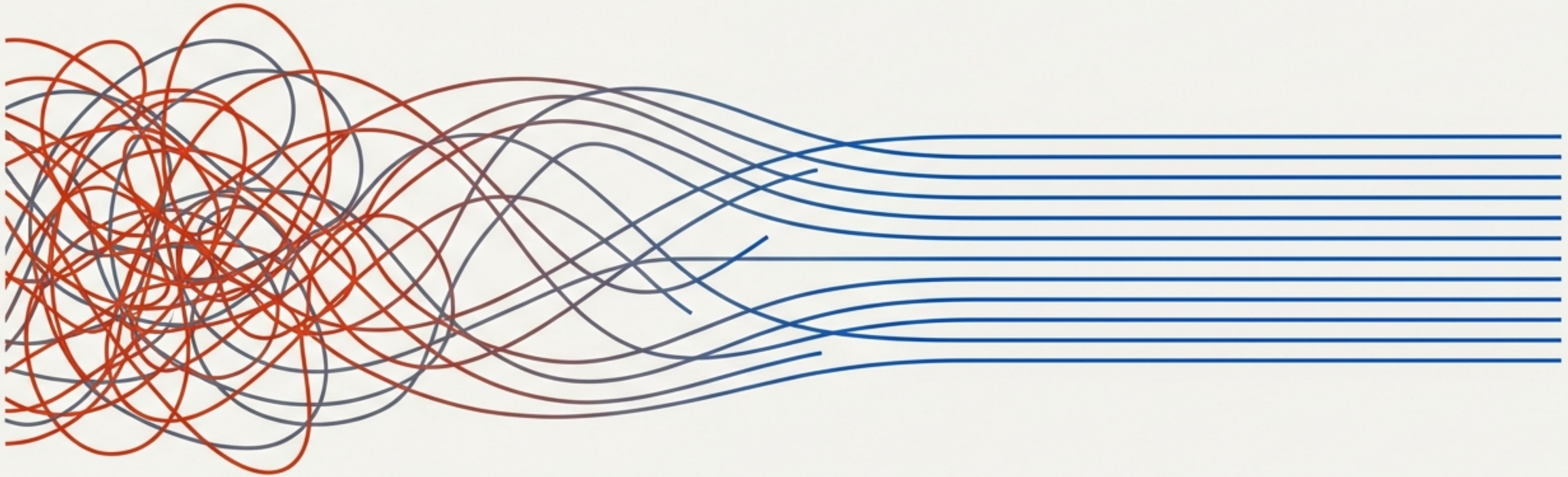
From Context Chaos to Code Cohesion

Engineering a Deterministic Snapshot for Reliable LLM-Powered Consolidation

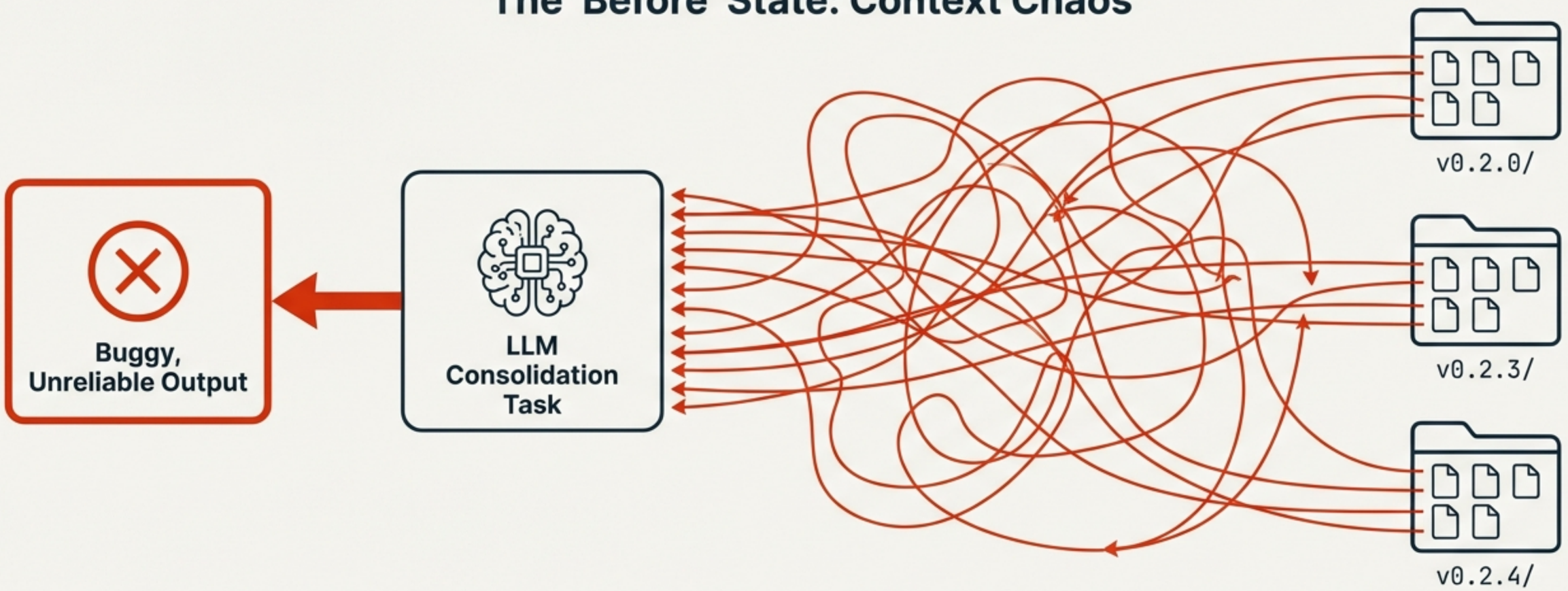


The IFD Snapshot Generator | v1.4.10

The Problem: LLM Merges Are Failing Due to Context Overload

When consolidating IFD minor versions (e.g., v1.1.x to v1.2.0), LLMs get overwhelmed by irrelevant context from scattered files, producing buggy outputs that **destroy developer trust**.

The ‘Before’ State: Context Chaos



The Core Principle: Zero LLM Involvement

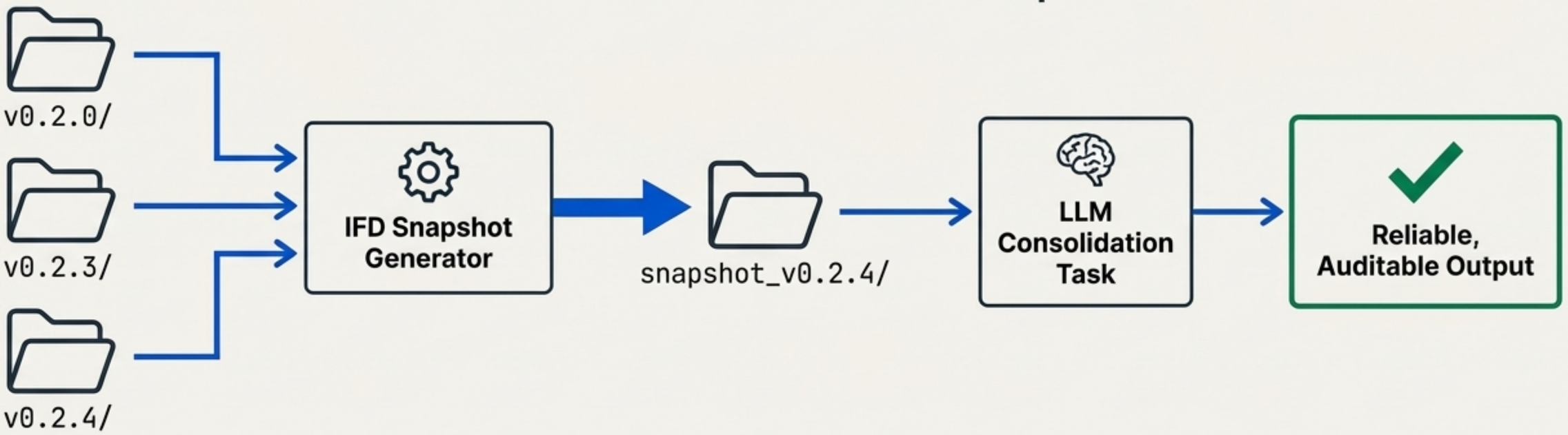
The Snapshot Generator is a purely **programmatic transformation**. The output is deterministic, auditable, and built on verifiable logic. Same input, identical snapshot. Every time.

The Solution: An Intermediate Snapshot to Isolate and Organise

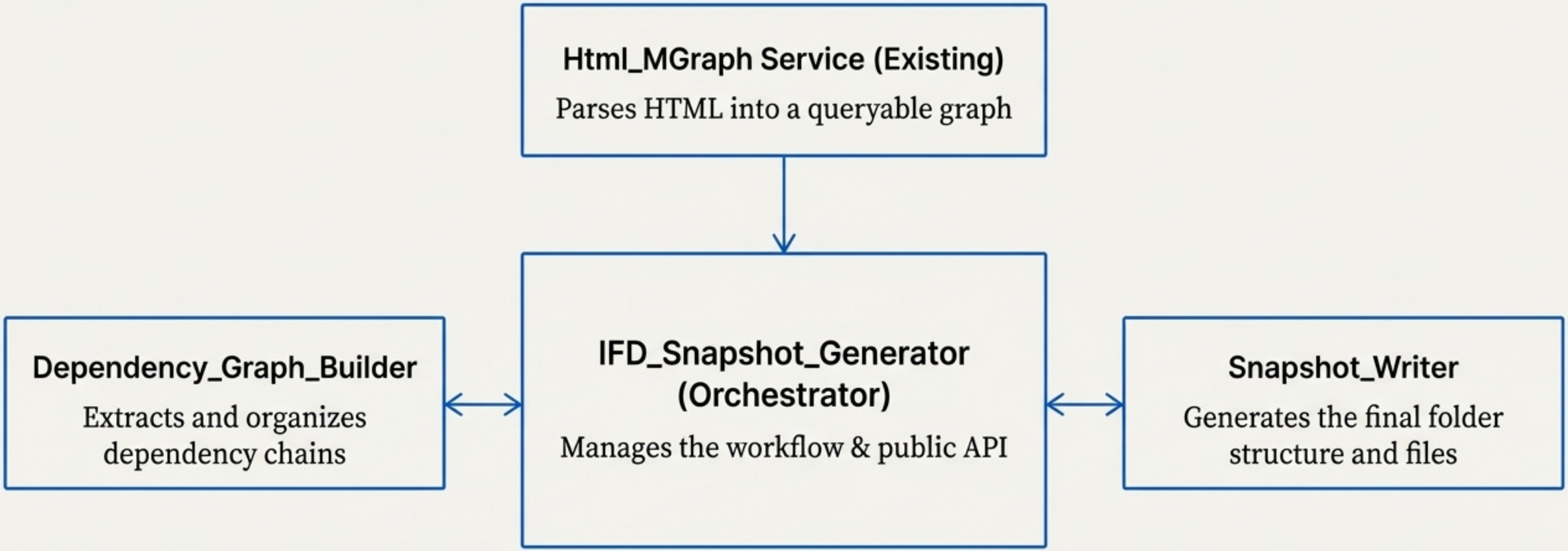
We introduce a new step: an automated generator that programmatically creates a self-contained "snapshot" of a specific IFD version.

1. **Discover:** Parses the HTML entry point to find all dependencies.
2. **Extract:** Copies only the required files from across version folders.
3. **Package:** Organises them into a single, clean folder with a manifest.

The 'After' State: A Self-Contained Snapshot

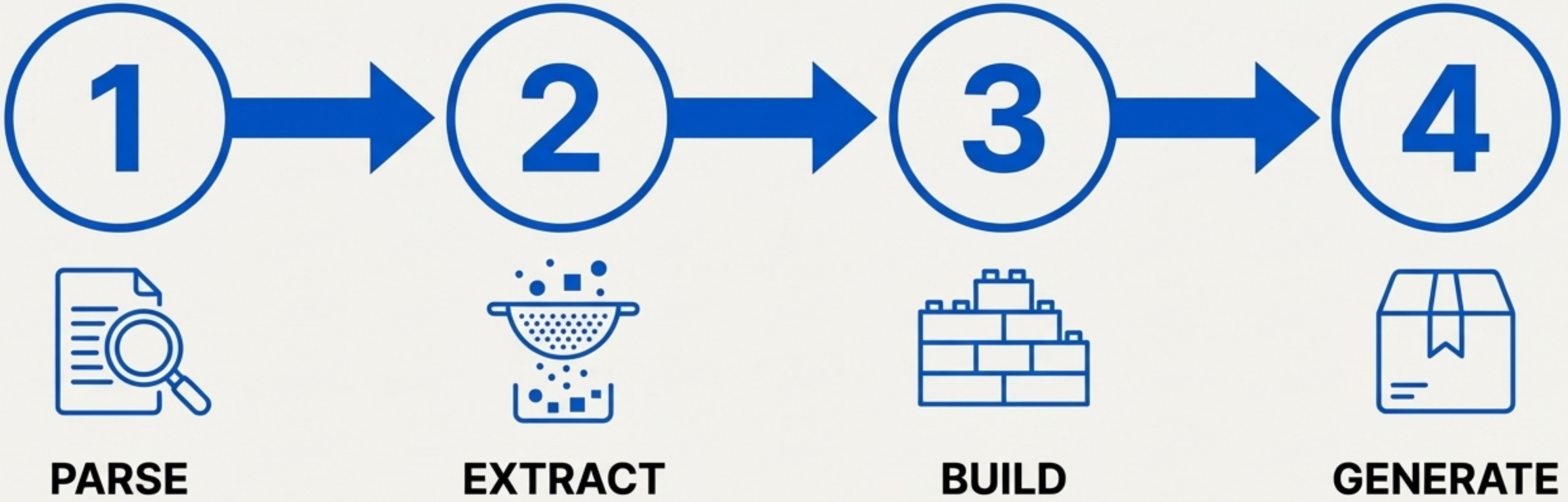


Solution Architecture: A Clear Division of Responsibilities



Dependencies: MGraph-DB v1.10.6+, Html_MGraph Service v1.4.0+

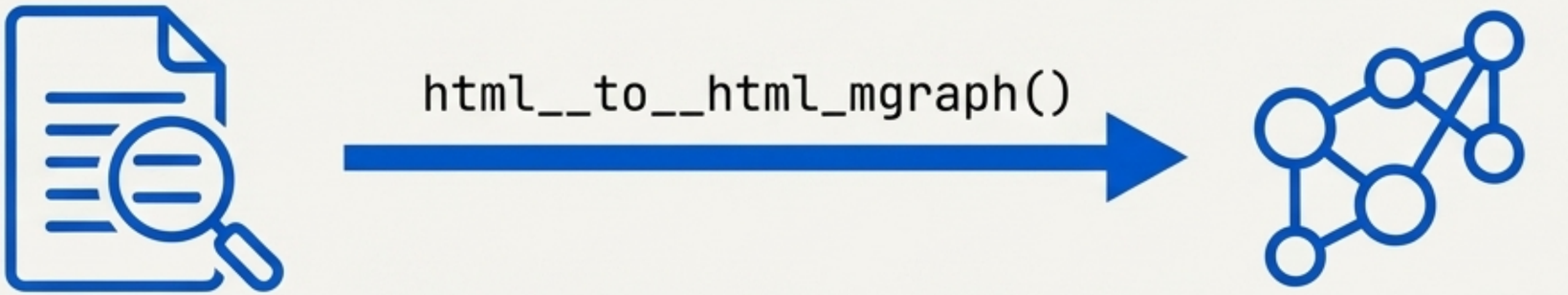
The Process Journey: A 4-Phase Transformation Pipeline



We will now walk through how an HTML entry point is systematically transformed into a complete, self-contained snapshot.

Phase 1: Parse the HTML into a Queryable Graph

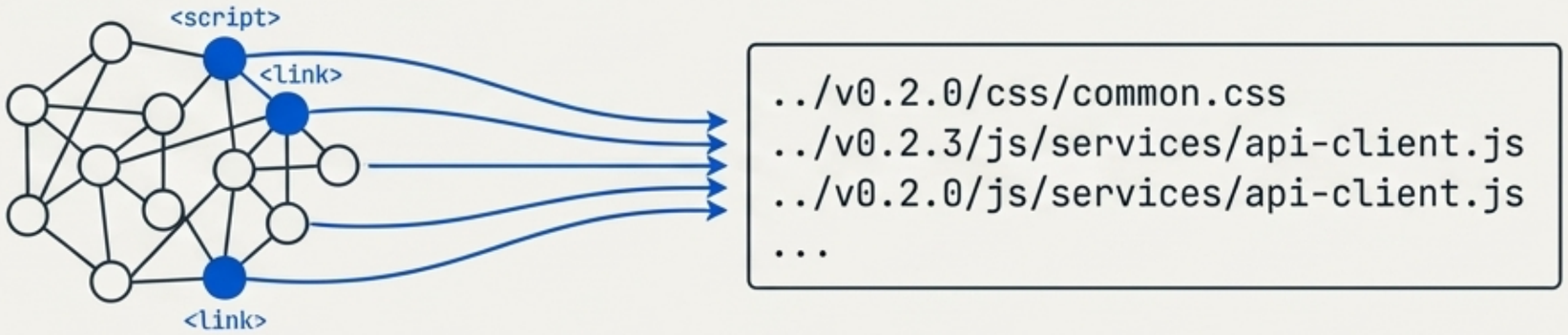
The process begins by taking the HTML entry point and converting it into an Html_MGraph document. This transforms the flat text file into a structured graph, making it possible to programmatically query its contents and relationships.



Item	Description
Input	Path to the HTML entry point file.
Key Operation	`html__to__html_mgraph()`
Output	An `Html_MGraph` document object.

Phase 2: Extract a Raw List of All Dependencies

With the HTML parsed into a graph, we can now query it to find all external resources. The system specifically targets `<link>` (for CSS) and `<script>` (for JavaScript) elements to discover every dependency required by the page.



Item	Description
Input	The `Html_MGraph` document from Phase 1.
Key Operation	Query for all `<link>` and `<script>` elements.
Output	A raw, unordered list of dependency file paths.

Phase 3: Build Organised ‘Load Chains’

The raw dependency list is processed to understand its structure. Files are grouped by their logical name (e.g., `api-client.js`), and the system detects the ‘base’ file versus subsequent ‘surgical override’ files. This creates ‘load chains’ that preserve the correct application order.

```
../v0.2.0/css/common.css
../v0.2.3/js/services/api-client.js
../v0.2.0/js/services/api-client.js
...
```

`api-client.js` Chain

`v0.2.0/.../api-client.js` — Base
`v0.2.3/.../api-client.js` — Surgical

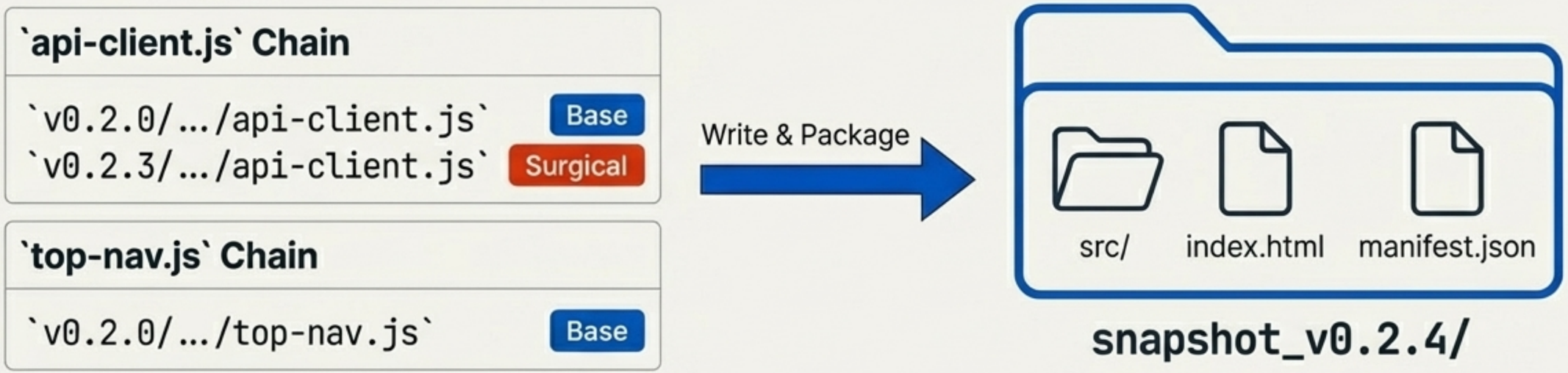
`top-nav.js` Chain

`v0.2.0/.../top-nav.js` — Base

Item	Description
Input	The raw dependency list from Phase 2.
Key Operation	Group by logical name; detect base vs. surgical.
Output	A structured dictionary of organised load chains.

Phase 4: Generate the Self-Contained Snapshot

In the final phase, the organised load chains are used to construct the output. All necessary files are copied into a new, clean directory structure, the original HTML is transformed to point to these new file paths, and a comprehensive manifest file is written.



Item	Description
Input	The organised load chains from Phase 3.
Key Operations	Copy files, transform HTML paths, write manifest.
Output	A complete, self-contained snapshot folder.

A Look Inside: Versioned Naming for Clarity and Provenance

To prevent naming conflicts and embed version information directly into the file system, files are renamed using a clear, filesystem-safe convention. The original directory structure is mirrored, but with version folders removed.

Delimiter `__`

Pattern `[logical_name]__[version].[extension]`

Original Path	Snapshot Path
`v0.2.0/css/common.css`	`src/css/common__v0.2.0.css`
`v0.2.4/css/maximize.css`	`src/css/maximize__v0.2.4.css`
`v0.2.0/js/services/api-client.js`	`src/js/services/api-client__v0.2.0.js`
`v0.2.3/js/services/api-client.js`	`src/js/services/api-client__v0.2.3.js`

The Source of Truth: The `manifest.json` File

The manifest is the brain of the snapshot. It's a machine-readable JSON file that documents everything about the build, providing a complete audit trail.

- **Preserves Load Order**: Explicitly defines the correct order for all CSS and JS files.
- **Documents Provenance**: Records the original path of every file.
- **Ensures Integrity**: Contains a SHA256 hash for every file to verify its contents.

```
{
  "metadata": { ... },
  "load_order_css": [
    "src/css/common__v0.2.0.css",
    "src/css/maximize__v0.2.4.css"
  ],
  "load_order_js": [ ... ],
  "files": {
    "src/css/common__v0.2.0.css": {
      "logical_name": "common.css",
      "original_path": "v0.2.0/css/common.css'
      "version": "v0.2.0",
      "file_type": "css",
      "sha256": "a1b2..."
    },
    ...
  }
}
```

Engineered for Reliability: Testing and Validation

The generator is built with a comprehensive testing strategy to ensure its output is always correct and to handle potential issues gracefully.

Comprehensive Testing Suite

- **Unit Tests:** Verify individual component logic (e.g., version extraction, path resolution).
- **Integration Tests:** Ensure the 4-phase pipeline works end-to-end.

The Round-Trip Test

A critical test that generates a snapshot and then validates its integrity against the manifest, guaranteeing correctness.

Built-in Validation

- Pre-generation checks for missing files or invalid paths.
- Post-generation `validate_snapshot()` method to verify an existing snapshot's integrity using file hashes.

Putting It to Work: API and CLI Usage

The generator is designed for easy integration into existing workflows, accessible as a Python library or a command-line tool.

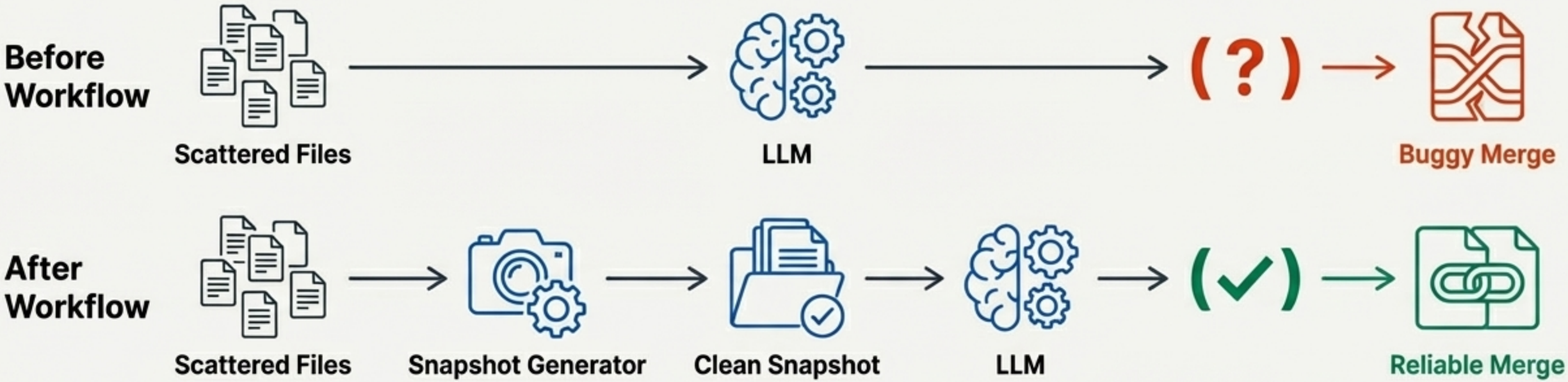
Python API Usage

```
from ifd_snapshot_generator import
IFD_Snapshot_Generator

generator = IFD_Snapshot_Generator(...)
manifest = generator.generate() # Full
run

# Dry run (manifest only)
manifest_only = generator.generate_mani
fest_only()
```

Command Line Interface (CLI)

```
# Generate a full snapshot
$ ifd-snapshot-generator --entry-point
v0.2.4/index.html --output-dir
./snapshot

# Run a dry run to get the manifest
$ ifd-snapshot-generator --entry-point
v0.2.4/index.html --manifest-only
```

The Result: A Deterministic Foundation for Consolidation



Summary of Key Benefits:

1. **Deterministic Output:** Guarantees identical snapshots from identical inputs.
2. **Complete Dependency Resolution**: Captures every file needed to run the version.
3. **Preserved Load Order:** The manifest acts as the source of truth for asset loading.
4. **Full Audit Trail:** Complete file provenance and integrity hashes in `manifest.json`.
5. **Self-Contained & Verifiable**: The snapshot runs with zero external dependencies and can be hash-verified.

The IFD Snapshot Generator restores trust and reliability to the consolidation workflow.