TOWARD A TRANSPARENT, AUDITABLE, AND DISTRIBUTED ARCHITECTURE FOR LLM TASKS USING W3C LINKED DATA NOTIFICATIONS AND REMOTE UV SCRIPTS

Géraldine Geoffroy
SmartBibl.IA Solutions
geraldine.geoffroy@smartbiblia.fr
Orcid ID: 0000-0002-2986-3083

ABSTRACT

This paper proposes a novel architecture for distributed, traceable, and event-driven execution of LLM-related tasks by combining W3C Linked Data Notifications (LDN) with remote Python scripts executed via `uv run`. This architecture enables any AI task -especially inference- to be executed locally with no software installation, traced via interoperable notifications, and archived with full provenance metadata (e.g., models, parameters, etc.). To achieve this, the system leverages LDN as a semantic pub-sub orchestration layer, combined with `uv`-based scripts as reproducible, stateless microservices. We demonstrate the value of this architecture for building transparent, auditable, and distributed Large Language Model (LLM) inference workflows with three working proof-of-concepts : (1) a basic semantically-notified inference where notifications populate a register of evidences for transparency, (2) a Retrieval-Augmented Generation (RAG) pipeline triggered by `Create` events and executed through script-based stages, and (3) a distributed inference setup where task-specific SLM agents independently process jobs and respond via `Announce` messages. Each stage archive full provenance metadata (model version, script SHA, parameters, runtime) using PROV-O, supporting reproducibility and auditability. This architecture lays the groundwork for a lightweight, decentralized, and FAIR-aligned standard for orchestrating LLM tasks.

The demo code repository is available at : https ://github.com/gegedenice/LLM-notify

1 Introduction

As LLMs become deeply integrated into scientific, industrial, and public sector workflows, there is growing concern over their opacity, reproducibility, and trustworthiness. Most LLM interactions remain centralized, opaque, untraceable, and non-reproducible : it is often impossible to know which version of a model was used, what inputs it received, and under what parameters it operated.

This issue is particularly recognized and addressed by institutions such as libraries and archives, which are deeply engaged in the open science movement and strongly advocate for the adoption of FAIR (Findable, Accessible, Interoperable, and Reusable) principles. These institutions promote practices and infrastructures that improve transparency and auditability in digital research workflows, including the management and provenance of AI and LLM output. Their involvement is crucial for enabling reproducibility and long-term stewardship, ensuring that AI-generated content and its context can be traced, verified, and reused according to open science standards for both scientific rigor and public trust in LLM-driven discoveries and applications.

Simultaneously, institutions lack flexible, low-friction tools to deploy AI capabilities without extensive software dependencies or infrastructure. We believe the need is clear for a model of inference that is **transparent**, **distributed**, **modular**, and **auditable by design**.

Recent works have addressed the need to monitor the transparency and reliability of AI generative models through several complementary approaches : Explainable AI (XAI) methods to improve the interpretability of neural networks

and make models understandable ([1] and [2]) like analyzing the internal workings of transformer models ([3]), or auditing LLM outputs by incorporating human-in-the-loop verification ([4]).

While these contributions offer valuable tools for transparency, explainability, and reliability, the present paper proposes a distinct approach centered on the full traceability of LLM-driven workflows. Our architecture prioritizes the generation, orchestration, and archiving of semantically structured, provenance-rich notifications that document every stage of AI task execution, thereby addressing the comprehensive audibility, reproducibility, and long-term stewardship of LLM outputs within distributed and federated environments.

In this paper, we propose an architecture to meet that need. Our approach combines two key components :

1. **Remote, zero-install Python scripts** run locally via `uv run`, providing modular and reproducible execution units.
2. **W3C Linked Data Notifications (LDN)** to orchestrate, trace, and archive all AI tasks via semantically structured events.

We demonstrate this architecture through three proof-of-concepts :

1. A basic Inference provider-agnostic wrapper upon OpenAI API client
2. A notification-driven Retrieval-Augmented Generation (RAG) pipeline
3. A cooperative network of specialized Small Language Model (SLM) agents performing distributed inference

2 Motivation and Problem Statement

Despite advances in LLM capabilities, their deployment remains problematic : (i) Lack of transparency. Users cannot easily audit which prompt, model, or parameters produced a given response. (ii) Installation burden. Deploying inference pipelines requires Python environments, dependencies, and often infrastructure. (iii) Centralized architectures. Most systems are centralized, hard to federate, and difficult to integrate across organizational boundaries. (iv) Poor provenance. Outputs of AI systems are rarely linked to their full execution context (scripts, models, metadata).

Our goal is to provide a lightweight, extensible, and fully traceable foundation for distributed AI workflows, particularly for inference, without introducing installation or operational overhead.

3 Proposed Architecture

Our architecture addresses the above problems by combining **remote script hosting** and **notification-driven orchestration**.

3.1 Remote hosted but locally Executable Scripts with `uv run`

uv ([5]) is a high-performance package manager written in Rust that serves as a drop-in replacement for traditional tools like pip, pip-tools, poetry and virtualenv. It is first designed for managing dependencies, installing packages, resolving and locking dependencies, and managing project-specific virtual environments to performe faster and with better conflict resolution.

`uv run` enables users to locally execute Python scripts that are hosted remotely (including scripts accessible via URLs), without requiring any manual installation of dependencies or virtual environments, thanks to metadata declared in the script via PEP 723 [1].

In effect, uv leverages the PEP 723 inline metadata to : (i) Download a Python script from a local path or URL. (ii) Parse the embedded dependency and Python version requirements. (iii) Automatically set up an isolated execution environment on-the-fly (local venv). (iv) Install all the declared dependencies, if needed, ephemeral to the script only. (v) Execute the code locally without polluting the global environment or requiring project setup.

No local manual installation, project setup, or code modification is needed, scripts are single file, PEP self-describing and the user simply runs ‘uv run https ://someurl/script.py‘ with script including inline metadata [2]. :

1. See also uvx, a complementary alias for uv tool run, designed specifically for executing Python tools or scripts in a temporary, isolated environment
2. Dependencies can alternatively be specified with the command ‘uv run –with requests –with rich script.py‘

```
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "requests",
#     "rich"
# ]
# ///
```

The script is then executed on the user's machine in a fresh, isolated environment, fully determined by its inline PEP 723 metadata. This allows safe, reproducible, zero-install and zero-code-modification execution of Python code from URLs or files, directly on the local system.

To summarize, by hosting uv script files, for example on Git repository, and pinning URLs to specific commit SHAs, users gain :

- — No installation required
- — Immutable execution environments
- — Reproducibility via SHA-pinned URLs (same result every time)
- — Input/output isolation (‘–in‘, ‘–out‘, ‘–json‘)
- — No hidden updates or breaking changes
- — Local or remote (e.g., Hugging Face Jobs) execution
- — Trust and auditability

Example :

```
uv run https://raw.githubusercontent.com/user/repo/<SHA>/summarizer.py --in doc.txt --out summary.json
```

Notice that scripts can be run locally or « bounced » to cloud workers (e.g., Hugging Face Jobs) using a `-remote` flag.

3.2 Notification-Driven Orchestration with LDN

W3C Linked Data Notifications (LDN) as formalized in [6] is a protocol for exchanging event-driven messages in a decentralized, interoperable fashion across web applications and services. LDN enables any resource (such as a dataset, article, or model output) to advertise an Inbox URL to receive notification messages via HTTP. Notifications are structured in semantic web formats standards as RDF (more often JSON-LD), providing semantic markup and extensibility.

The use of LDN draws a direct lineage from the vision advocated by Tim Berners-Lee regarding the circulation and interoperability of data on the web ([7]). Beyond its original design, LDN has been recognized and explored as a generic protocol for architecting agent-based systems in which autonomous components communicate through message exchange. Theoretical analyzes ([8]) and applied implementations such as COAR Notify Protocol ([9]) have demonstrated LDN's suitability as a foundational layer for decentralized, event-driven coordination among distributed agents, reinforcing both the architectural modularity and transparency central to the proposed orchestration approach.

3.2.1 How LDN works

LDN messages follow a standard pattern : (i) Create : new document submitted for processing. (ii) Announce : outputs of each task. (iii) Update/Offer : model updates or rerun requests

Inbox Discovery : A sender or consumer discovers whether a resource supports LDN (i.e., has an Inbox) by inspecting the HTTP Link header or the body of the resource for the « ldp :inbox » relation (e.g.

```
Link: <https://example.org/inbox/123>; rel="http://www.w3.org/ns/ldp#inbox"
```

)

Sending Notifications : A sender (either human-triggered or automated process) creates a notification message (such as a new inference request, provenance event, or task status) and posts it as a JSON-LD serialized payload to the Inbox URL using HTTP POST.

Receiving Notifications : The receiver stores the notification and responds with the URI of the created notification (201 Created).

Access and Consumption : Other applications or agents (consumers) retrieve the list of received notifications by issuing GET requests to the Inbox URL. The response is a listing (often as RDF with the ldp :contains predicate) of notification URIs. Consumers can then retrieve and process these messages, trigger workflows, or present them to users.

3.2.2 Design Principles and Benefits

Decentralization : Senders, receivers, and consumers can be entirely independent and use different technology stacks, supporting modular and federated architectures.

Interoperability : Notifications are discoverable via HTTP GET requests, semantically structured, and reusable across applications.

Persistence : Each notification has its own URI and can be referenced, archived, and further described or linked to other web resources.

Flexible Payloads : LDN does not mandate content schemas ; applications define the meaning and structure of notifications.

Access Control and Validation : Receivers may restrict who can post/read notifications (e.g., via authorization) and may advertise data shapes (e.g., SHACL) for validation.

3.2.3 Relevance for Transparent LLM Orchestration

By providing a standardized and semantic mechanism to publish, receive and archive notifications (e.g., `Create`, `Announce`) about resources or events, LDN can act in our architecture as a semantic web-native event-system for orchestrating LLM and AI tasks : (i) **Ask creation**. Tasks or jobs (e.g., inference, evaluation) are announced via LDN `Create` messages to Inbox endpoints. (ii) **Result/Status Sharing**. Consumers respond by executing relevant uv scripts, and after task completion, results (with full provenance and metadata) are emitted and shared via Announce messages carrying both payloads and metadata, making the process traceable and auditable. (iii) **Real-time Processing**. Consumers subscribe to inboxes and react to events in real-time, enabling distributed workflows and modular pipeline construction (iv) **Archiving and FAIRness**. All notifications (including provenance traces) are persistent, URI-addressable, and structured for reuse, supporting robust audit trails and FAIR data practices.

3.3 Provenance and Audit via PROV-O

The W3C PROV Ontology (PROV-O) as described in as described in [10] is a standard for representing, exchanging, and interlinking provenance information on the Web. Provenance, in this context, refers to metadata that describes the origins, history, and context of digital resources, enabling assessments of data quality, reliability, and trustworthiness.

3.3.1 Core concepts

PROV-O organizes provenance descriptions around three primary classes : (i) **Entity (prov :Entity)**. Any physical, digital, conceptual, or other item with a fixed aspect. In AI/LLM workflows, this might be an input file, a model weight file, a dataset, an output result, or a log artifact. (ii) **Activity (prov :Activity)**. An action or process that interacts with entities, usually over a span of time e.g., running a Python script, training a model, or performing inference. (iii) **Agent (prov :Agent)**. Someone or something responsible for an activity : this can describe a human, institution, software process, or even an automated agent in orchestrated workflows.

Activities use entities as inputs (prov :used), generate new entities as outputs (prov :wasGeneratedBy), and are associated with agents (prov :wasAssociatedWith). Agents can also be attributed as creators of entities (prov :wasAttributedTo). Activities have rich temporal features (e.g., prov :startedAtTime, prov :endedAtTime), which create auditable execution trails.

3.3.2 Integraion in Distributed LLM Auditing

For every task in the architecture, a machine-readable provenance trace is automatically generated using PROV-O and included in the LDN message :

— Who executed the task (`prov:wasAssociatedWith`)
— What input was used (`prov:used`)
— What parameters and models were applied (`prov:parameters`)
— When execution occurred (`prov:startedAtTime`, `prov:endedAtTime`)
— What environment was used (`uv` version, dependency hash)

— Where output is stored and its content hash

LDN including PROV-O vocabulary thus provides a universal, standards-based backbone for semantic, transparent, and auditable orchestration of modular AI workflows, aligned with open science and FAIR principles by enabling scientific reproducibility [3], regulatory audit, and traceable inferencing chains.

4 Implementation Examples

4.1 Use Case 1 : Basic LLM Inference with semantic Notifications

4.1.1 Overview

We first present a foundational proof-of-concept that establishes the core architectural principles using basic LLM inference. This minimal example shows how any OpenAI-compatible LLM inference task can be triggered, orchestrated, and traced entirely through W3C Linked Data Notifications, with zero local installation requirements.

See this repo for demo : https ://github.com/gegedenice/LLM-notify/tree/main/inference-notify-demo

The inference demo consists of four lightweight components :

— inbox_server.py : Simple LDN inbox server that receives notifications and serves results
— send_ldn.py : Client that constructs detailed inference job notifications
— poll_and_run.py : Orchestrator that polls the inbox and executes remote inference scripts
— inference.py : Universal OpenAI-compatible client supporting multiple providers

4.1.2 Workflow Architecture with example messages

Job Submission : A client uses send_ldn.py to generate a Create notification that contains all inference parameters.

```
  "@context": "https://www.w3.org/ns/activitystreams",
  "id": "urn:uuid:bde3e011-8769-4ff5-926e-562476a96c9e",
  "type": "Create",
  "actor": "https://smartbiblia.fr/agents/cli-user-inference",
  "object": {
    "provider": "groq",
    "model": "moonshotai/kimi-k2-instruct-0905",
    "user_prompt": "Explain reranking in RAG.",
    "system_prompt": "You are an expert in IA systems.",
    "temperature": 0.1,
    "max_tokens": 500
  },
  "instrument": {
    "type": "Service",
    "action": "infer"
  }
}
```

Notification Reception : The LDN inbox server (inbox_server.py) receives and stores the notification as JSON in a local state/inbox/ directory.

Job Orchestration : The orchestrator (poll_and_run.py) continuously polls the inbox, detects new inference jobs, and dynamically builds uv run commands :

```
uv run https://raw.githubusercontent.com/<user>/LLM-notify/<SHA>/inference-notify-demo/inference.py \
  --provider groq \
  --model moonshotai/kimi-k2-instruct-0905 \
  --user-prompt "Explain reranking in RAG." \
  --system-prompt "You are an expert in IA systems." \
  --temperature 0.1 \
  --max-tokens 500
```

3. Reproducibility here refers to the ability to reproduce workflows and inference chains -not necessarily the exact outputs of inference-due to the inherent non-determinism of large language model behavior.

Remote Execution : The inference.py script executes with zero installation via uv run, supporting multiple LLM providers : OpenAI, Groq, Ollama, and HuggingFace. The script includes inline PEP 723 metadata specifying Python version and dependencies.

Result Publication : The user receives the LLM response :

```
Orchestrator started. Polling inbox at http://localhost:8080/inbox...

Received new inference job: 576c84413a0e1346a6a664008edaccd704bef558

Running command: uv run https://raw.githubusercontent.com/<user>/LLM-notify/<SHA>/inference-notify-demo.

Reranking in RAG is a second-stage reordering of the documents that the first-stage retriever returned.
Its goal is simple: move the chunks that are actually useful for answering the question to the top of th
Why we need it
The initial retriever (dense, sparse, or hybrid) is optimized for recall, not precision.
The generator is sensitive to order: the first 1-3 passages dominate what it writes.
The context window is small and expensive; we want to fill it with the best material.
How it works
Retrieve: Top-k (e.g., 30-100) passages are fetched quickly.
Rerank: A slower but smarter model scores each passage on the query-passage pair.
Truncate: Keep the top-n (e.g., 3-5) passages and feed them to the generator.
Typical rerankers
Cross-encoder BERT: Concatenate query and passage, output relevance score
...
```

And upon completion, the orchestrator saves the LLM response to state/inference-result-uuid.txt and posts an Announce notification :


```
{
  "@context": ["https://www.w3.org/ns/activitystreams", "https://www.w3.org/ns/prov#"],
  "type": "Announce",
  "actor": "https://smartbiblia.fr/actors/inference-runner",
  "object": {
    "type": "Document",
    "id": "urn:smartbiblia:state:inference-result-abc123.txt",
    "name": "inference-result-abc123.txt",
    "url": "http://localhost:8080/state/inference-result-abc123.txt"
  },
  "prov:wasGeneratedBy": "urn:uuid:original-create-notification-id"
}
```


4.1.3 Key Architectural Benefits

Zero-Installation Reproducibility Every script runs via uv run with no local dependencies. For production stability, URLs can be pinned to specific commit SHAs.

Complete Parameter Auditability The original Create notification preserves every inference parameter (model, provider, temperature, prompts), enabling full reproducibility and compliance auditing.

Provider-Agnostic Design The same notification structure works across OpenAI, Groq, Ollama, HuggingFace or any OpenAI server-like compatible (local or cloud) provider, with automatic API key inference from environment variables.

Asynchronous Decoupling The client submits jobs and disconnects. Results are available via HTTP once processing completes, supporting scalable batch workflows.

Standards-Based Federation Organizations can deploy their own LDN inboxes and inference runners, creating a federated network of AI capabilities without vendor lock-in.

4.2 Use Case 2 : Notification-Driven RAG Pipeline

4.2.1 Overview

To further demonstrate the notification-driven orchestration using LDN without complexity, we introduce a very basic RAG demonstration example (minimal chunking strategy, no vector store database, no reranking model, only semantice retrieval, no cache layer...).

See this repo for demo : https ://github.com/gegedenice/LLM-notify/tree/main/RAG-notify-demo

The proposed pipeline is : a new document triggers a `Create` notification and a runner script (poll_and_run.py) executes a chain of `uv`-based tasks assorted of `Announce` messages with full provenance emitted and archived on Inbox at each step :

1. splitter.py : splits text into 900 character chunks
2. embedder.py : computes vector embeddings using MiniLM
3. indexer.py : stores the vectors in a simple JSONL index
4. query.py : retrieves top-K similar chunks and builds the final prompt

4.2.2 Example Messages

Create Notification : Ingest document into pipeline

```
{
  "@context": ["https://www.w3.org/ns/activitystreams"],
  "type": "Create",
  "actor": "https://smartbiblia.fr/actors/publisher",
  "object": {
    "type": "Document",
    "id": "https://example.org/docs/sample.txt",
    "mediaType": "text/plain",
    "name": "Sample Document",
    "url": "https://example.org/docs/sample.txt"
  },
  "target": "https://pipeline.smartbiblia.fr/inbox",
  "instrument": {
    "type": "Service",
    "name": "RAG Indexing Pipeline",
    "action": "index"
  }
}
```

Announce from splitter.py : Chunks from Splitter

```
{
  "@context": [
    "https://www.w3.org/ns/activitystreams",
    "https://www.w3.org/ns/prov#",
    "https://schema.org"
  ],
  "type": "Announce",
  "actor": "https://smartbiblia.fr/agents/splitter",
  "object": {
    "type": "schema:Dataset",
    "id": "urn:smartbiblia:state:chunks:sample",
    "name": "Text Chunks from Sample.txt",
    "url": "file://./state/chunks.jsonl",
    "mediaType": "application/jsonl"
  },
  "prov:wasGeneratedBy": {
    "type": "prov:Activity",
```

```
    "prov:used": "https://example.org/docs/sample.txt",
    "prov:wasAssociatedWith": "splitter.py@2e9fc12",
    "prov:startedAtTime": "2025-10-01T09:00:12Z",
    "prov:endedAtTime": "2025-10-01T09:00:13Z",
    "prov:parameters": {
      "chunk_size": 900
    },
    "prov:generated": {
      "outputHash": "sha256:3adf...1a9",
      "format": "jsonl",
      "executionEnv": "uv@0.1.30"
    }
  }
}
```


Announce from embedder.py : embeddings generation


```
{
  "@context": [
    "https://www.w3.org/ns/activitystreams",
    "https://www.w3.org/ns/prov#",
    "https://schema.org"
  ],
  "type": "Announce",
  "actor": "https://smartbiblia.fr/agents/embedder",
  "object": {
    "type": "schema:Dataset",
    "id": "urn:smartbiblia:state:embeddings:sample",
    "name": "Embeddings for Sample.txt",
    "url": "file://./state/embeddings.jsonl",
    "mediaType": "application/jsonl"
  },
  "prov:wasGeneratedBy": {
    "type": "prov:Activity",
    "prov:used": "file://./state/chunks.jsonl",
    "prov:wasAssociatedWith": "embedder.py@1baf37d",
    "prov:startedAtTime": "2025-10-01T09:00:14Z",
    "prov:endedAtTime": "2025-10-01T09:00:17Z",
    "prov:parameters": {
      "model": "sentence-transformers/all-MiniLM-L6-v2",
      "normalize_embeddings": true
    },
    "prov:usedSoftware": {
      "type": "schema:SoftwareApplication",
      "name": "sentence-transformers",
      "version": "3.0.0"
    },
    "prov:generated": {
      "outputHash": "sha256:fb79...ce4",
      "format": "jsonl",
      "executionEnv": "uv@0.1.30"
    }
  }
}
```


Announce from indexer.py


```
{
  "@context": [
    "https://www.w3.org/ns/activitystreams",
```

```
    "https://www.w3.org/ns/prov#",
    "https://schema.org"
  ],
  "type": "Announce",
  "actor": "https://smartbiblia.fr/agents/indexer",
  "object": {
    "type": "schema:Dataset",
    "id": "urn:smartbiblia:state:index:sample",
    "name": "Vector Index for Sample.txt",
    "url": "file://./state/index.jsonl",
    "mediaType": "application/json"
  },
  "prov:wasGeneratedBy": {
    "type": "prov:Activity",
    "prov:used": "file://./state/embeddings.jsonl",
    "prov:wasAssociatedWith": "indexer.py@e93ba1f",
    "prov:startedAtTime": "2025-10-01T09:00:18Z",
    "prov:endedAtTime": "2025-10-01T09:00:19Z",
    "prov:parameters": {
      "index_format": "jsonl",
      "dimension": 384
    },
    "prov:generated": {
      "outputHash": "sha256:9c2f...d41",
      "format": "json",
      "executionEnv": "uv@0.1.30"
    }
  }
}
```


Announce from query.py (with retrieved chunks)


```
{
  "@context": [
    "https://www.w3.org/ns/activitystreams",
    "https://www.w3.org/ns/prov#",
    "https://schema.org"
  ],
  "type": "Announce",
  "actor": "https://smartbiblia.fr/agents/query-runner",
  "object": {
    "type": "schema:CreativeWork",
    "id": "urn:smartbiblia:outputs:response:query123",
    "name": "LLM Response to Query",
    "url": "file://./outputs/response_query123.json",
    "text": "The document discusses the environmental impacts of urbanization...",
    "mediaType": "text/plain"
  },
  "prov:wasGeneratedBy": {
    "type": "prov:Activity",
    "prov:used": [
      "file://./state/index.jsonl",
      "query:text:What are the main themes in the document?"
    ],
    "prov:wasAssociatedWith": "query.py@da8b12e",
    "prov:parameters": {
      "model": "gpt-4o-mini",
      "top_k": 5,
      "embedding_model": "sentence-transformers/all-MiniLM-L6-v2",
      "prompt_template": "context-injection-rag-v1"
```

```
    },
    "prov:qualifiedUsage": [
      {
        "prov:entity": "chunk-003",
        "prov:role": "retrieved-context",
        "prov:location": "file://./state/chunks.jsonl",
        "prov:value": "Urbanization affects biodiversity through habitat loss..."
      },
      {
        "prov:entity": "chunk-007",
        "prov:role": "retrieved-context",
        "prov:location": "file://./state/chunks.jsonl",
        "prov:value": "The report emphasizes sustainable development goals (SDGs)..."
      }
    ],
    "prov:startedAtTime": "2025-10-01T09:05:01Z",
    "prov:endedAtTime": "2025-10-01T09:05:02Z",
    "prov:generated": {
      "outputHash": "sha256:fa88...b4c",
      "format": "plain/text",
      "executionEnv": "uv@0.1.30"
    }
  }
}
```


4.3 Use Case 3 : Distributed Inference via Specialized SLMs and Federated Micro-Agent Networks

Furthermore, the semantic notification-driven RAG pipeline (Use Case 2) and auditable proof-of-concept on LLM inference (Use Case 1), we can imagine multiple logical extensions, like federated networks of specialized discrete micro-model agents deployed across organizations where each agent runs a Small Language Model (SLM) designed for a specific, well-defined academic task, such and communicate over each other only via notifications.

In this architecture, instead of a single monolithic LLM serving all requests, each agent subscribes to specific task notifications published via the W3C LDN standard, processes only the relevant incoming data, and publishes its results with full provenance and audit metadata utilizing the W3C PROV-O standard.

4.3.1 Mechanism of Operation

(i) Individual agent (could be research labs, libraries, academic departments, etc.) operate dedicated SLM agents, e.g., a summarization agent for scientific abstracts, an entity extraction agent for digital humanities corpora, a classifier for research domain assignments.

(ii) Agents communicate only via notifications ; there is no direct inter-agent coupling, promoting strong modularity and institutional autonomy.

(iii) Each microagent independently :

- Subscribes to relevant task types (e.g., « summarization », « entity-extraction »)
- Downloads and processes only the minimal needed input data
- Executes its assigned function with pinned scripts for reproducibility
- Publishes its output and associated provenance metadata as an Announce notification to institutional or shared inboxes.

This distributed design renders the network federated (no central controller required), auditable (every inference and its provenance is traceable), and scalable across institutions (each organization can deploy new micro-agents as needed, interconnecting with others through common notification protocols).

4.3.2 Example workflow

1. A faculty sends a document to the institutional LDN inbox, requesting both a summary and an entity list.
2. The inbox emits Create notifications for « summarization » and « entity-extraction ».

3. The summarizer.py agent (runned by another faculty) receives and processes the summarization task, while ner.py independently handles the NER task.
4. Each agent publishes its results as separate Announce notifications, complete with provenance records, allowing results to be archived, reviewed, and validated by any agent.

This makes the architecture **federated**, **auditable**, and **scalable across institutions**.

4.3.3 Academic Context and Benefits

(i) Research institutions benefit from enhanced reproducibility : Every inference is logged, versioned, and described with machine-readable audit trails.

(ii) Collaboration is facilitated : multisite archival institutions, libraries, and university labs can compose modular workflows simply by subscribing agents to topic- or task-specific event streams.

(iii) Compliance with FAIR principles is built-in : metadata about models, scripts, parameters, and runtime environments is persisted and queryable, supporting rigorous scientific requirements for transparency and stewardship.

(iv) Each agent can be independently upgraded, swapped, or retired-accelerating adoption of state-of-the-art models, tailored to discipline-specific needs (e.g., legal NER, biomedical summarization).

5 Discussion

5.1 Benefits

— Zero-install, reproducible execution
— Native auditability (LDN + PROV-O)
— Event-driven orchestration
— Seamless federation across agents
— Transparent prompt reconstruction and usage tracking

5.2 Limitations

— Performance : requires batching or cloud offload for large jobs
— Security : SHA pinning is essential for trust ; Authentication layer might be necessary in some cases to secure Inbox access
— Maturity : LDN not widely adopted yet outside digital libraries

6 Related Work

— Retrieval-Augmented Generation (RAG) architectures
— Federated learning and inference frameworks
— FAIR data and PROV-O for provenance
— W3C standards : ActivityStreams, LDN, PROV-O
— Serverless AI deployments and uv tooling

7 Future Work

— Support `-remote` execution using HF Jobs or Modal
— Convert LDN messages to RDF/Turtle for SPARQL querying
— Visualize full inference traces (e.g., provenance graphs)
— Define a lightweight « LDN-AI » profile for standardization

8 Conclusion

We have proposed and demonstrated a novel architecture for transparent, auditable, and distributed AI inference. By combining `uv run` and W3C LDN, we enable any organization to build FAIR-compliant, zero-install, event-driven pipelines using modular scripts and traceable notifications.

Our basic but working prototypes show the feasibility and versatility of this approach. We invite the community to extend these ideas toward an open standard for distributed AI orchestration.

Références

[1] Ahsan Bilal, David Ebert, and Beiyu Lin. Llms for explainable ai : A comprehensive survey, march 2025. arXiv :2504.00125 [cs] version : 1.

[2] Avash Palikhe, Zhenyu Yu, Zichong Wang, and Wenbin Zhang. Towards transparent ai : A survey on explainable large language models, june 2025. arXiv :2506.21812 [cs] version : 1.

[3] Igor Tufanov, Karen Hambardzumyan, Javier Ferrando, and Elena Voita. Lm transparency tool : Interactive tool for analyzing transformer language models, april 2024. arXiv :2404.07004 [cs].

[4] Maryam Amirizaniani, Jihan Yao, Adrian Lavergne, Elizabeth Snell Okada, Aman Chadha, Tanya Roosta, and Chirag Shah. Llmauditor : A framework for auditing large language models using human-in-the-loop, may 2024. arXiv :2402.09346 [cs] version : 3.

[5] Astral. astral-sh/uv, October 2025. Accessed October 2025.

[6] W3C. Linked data notifications, 2017. Accessed October 2025.

[7] Sarven Capadisli, Amy Guy, Christoph Lange, SÃűren Auer, Andrei Sambra, and Tim Berners-Lee. Linked data notifications : A resource-centric communication protocol. In *The Semantic Web : 14th International Conference, ESWC 2017, PortoroÅ¿, Slovenia, May 28 âĂŞ June 1, 2017, Proceedings, Part I*, pages 537–553, Berlin, Heidelberg, may 2017. Springer-Verlag.

[8] Jean-Paul Calbimonte, Davide Calvaresi, and Michael Schumacher. Multi-agent interactions on the web through linked data notifications. *Multi-Agent Systems and Agreement Technologies*, 2018. ISBN : 9783030017125 Meeting Name : 15th European Conference on Multi-Agent Systems (EUMAS 2017) Num Pages : 8 Place : Cham Publisher : Springer Series Number : Lecture Notes in Computer Science, vol. 10767.

[9] Matteo Cancellieri, Martin Docekal, David Pride, Morane Gruenpeter, David Douard, and Petr Knoth. Interoperable verification and dissemination of software assets in repositories using coar notify, august 2025. arXiv :2508.02335 [cs].

[10] W3C. Prov-o : The prov ontology, 2013. Accessed October 2025.