Multimodal AI: Towards Deterministic Verification Layers for Large Language Models

Author:

Nikhil Tripathi
Solution Architect with over a decade of experience in enterprise IT architecture, LLMs, AI systems integration, and digital transformation.

Email: ***nikhil.k.tripathi@gmail.com***

1. Executive Summary

Large Language Models (LLMs) can now interpret and generate text, images, and code fluently — but their reasoning remains **probabilistic**, not **proven**.
Humans verify through **perception, logic, and experience**; LLMs predict words based on **statistical likelihood**.

To close this gap, we introduce the **Deterministic Verification Layer (DVL)** — a symbolic, rule-based layer that **computes, validates, and explains** before the LLM is allowed to narrate.

The DVL fuses:

- **RDF** knowledge graphs for structured semantics
- **SHACL** constraint reasoning for logical validation
- **Deterministic computation** for proof-based accuracy
- **Provenance tracking** for auditability
- A **policy of refusal** for impossible statements

This approach gives machines a **human-like verification sense** — they "check before believing."

Current Status

- **Math Verification Layer — Completed (PoC ready)**
  Deterministic evaluator + RDF/SHACL validation + Gradio UI in Google Colab.
- **Biology Validation Layer — In Progress**
  Ontology (Plant/Animal) and exception rules implemented; controller wiring and test coverage underway.

---

2. Problem Context

2.1 The Human–Machine Perception Gap

Humans assess reality through multimodal perception — sight, sound, memory, and logic — ensuring internal consistency before acting.
LLMs, however, operate on **token prediction**, not factual verification.
They can:

- Produce numerically wrong yet fluent answers,
- Contradict themselves between sessions,
- Lack an audit trail or ground truth link.

When deployed in regulated sectors (finance, law, engineering, healthcare), these weaknesses become structural risks.

2.2 Why Existing Safeguards Fall Short

Approach	Strength	Limitation
Guardrails & Filters	Style & safety	Don’t verify correctness
RLHF	Improves helpfulness	Doesn’t prove truth
Confidence Scores	Internal metrics	Non-reproducible
Provenance Frameworks (C2PA)	Track content origin	Don’t test factuality

Hence, a **verification-first** architecture — independent of model probability — is needed.

---

3. The Deterministic Verification Layer (DVL)

3.1 Concept Overview

The DVL acts as a **semantic firewall** between the LLM and its users.
Each claim generated by an AI system is:

1. Encoded into **RDF triples** (facts),
2. Validated by **SHACL constraints**,
3. Recomputed or verified deterministically,
4. Returned **only if validated** — else politely refused.

3.2 The Human-Perception Analogy

Human Function	DVL Equivalent
Sensory Perception	Multimodal input parsing (text, image, audio)
Logical Reasoning	RDF graph reasoning
Consistency Checking	SHACL constraint validation
Calculation & Estimation	Deterministic computation engine
Moral/Contextual Response	Polite refusal on impossibilities

The DVL brings *human-like perception and judgment* to AI — ensuring it reasons before responding.

---

4. Modular Architecture

4.1 Layered Overview

```
+----------------------------------------------------------------+
|                  USER / FRONTEND INTERFACE                      |
|     (Gradio UI / Web App / Teams Bot / MCP Client)              |
+----------------------------------------------------------------+
|              DETERMINISTIC VERIFICATION LAYER (DVL)             |
|----------------------------------------------------------------|
|  1. Input Interpreter          | Entity & Relation Extraction   |
|  2. RDF Encoder                | Structured Triple Generation   |
|  3. Domain Validator           | SHACL Constraint Validation    |
|  4. Deterministic Engine       | Math / Logic Verification      |
|  5. Exception Reasoner         | Impossible / Context Rules     |
|  6. Provenance Recorder        | Hash, Timestamp, Audit Trail   |
|  7. Language Narrator (LLM)    | Post-Validation Explanation    |
+----------------------------------------------------------------+
|           FOUNDATION MODEL (DeepSeek / GPT / Claude)            |
|       Optional — used for narration after validation            |
+----------------------------------------------------------------+
|         TOOL ORCHESTRATION LAYER (MCP-Compatible)               |
|     math.evaluate | rdf.query | shacl.validate | provenance.log |
+----------------------------------------------------------------+
```

4.2 Module Descriptions

1. Input Interpreter

- Accepts text or multimodal inputs (e.g., image + caption).
- Detects entities, verbs, and relationships.
- Outputs structured claims for validation, e.g.:
  - "The cat ran." → (`Cat`, `performed`, `Running`)
  - "5 + 7 = 12." → (`Equation`, `operand1`, `operand2`, `operator`).

2. RDF Encoder

- Encodes inputs as RDF triples with namespaces and unique identifiers.
- Example:
- `ex:RunEvent1 a act:Running ;`
- `    act:agent ex:ThisCat ;`
- `    act:time  "2025-10-26T12:00Z"^^xsd:dateTime .`

3. Domain Validator (SHACL)

- Applies SHACL shapes to enforce rules:
  - Structural (e.g., every event must have an agent).
  - Semantic (e.g., only animals can run).
  - SPARQL constraints for exception messages.
- Returns deterministic validation reports.

4. Deterministic Engine

- Recomputes mathematical or logical expressions exactly.
- Implements safe AST parsing using Python's `decimal` for precision.
- Used in the Math PoC for verifiable arithmetic and Boolean logic.

5. Exception Reasoner

- Contains explicit "impossible" rules (e.g., `Plant cannotPerform Running`).
- When violated, returns:

  "I politely refuse your statement. What you are saying is highly unlikely unless there is some exception."

6. Provenance Recorder

- Captures rule version, timestamp, and validation result.
- Generates immutable logs suitable for audit or blockchain anchoring.
- Aligns with **C2PA** and **ISO 42001** provenance standards.

7. Language Narrator

- Invokes the LLM **only after validation passes**.
- Generates natural, human-readable explanations of validated facts.
- For invalid claims, produces the standardized **polite refusal** message.

4.3 Tool Orchestration (MCP Integration)

All components can be exposed as MCP-style tools:

```
math.evaluate()         → deterministic calculator
rdf.query()             → triple retrieval
shacl.validate()        → constraint checking
provenance.log()        → audit record creation
```

This allows modular, service-based scaling while ensuring strict sequencing — validation **always precedes** narration.

---

5. Proofs of Concept

5.1 Mathematical Verification PoC — Status: Completed

Stack: Python + RDFLib + PySHACL + Gradio
Functionality:

1. Parse arithmetic expression or DeepSeek JSON response.
2. Compute deterministically with high precision.
3. Encode as RDF triples (`math:operand1`, `math:operator`, `math:computedResult`).
4. Validate structure and datatypes with SHACL.
5. Return a structured verdict: ✅ Valid / ❌ Mismatch / ⚙️ Unverified.
6. Log timestamped provenance.

UI: Interactive Gradio app (Google Colab).
Optional: DeepSeek API integration for automatic JSON extraction.

Output Example


```
{
 "status": "valid",
 "answer": {"computed": "13.0000"},
 "evidence": {"constraints_passed": true},
 "provenance": {"timestamp_utc": "2025-10-26T12:34Z"}
}
```

5.2 Biological Validation PoC — Status: In Progress

Domain: Biology — Plants & Animals
Ontology: `Plant`, `Animal`, `Cat`, `Photosynthesis`, `Running`

Implemented:

- RDF ontology and capabilities (`Plant canPerform Photosynthesis`, `Cat canPerform Running`)

- Exception rules (`Plant cannotPerform Running`, `Cat cannotPerform Photosynthesis`)
- SHACL shapes + SPARQL constraints to enforce capability logic
- **Polite refusal mechanism** triggered on biologically impossible statements

Pending Work:

- Controller wiring (capability → event → SHACL → response)
- Test cases (`cat ran` ✅, `plant ran` ❌, `plant photosynthesized` ✅, `cat photosynthesized` ❌)
- Integration with MCP tool endpoints (`rdf.query`, `shacl.validate`)

Expected Outcome:
Validation results identical to deterministic perception — only feasible actions accepted.

---

6. Expanding Toward Full Multimodal Perception

6.1 Domain Modules

- **Mathematics:** Arithmetic, algebraic consistency, logical evaluation.
- **Biology:** Capability mapping, ecological logic.
- **Languages:** Lemma → Action mapping, tense and polarity handling.
- **Engineering:** Mechanical feasibility, energy constraints, material logic.
- **Social Sciences:** Normative cause-effect validation.
- **Arts:** Conceptual relations and emotional semantics.

Each module is stored as an RDF graph pack:
`/core`, `/capabilities`, `/events`, `/shapes`, `/exceptions`.

6.2 Bridge Graphs

Domain bridges link:

- Language ↔ Action (`ran` → `act:Running`)
- Vision ↔ Object (detected image class → RDF entity)
- Math ↔ Physics (quantitative laws → physical validation)

These enable human-like **cross-domain perception** across text, image, and numeric modalities.

---

7. Evaluation Metrics

Metric	Description	Target
Deterministic Accuracy	Correctness after validation	≥ 95%
Explainability	Outputs with RDF + rationale	100%
Latency Overhead	Added verification time	< 500 ms
Consistency	Reproducibility across runs	100%
Provenance Integrity	Immutable audit chain	Full traceability

---

8. Strategic Significance

- Establishes a **trust layer** for all multimodal AI systems.
- Enables **verifiable intelligence** — beyond token prediction.
- Provides **regulatory compliance** under EU AI Act / NIST RMF / ISO 42001.
- Bridges **symbolic precision** with **foundation model perception**.
- Lays groundwork for **Provenance-as-a-Service (PaaS)** ecosystems.

---

9. Roadmap

Phase	Focus	Deliverables	Status
M0	Math PoC	Deterministic RDF/SHACL validator + Gradio UI	✅ Completed
M1	Biology PoC	Ontology + Capabilities + SHACL Exceptions + Controller Tests	🔄 In Progress
M2	Language & Lemma Mapping	NLP → Ontology Bridge + Coreference Resolution	Planned
M3	MCP Orchestration	Tool Endpoints (`math.evaluate`, `rdf.query`, `shacl.validate`, `provenance.log`)	Planned
M4	Domain Expansion	Arts / Engineering / SocialScience Packs + Bridge Graphs	Planned
M5	Provenance & Dashboard	Immutable Logs + Metrics Interface	Planned
M6	Publication & Demo	Public Repo + White Paper Release	Planned

10. Human-Centric Interpretation

LLMs were built *for* humans — to converse naturally — but not *like* humans, who verify facts before speaking.
By adding the DVL, we give LLMs a **verification sense**:

- They *see* through RDF structure,
- They *judge* through SHACL validation,
- They *re-check* via deterministic computation,
- They *respond* with contextual politeness when logic fails.

Thus, DVL is the missing bridge between **human perception** and **machine reasoning**.

11. Conclusion

The **Deterministic Verification Layer** transforms large language models into **auditable reasoning systems**.
It merges **symbolic precision**, **mathematical determinism**, and **human-like perception** into a single verifiable architecture.

By computing and validating before narrating, AI systems evolve from *probabilistic assistants* to *accountable intelligence partners* — the foundation of trustworthy multimodal AI.

---

12. Appendix

Appendix A — Math Verification Layer

- **Stack:** Python + RDFLib + PySHACL + Gradio
- **Functions:** Deterministic evaluation → RDF encoding → SHACL validation → Provenance logging.
- **Verdicts:** ✅ Valid / ❌ Mismatch / ⚙️ Unverified.
- **UI:** Gradio interface with shareable Colab link.

Appendix B — Biology Validation Layer

Status: 🔄 In Progress

- RDF ontology + SHACL exception constraints implemented.
- Controller + test cases pending.
- Produces polite refusal on impossible events.
- Expands human-like plausibility perception in LLM systems.