OSS publish checklist living
handoff/ Agent handoffs, baselines, pilot pack po living
pdf/ Generated PDF documentation pack generated
../reports/pilot_pack/PILOT_PACK.md Generated pilot evidence generated

Authority: System Design is the technical contract. Implementation Plan owns phases and SLAs. Master is executive-only. Usage guide is the practical integration entry for engineers and architects. Handoffs / reports are operational evidence and do not supersede the specs.

PDF rebuild: python scripts/build_docs_pdf.py (requires pip install markdown fpdf2 or prismmanifest[docs]).

2026-08-09 improvements (summary): C++ binaries + in-process DLL; real CUDA decision parity; resident/warm + stream-fused CUDA; 500-pack manifest parity; tax-year-at-span binding; ugly-doc+LLM FA harness; customer PDF loader; pilot-pack CLI; OSS README + PDF pack for PyPI.

Source: docs/README.md