LETS: The Deterministic Pipeline Architecture

Implementing Type_Safe, Debuggable Data Flows for Large Language Models.

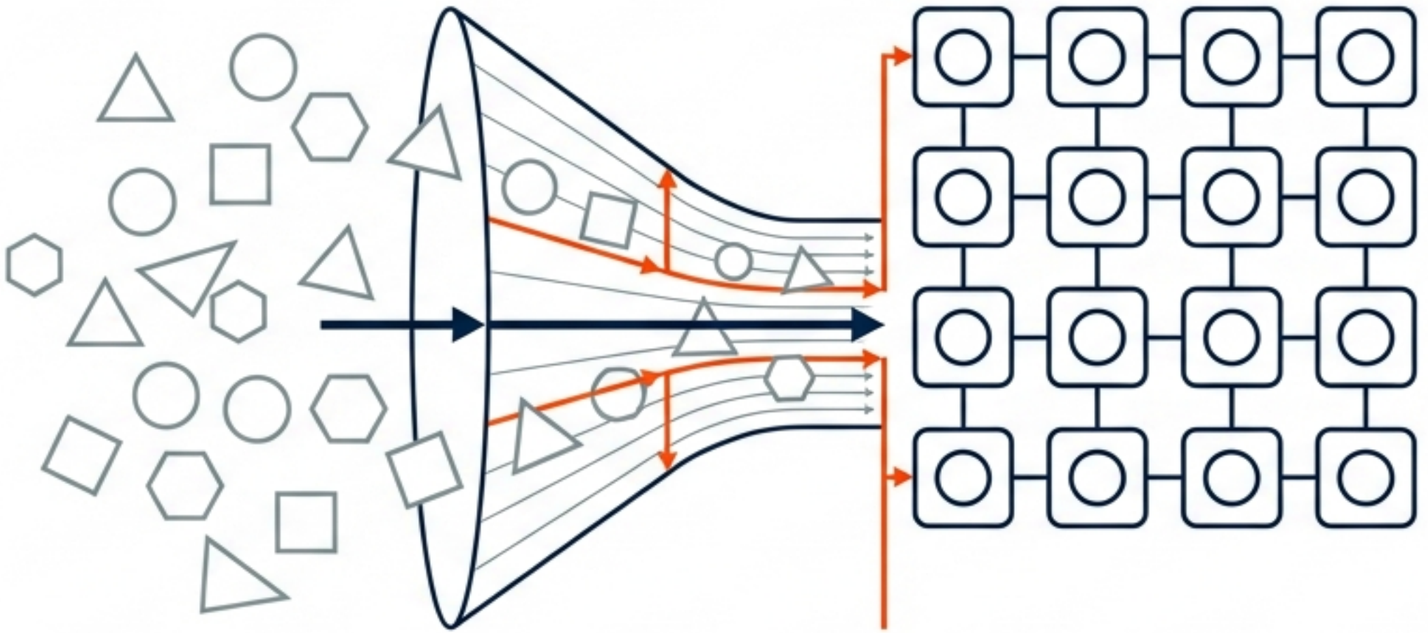


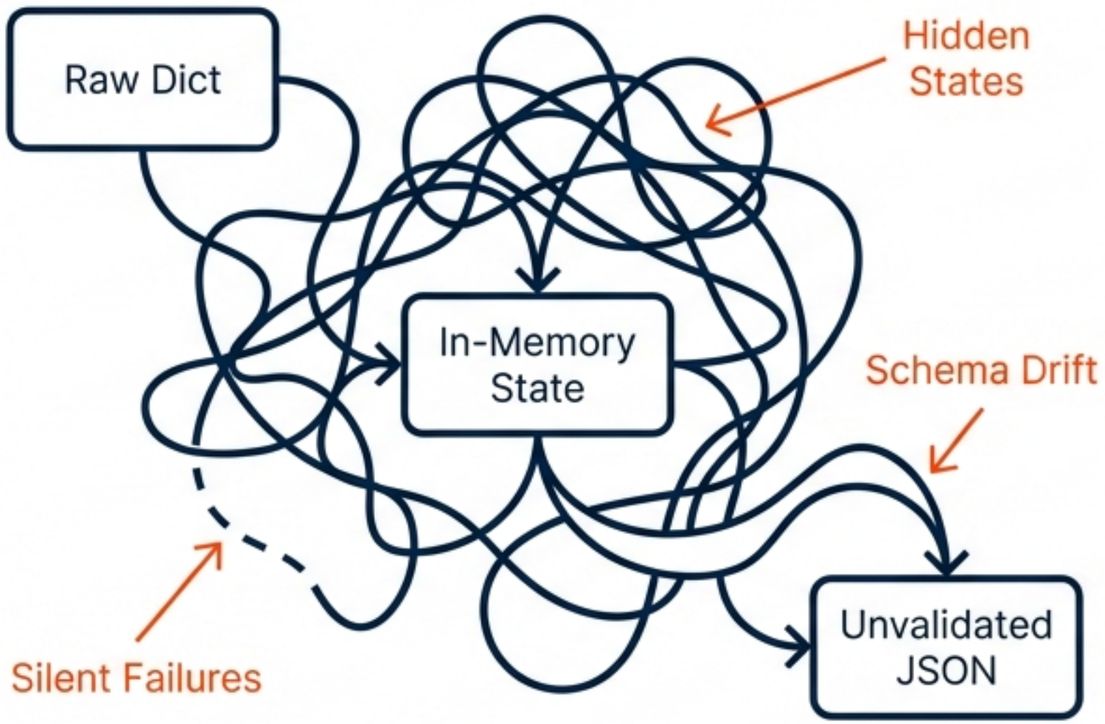
Figure 1.0: Transformation of Unstructured Context into **Deterministic State**.

Methodology v3.69.1
Author: Dinis Cruz & ChatGPT Deep Research
Doc Type: Architectural Specification

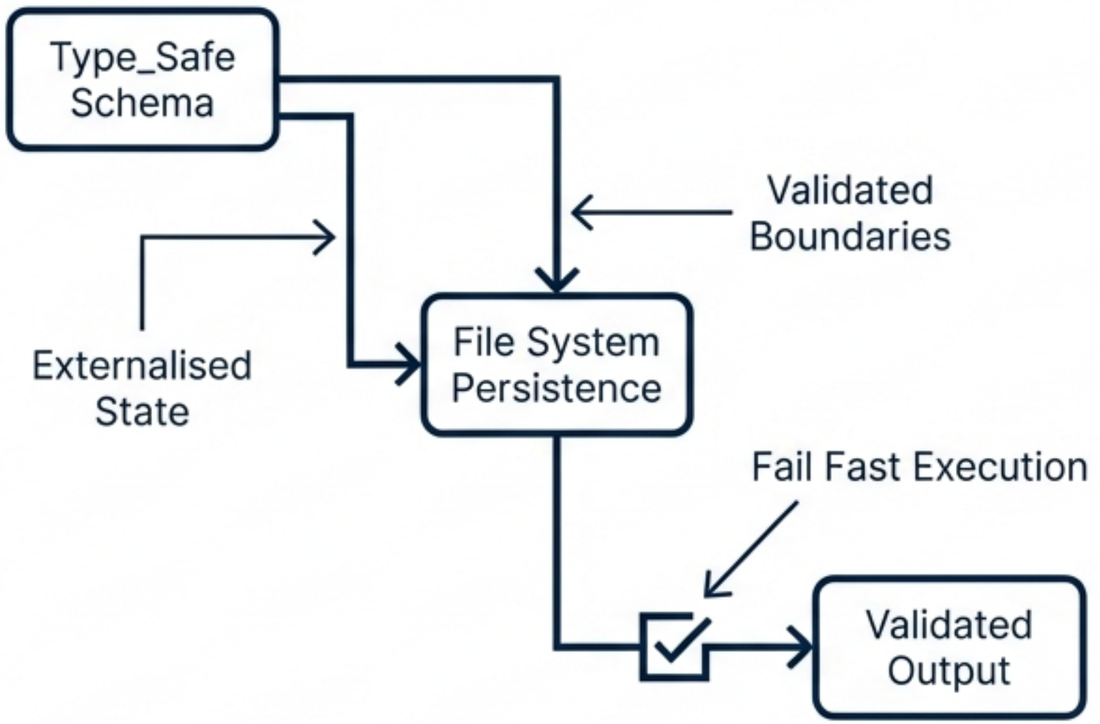
The Challenge: Taming LLM Non-Determinism

Why traditional ETL pipelines fail when applied to probabilistic models

Traditional Approaches (The Spaghetti Pipeline)



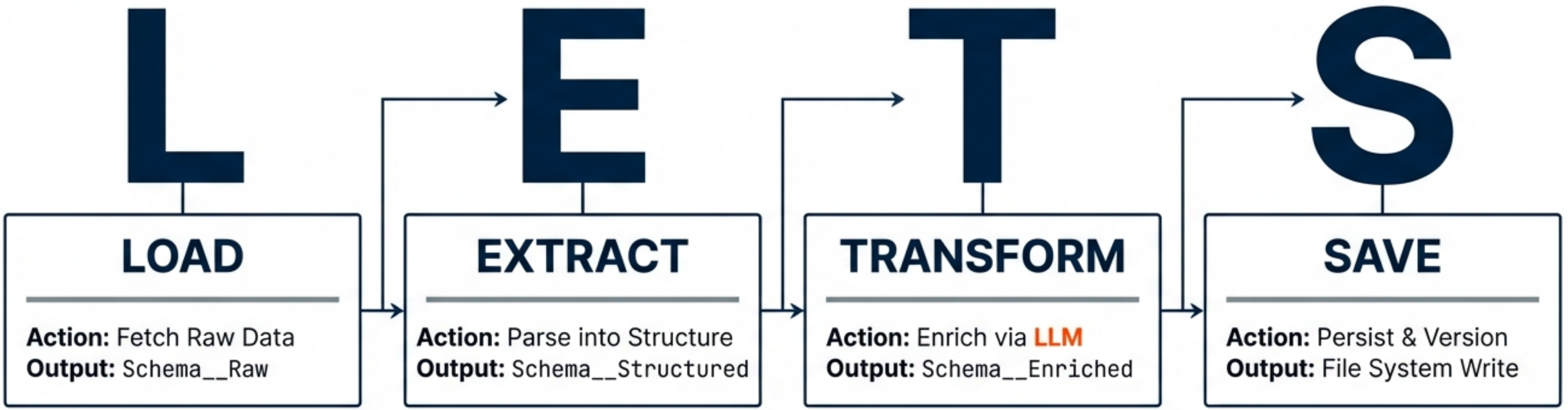
The LETS Architecture



Key Insight: The problem isn't the LLM; it's the lack of strict boundaries around the LLM.

The Methodology: L.E.T.S.

A paradigm shift from In-Memory State to File System as Database.



Core Value Proposition

- `Type_Safe` schemas in → `Type_Safe` schemas out
- Automatic serialization/deserialization
- Full traceability (Provenance)
- Deterministic behavior

The Foundation: Why Type_Safe Matters

LETS relies on Type_Safe classes to preserve data integrity through JSON round-trips. Standard Python types are insufficient for robust pipelines.

The Fragile Way (Raw Dict)

```
data = {
    'email': 'user@test.com',
    'port': 8080
}

# Keys are arbitrary strings
# Values are unvalidated
```

The Type_Safe Way

```
class Schema__User(Json):
    email: Safe_Str__Email
    port : Safe_UInt__Port

# Validated at instantiation
# Self-documenting
```

Result: Validation happens at the boundary (instantiation), preventing errors deep in business logic.

Safe Primitives: Domain-Specific Integrity

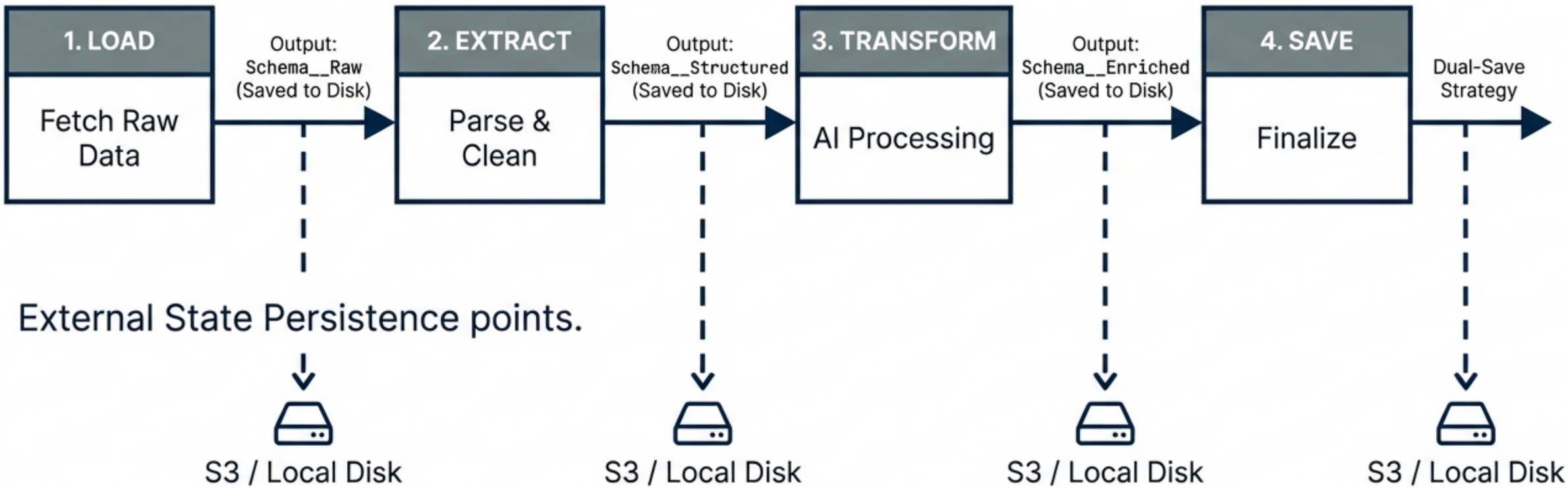
Rule: Use **Safe Primitives, NEVER raw primitives.**

Raw Type	Risk	Safe Primitive Replacement	Benefit
`str`	Allows empty strings, SQL injection, massive payloads	`Safe_Id`, `Safe_Str__Url`, `Safe_Str__Email`	Enforces format (e.g., valid URL) and length constraints.
`int`	Allows negative numbers, overflow	`Safe_UInt`, `Safe_UInt__Port`	Ensures non-negative values and domain bounds (0-65535).
`dict / list`	Structure unknown, keys untyped	`List__[T]`, `Dict__[K,V]`	Strictly typed containers. Rejects invalid items.

Why it matters:

A `Safe_Str__Url` cannot contain a non-URL string. The pipeline crashes immediately if an LLM hallucinates an invalid format, protecting downstream systems.

The 4 Stages: A High-Level View



External State Persistence points.

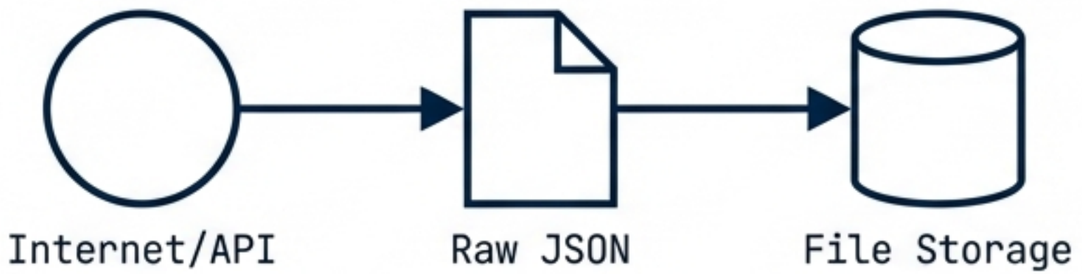
There is no hidden state. Every arrow implies a file write.

Stage Deep Dive: Load & Extract

Getting data into the system safely.

Stage 1: Load

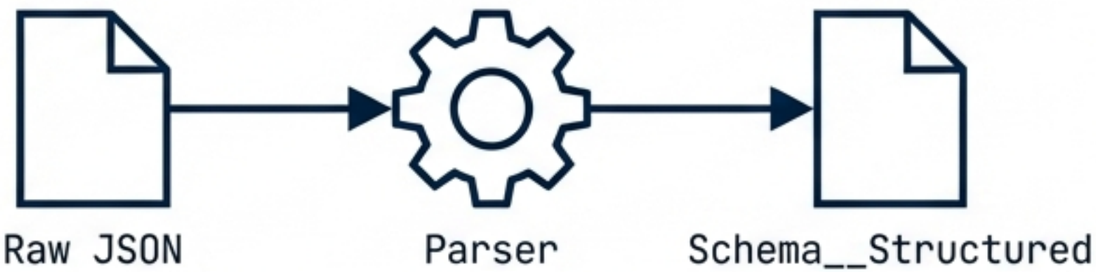
Goal: Capture raw input exactly as received.



Save **immediately**. Do not process yet.
Preserves the exact source of truth.

Stage 2: Extract

Goal: Convert raw data into structured Type_Safe objects.



Validate immediately. Malformed LLM output must fail at `from_json()`, not later.

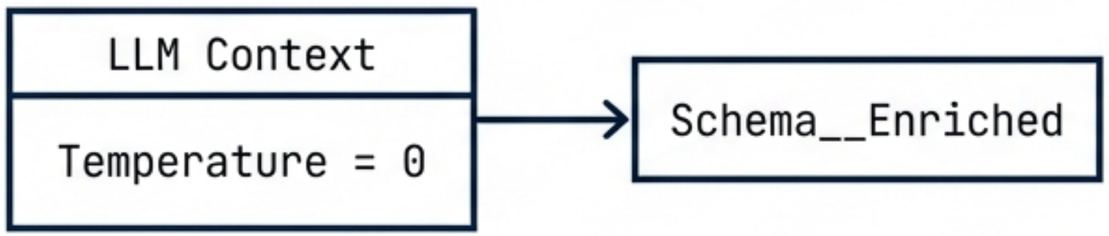
```
data = Schema__Structured.from_json(raw_json)
# If raw_json doesn't match the schema, this line raises an Exception.
```

Stage Deep Dive: Transform & Save

The 'Work' and the 'Commit'.

Stage 3: Transform

Goal: Enrichment via LLM / ML.



Maximum determinism settings applied.

Stage 4: Save (The Dual-Save Strategy)

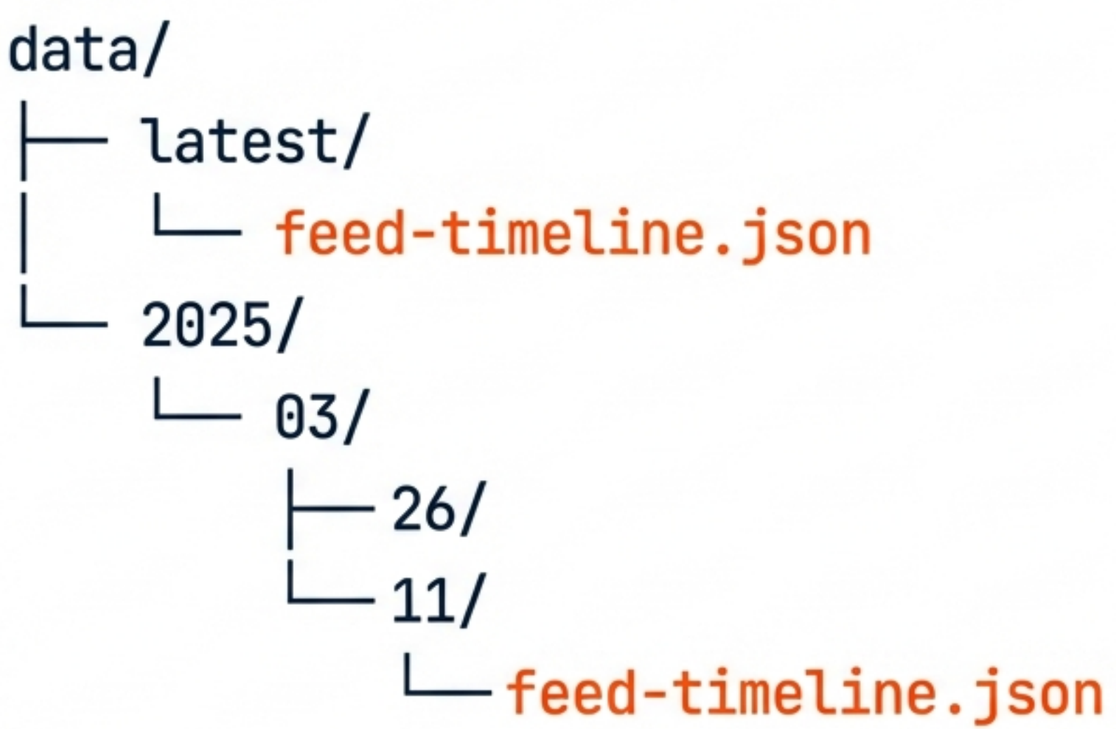
Goal: Persistent Versioning.

Path A: Current state pointer for immediate access.

Path B: Append-only history for debugging and time-travel.

The Data Store: File System as Database

Why we use S3/Disk/Git instead of a complex SQL DB.



Benefits

1. **Human-readable paths.** (Semantic storage).
2. **Zero-tool debugging.** (Just open the JSON file in any editor).
3. **Built-in version control.** (Via temporal folders).
4. **Compatible with MGraph-AI Cache Service.**

The file system provides atomicity and durability without the overhead of a database management system.

The 5 Principles of LETS

1. Ephemerality

Compute is temporary; storage is persistent. No in-memory state between runs.

2. Traceability

Every output links to its input (Provenance metadata).

3. Determinism

Same input + Same code = Same output.

4. Modularity

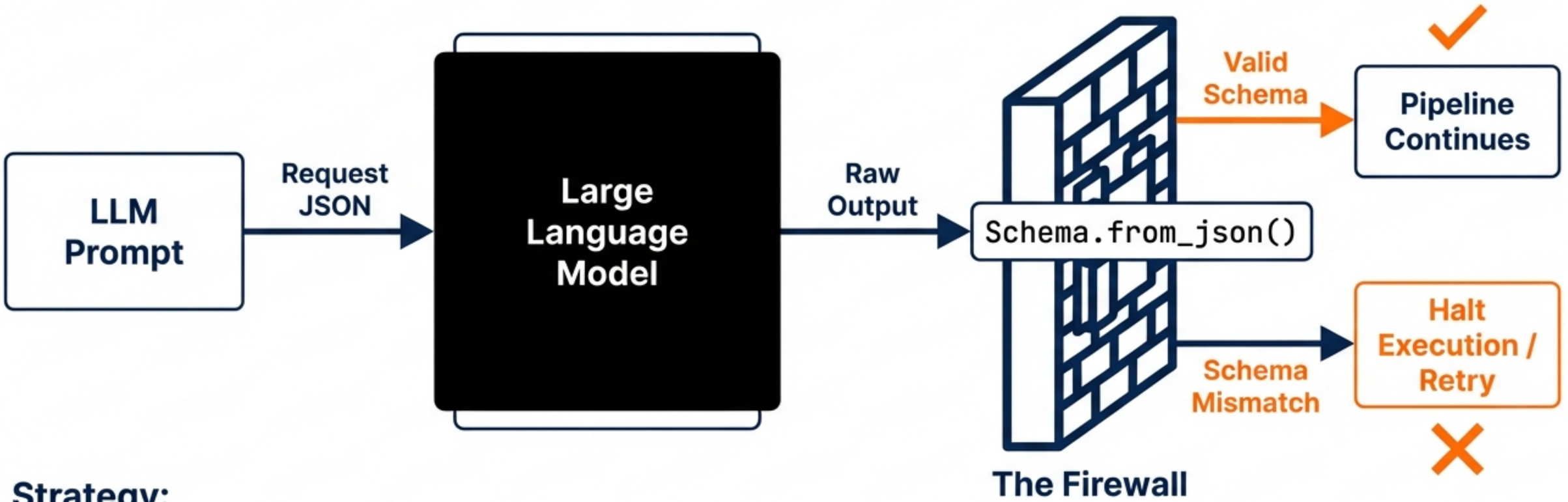
Single responsibility per stage.

5. MVP (Minimum Viable Propagation)

Process only what changed.

LLM Integration Strategy

Preventing hallucinations via strict schema enforcement.

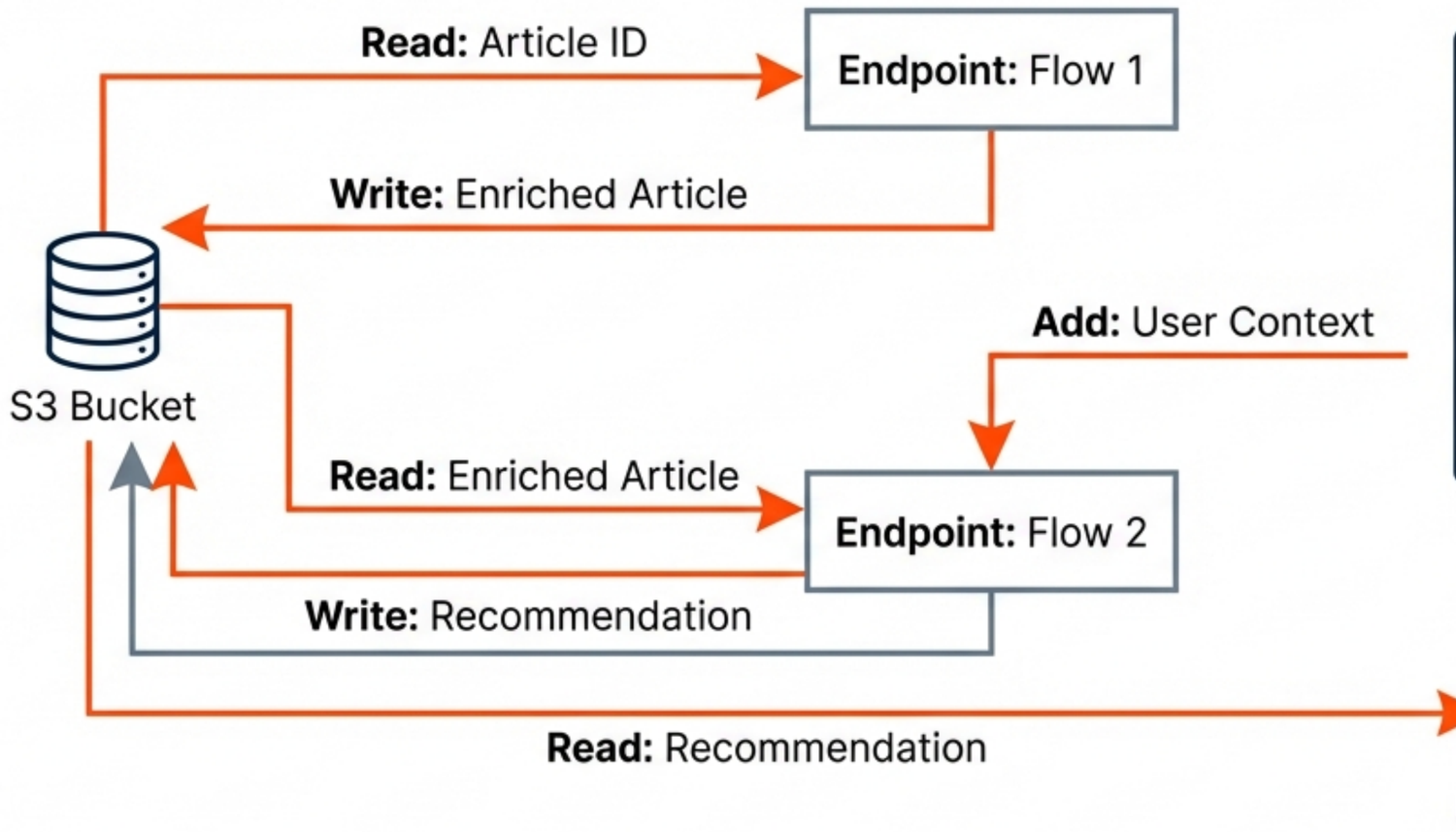


Strategy:

- Never request free-form text.
- Always request structured `JSON` matching the `Type_Safe` schema.
- If outputs vary, check `Temperature > 0`.

Real-World Example: MyFeeds.ai

Production Case Study in Content Personalisation.



Traceability in Action: File naming includes `Article IDs`. Questions like "Why was this recommended?" are answered by simply walking the file path in S3.

Testing & Quality Assurance

Unit Tests

Test each stage independently. Files decouple the stages, allowing isolation.

Golden Files

Store ‘perfect’ JSON versions. Run pipeline. Compare resulting Type_Safe object using `==`.

```
assert result_obj == golden_obj
```

Determinism Check

Ensure re-running the pipeline on the same input produces identical bytes.

The Rules of Engagement

Summary Checklist for Developers

DO

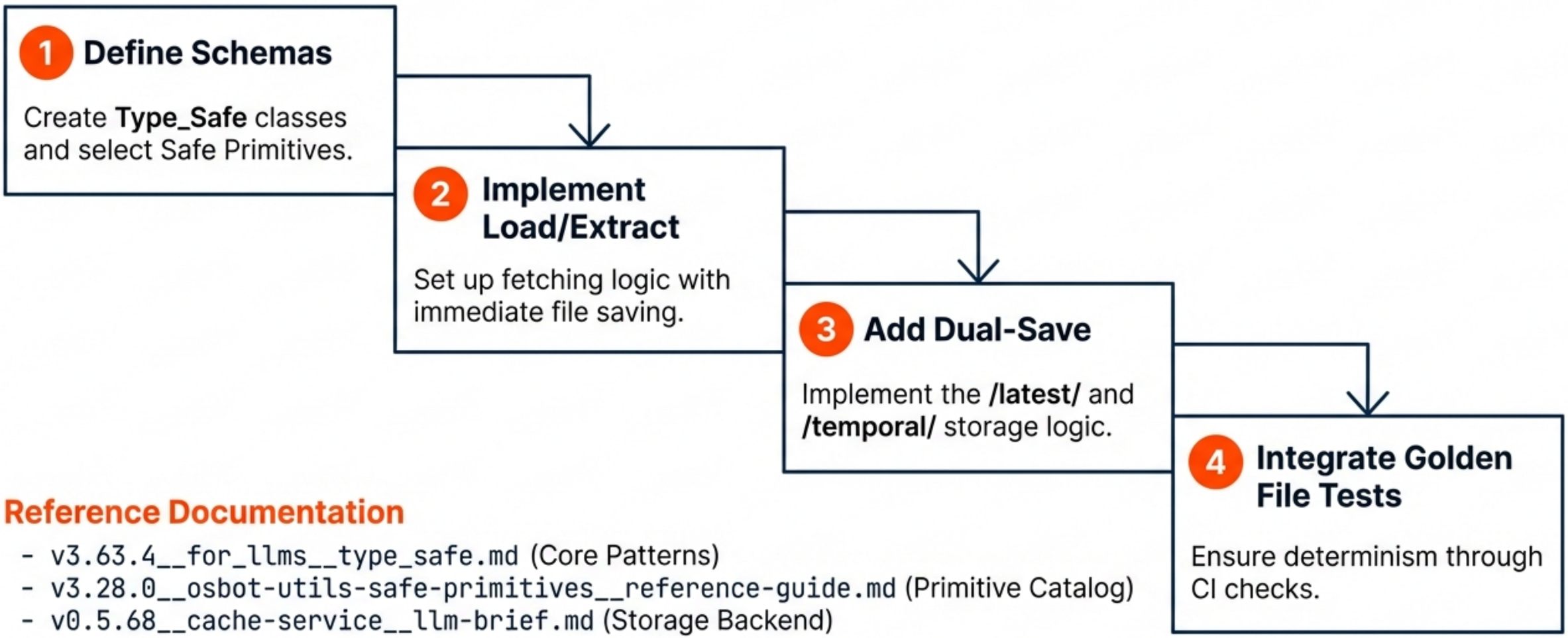
- ✓ Use Safe Primitives (`Safe_Id`), never raw strings.
- ✓ Define Schemas (Pure Data) for *every* stage.
- ✓ Save after *every* stage (Load, Extract, Transform, Save).
- ✓ Use `.json()` and `.from_json()` for serialization.

DON'T

- ✗ Use free-form LLM outputs.
- ✗ Hold state in memory between runs.
- ✗ Put methods inside Schema classes (keep them Pure Data).

Implementation Roadmap

Steps to convert an existing pipeline to LETS.



Reference Documentation

- `v3.63.4__for_llms__type_safe.md` (Core Patterns)
- `v3.28.0__osbot-utils-safe-primitives__reference-guide.md` (Primitive Catalog)
- `v0.5.68__cache-service__llm-brief.md` (Storage Backend)