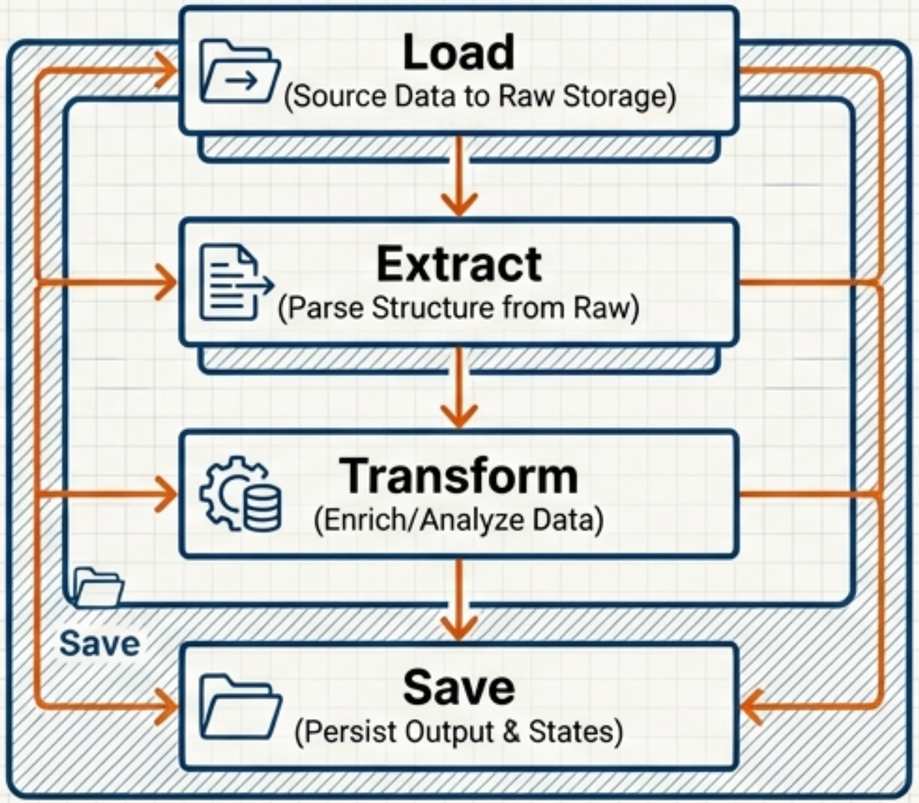
LETS: A Deterministic & Debuggable Data Pipeline Architecture

Moving from Black-Box ETL to Transparent, Versioned Data Lineage

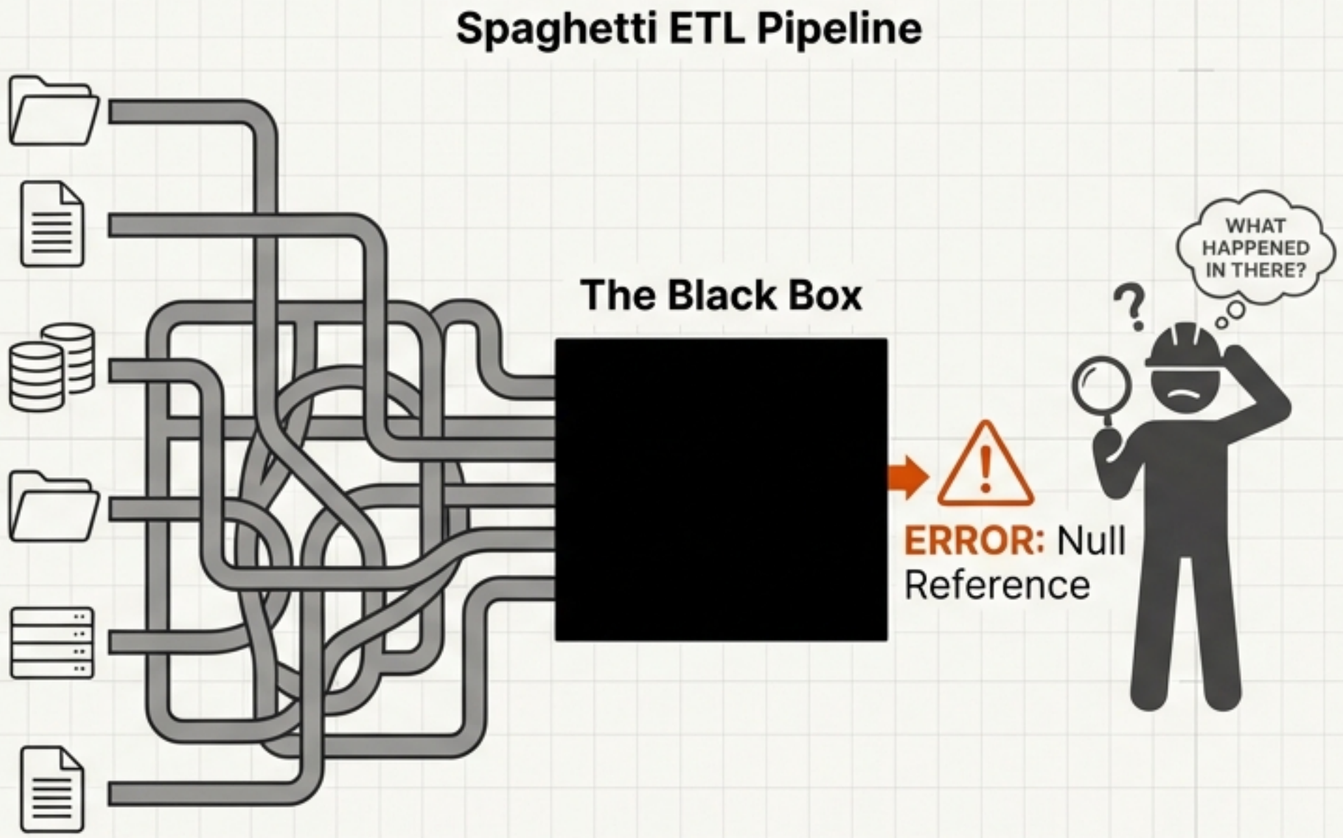


A deliberate evolution of the traditional ETL/ELT paradigm.

Formalising Ephemerality, Traceability, and Modularity for modern AI-driven applications.

Authors: Dinis Cruz & ChatGPT Deep Research

The “Black Box” Crisis in Traditional ETL



1. Opacity

Traditional pipelines perform hidden magic in memory or transient staging. Intermediate states are lost, making it impossible to audit or understand the data transformation journey step-by-step.

2. The Debugging Trap

Failures become detective work rather than unit tests. When a pipeline breaks, you cannot easily answer: “Which specific source record caused this?” The lack of granular visibility turns simple errors into complex investigations.

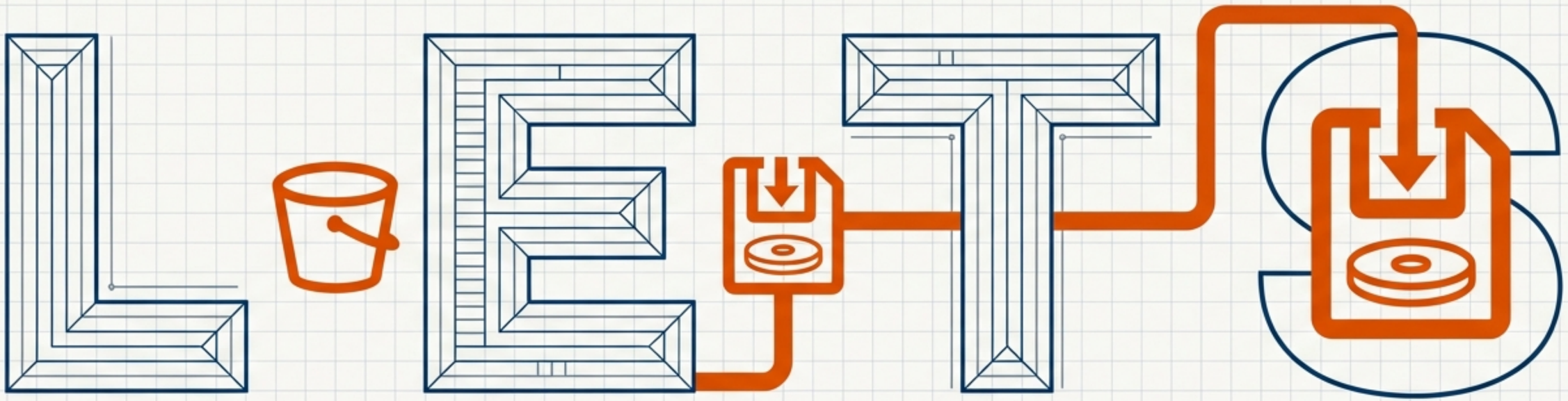
3. Lack of Provenance

A lack of versioning and reproducibility makes it difficult to provide any form of accountability. Without a clear lineage, it is hard to prove the integrity or origin of the final data product.

4. The AI Risk

In the age of GenAI, black-box pipelines create hallucinations that cannot be traced back to their source. If the training data is processed obscurely, the resulting model outputs can be unreliable and undefendable.

The LETS Paradigm: Persistence as the Default



Load
(Subtext: Source Data to Raw Storage)

Extract
(Subtext: Parse Structure from Raw)

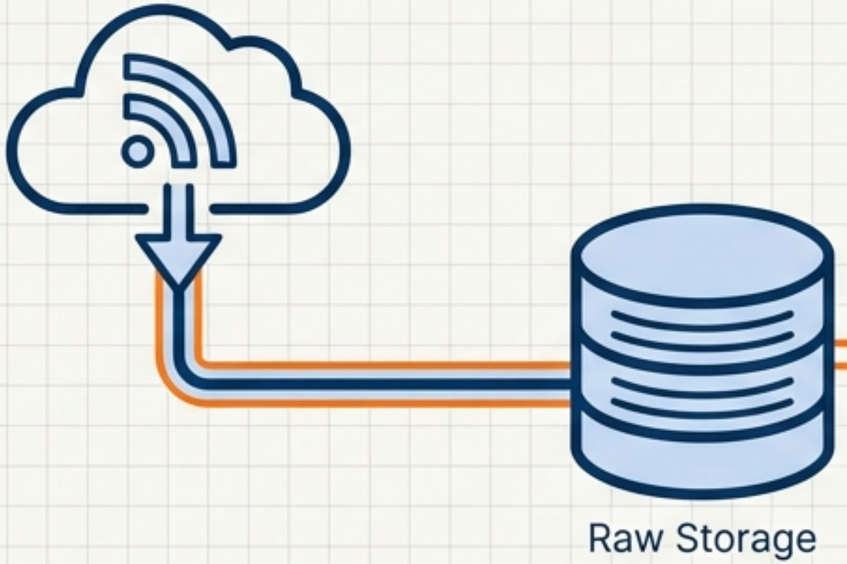
Transform
(Subtext: Enrich/Analyse Data)

Save
(Subtext: Persist Output & States)

The Differentiator: While 'Save' is the final letter, in practice, it occurs after every major phase. The file system is treated as a first-class database, making intermediate states durable and inspectable.

Load and Extract: Establishing an Immutable Baseline

Load



Capture source data verbatim.
This is the immutable baseline.

```
flow-1-download-rss-feed
Output: feed-data.json (Raw XML/JSON)
```

Extract

Parsing structure, not logic.
Converting raw HTML/XML into typed JSON.

```
flow-2-create-articles-timeline
Output: .../articles/{article_id}/feed-article.json
```

Why It Matters: If downstream steps fail, you never need to re-fetch from the external API. You simply replay from the saved raw file.

Transform and Save: The ‘Time Machine" for Data

Arbitrary Complexity, Modular Stages: Logic can be simple (merging datasets) or complex (GenAI inference), but it is always contained within modular modular stages.

The “Time Machine” Effect: Because every transform saves its output, you can replay any step without re-running the whole chain. If the matching logic changes, you don’t need to re-extract the entities.

Case Study: Provenance in GenAI with MyFeeds.ai

Solving the “Why did the AI say that?” problem.

Input:
Raw RSS Article

Transform 1:
Entity Graph
(JSON)

Transform 2:
Persona Graph
(JSON)

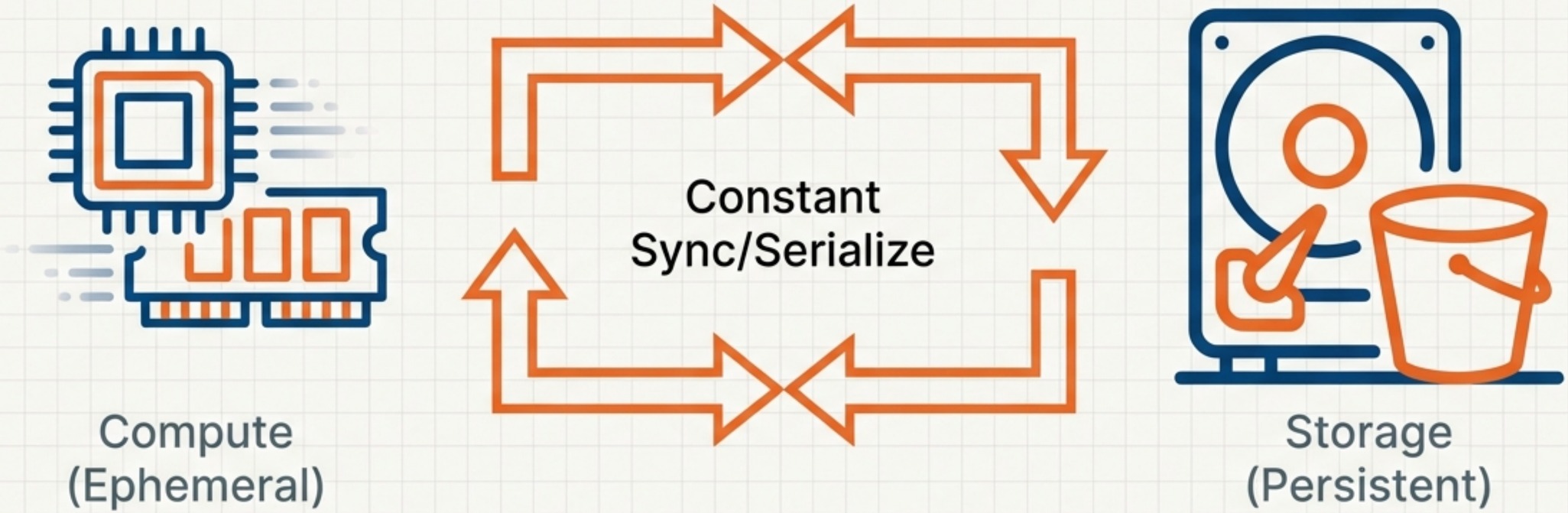
Transform 3:
Match Logic

Output:
Personalised
Summary

Traceability

Every recommendation can be explained by tracing the chain of JSON outputs. We can prove the article was shown because ‘Interest A” matched ‘Topic B’ in the saved intermediate files.

The Memory-FS Hybrid: Speed of RAM, Safety of Disk

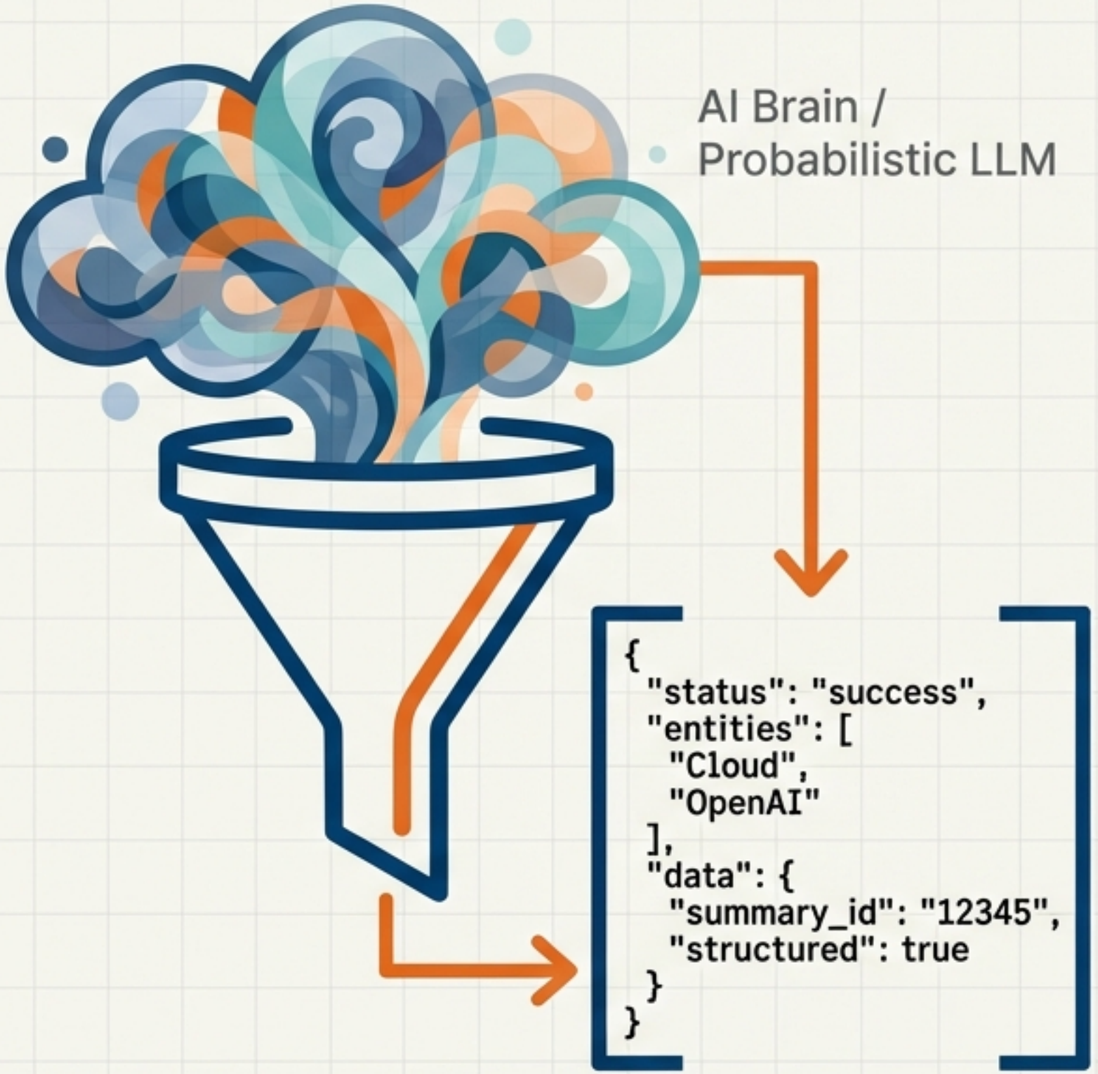


Source: MGraph-AI (Memory-First Graph Database)

Mechanism: Graphs are kept **in-memory** during computation for maximum speed. Every update is immediately serialized to JSON on the file system.

Key Benefit: Getting the performance of an in-memory database with the reliability of persistent storage.

Enforcing Determinism in the Age of AI



The Challenge: AI is probabilistic; Engineering pipelines must be deterministic.

The Solution: **Type Safety & Schema Enforcement.**

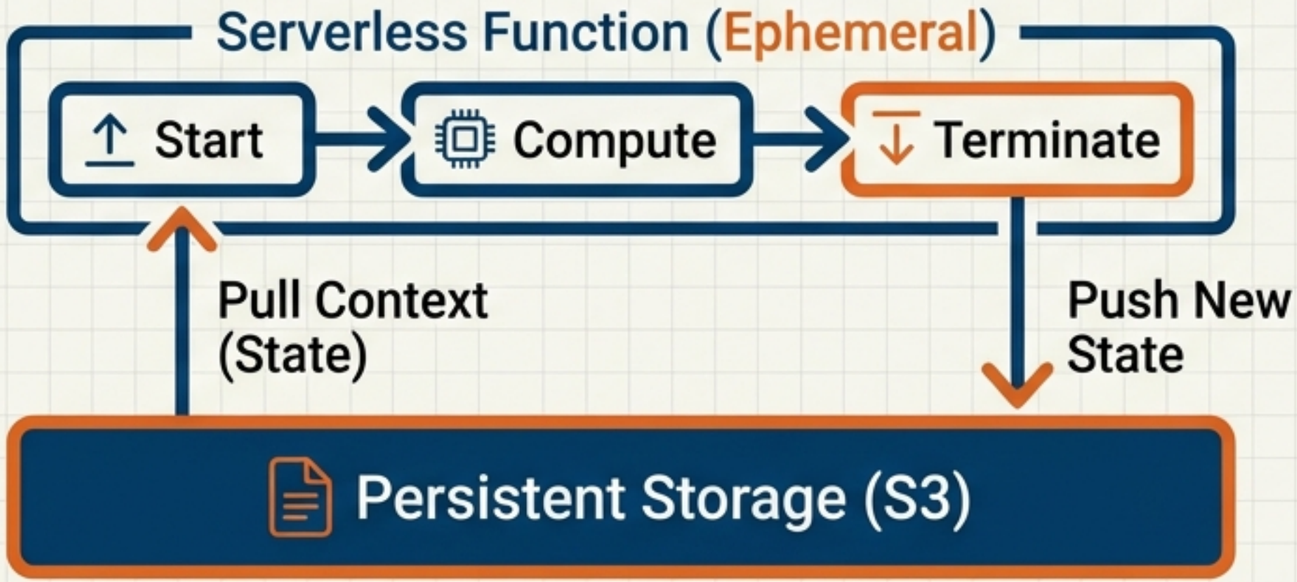
Technique: Use frameworks like OSBot-Utils Type_Safe to define Python data classes that mirror JSON structure. The LLM is prompted to fill a specific JSON schema, turning ‘creative writing’ into ‘structured data entry’.

Outcome: Every LLM stage outputs a structured JSON file. If the schema doesn’t match, the pipeline fails fast.

Case Study: Serverless Ephemerality at The Cyber Boardroom

Context: Athena, an AI-powered advisor for board executives.

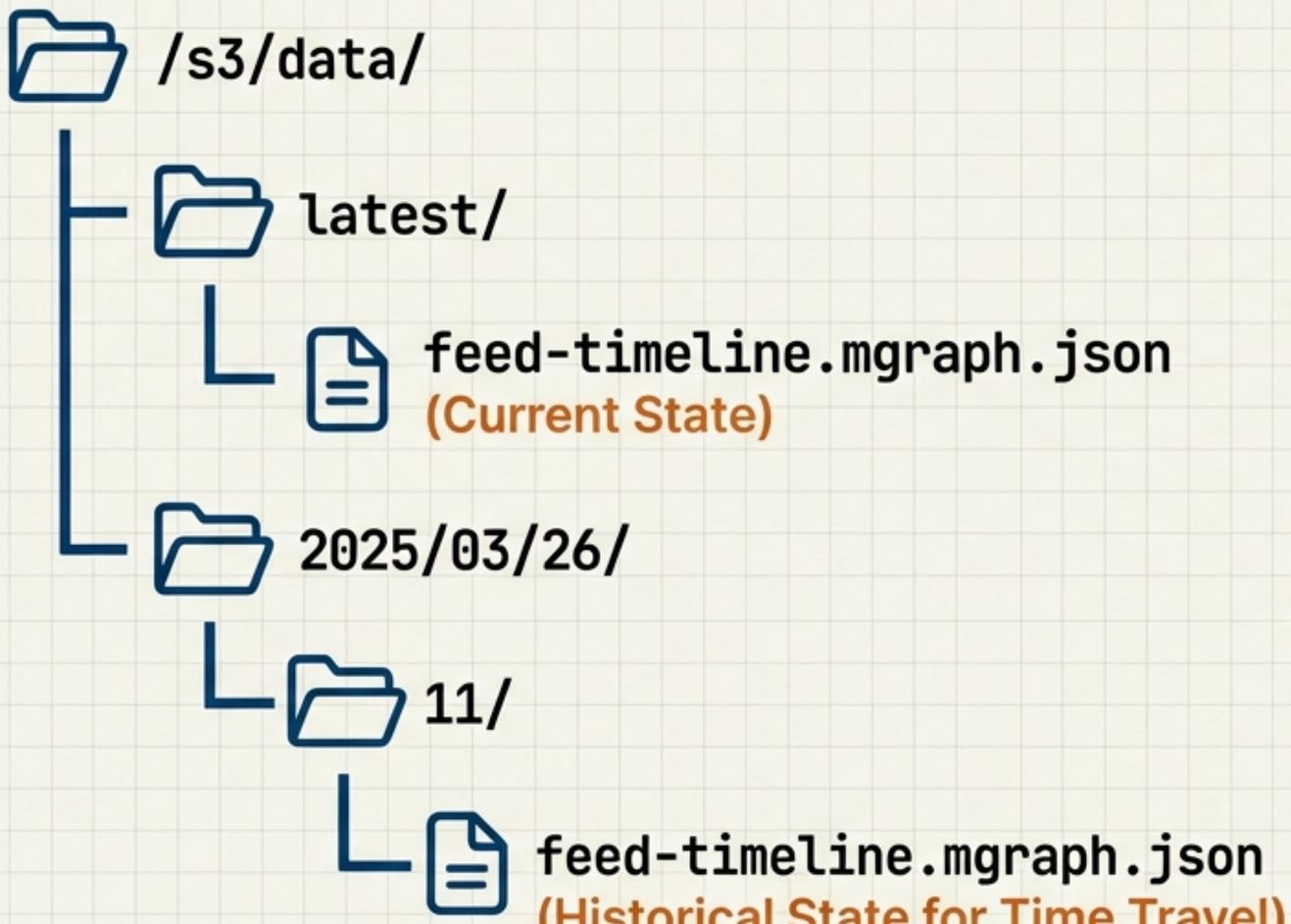
Architecture: Serverless functions spin up, compute, and terminate. No persistent process holds state.



The Ephemerality Principle: Compute dies; data survives.

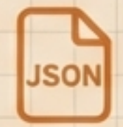
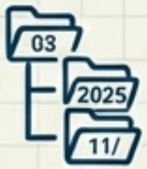
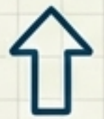
Discipline: Because memory is wiped after execution, all state **must* be saved to storage. This enables **enables infinite scaling** of bots without database bottlenecks.

Implementation: The File System is the Database



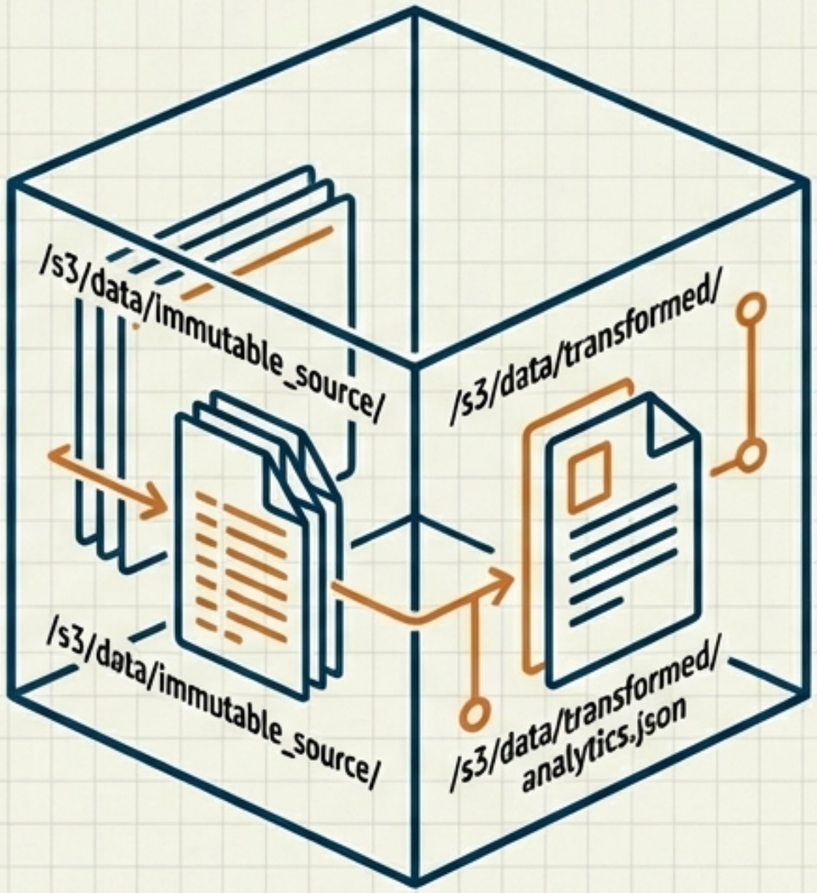
- **Strategy:** Treat S3 or Disk as your primary database.
- **Versioning:** Use timestamped folders. Do not overwrite; append and reference.
- **Minimum Viable Propagation:** Only process the delta. If a new file appears in the '11:00' folder, trigger the pipeline *only* for that file.

LETS vs. Traditional ETL: The Trade-Offs

Feature	Traditional ETL	LETS Architecture
Debugging	Log parsing, detective work.	 Open the file. Inspect the JSON.
Versioning	Often non-existent for intermediate data.	 Built-in via file system (Time Travel).
AI Suitability	Black Box (Risk of hallucinations).	 Glass Box (Explainable lineage).
Cost	Lower storage cost.	 Higher storage usage (cheap), higher I/O.

Verdict: LETS accepts higher storage usage in exchange for massive gains in **resilience** and **transparency**.

The “White Box” Future



- LETS brings software engineering rigour to the ungoverned world of data and AI.
- **Load:** The immutable start.
- **Extract:** The structured foundation.
- **Transform:** The deterministic logic.
- **Save:** The permanent record.

Stop building black boxes. Start building pipelines you can Load, Explain, Trust, and Secure.