

From Passive Datasets to Agent-Enhanced Data Services: A Service-Composition View of Open Data Question Answering

Nasredine Cheniki (1), Salman Youssef (2), Yacine Sam (1),
Abderrahmane Maaradji (3)

(1) Université de Tours, France

(2) University of Windsor, Canada

(3) University of Doha for Science and Technology, Qatar

Abstract. Open government data portals publish hundreds of thousands of datasets, yet answering simple factual questions requires a human to traverse metadata search, heterogeneous file formats, and ad-hoc analysis. We argue this is a modeling failure: portals treat data as passive resources to be retrieved, while users need capabilities to be *composed*. We propose *Agent-Enhanced Data Services (AEDS)*, a model in which each dataset is exposed as an active service bundling data with domain-specialized reasoning. AEDS has three components: a *Publisher Agent* that curates and semantically transforms data at ingestion; an *Agent Registry* that advertises domain-specialized services; and an *Orchestrator Agent* that composes them in response to natural-language queries. We instantiate the consumption layer as *AGORA*, an orchestrator over data.gouv.fr (~73k datasets) with two-level retrieval and per-dataset routing between descriptive and tool-based reasoning. An empirical evaluation over eight benchmark questions spanning four domains characterizes routing behaviour, query decomposition, and data-quality robustness. The results indicate that planning depth, rather than routing calibration, is the primary determinant of system performance. Finally, we discuss how the proposed Publisher Agent and Agent Registry could extend AGORA toward a complete AEDS ecosystem.

Keywords: Open government data · agent-enhanced data services · service composition · multi-agent systems · Large Language Models.

1 Introduction

Open government data (OGD) catalogues now expose tens of thousands of datasets, yet practical reuse still demands substantial technical mediation. Despite proliferation of portals and publication policies, four barriers persist in today’s OGD ecosystems: *(i)* lack of integrated reasoning, as dataset APIs provide access to raw data only and lack embedded reasoning capabilities, forcing consumers to build their own interpretation and analysis logic; *(ii)* human-centric design, since most data are structured for human consumption (simple CSVs, unstructured text) rather than for machine agents, lacking semantic markup optimized for LLMs or autonomous reasoners; *(iii)* publication bottlenecks, as publishing high-quality, semantically rich data is time-consuming and converting

raw data into RDF or ontology-backed formats requires technical expertise that many publishers lack; and a *(iv)* passive data model, whereby datasets act as passive resources and traditional access is framed as a retrieval problem, whereas modern usage demands dynamic composition of data and logic.

This is not only an interface issue; it is a modeling issue. The community still largely treats the task as either document retrieval or generic conversational QA, while open-data settings require both catalogue-scale discovery and capability-level reasoning over heterogeneous resources [14, 20, 4].

Our position is that open-data access is better modeled as service composition over a registry of specialized capabilities than as document retrieval. Concretely, we propose Agent-Enhanced Data Services (AEDS): each dataset is exposed as an active service bundling data with domain-specialized reasoning, discoverable through a registry and composed on demand by an orchestrator. This reframing has two immediate consequences. First, it draws on two decades of research combining service-oriented architectures with multi-agent systems [24, 12, 11, 18], providing grounded infrastructure patterns (registries, typed endpoints, composition) rather than starting from scratch. Second, it splits the problem into a supply-side concern (preparing data to be callable) and a consumption-side concern (composing calls to answer a question), each of which can be engineered and evaluated separately. This paper makes four contributions, ranging from conceptual foundations to experimental validation.

1. **AEDS conceptual model** (conceptual): we reframe open-data access from retrieval to service composition and articulate a dual-agent ecosystem spanning supply and consumption.
2. **Three-component architecture** (design): Publisher Agent, Agent Registry, and Orchestrator with Domain-Specialized Agents, with component interfaces and transparency as cross-cutting invariants.
3. **AGORA, the consumption-layer instantiation** (built and operated): a two-level retrieval-and-reasoning pipeline over `data.gouv.fr` with per-dataset routing between descriptive (RAG) and tool-based (TECH) modes, realizing the Orchestrator Agent of AEDS.
4. **Evaluation** (empirical): eight benchmark questions over the full 73,860-dataset catalogue characterising routing behaviour, query decomposition, and data-quality robustness; a controlled LLM comparison (GPT-5-mini vs. LLaMA-3.3-70B-Instruct) reveals that planning depth is the primary performance differentiator.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the AEDS conceptual model and its three-component architecture. Section 4 describes AGORA, our implementation of the AEDS consumption layer, including the orchestration protocol and implementation details. Section 5 reports a preliminary experimental evaluation. Section 6 discusses the limitations of the current work and outlines the future evolution of AEDS, including the Publisher Agent and Agent Registry. Finally, Section 7 concludes the paper and highlights future research directions.

2 Related Work

OGD usability studies consistently report that publication effort outpaces practical reuse: metadata is fragmented, licensing is inconsistent, and semantic cues are weak across portals [14, 20]. The FAIR principles [23] argue for data that is machine-actionable by default; Croissant [2] carries this agenda into ML-ready settings. Most portal platforms (CKAN [15], DCAT-AP [7]) focus on metadata interoperability rather than embedded reasoning. Our work targets the gap between FAIR metadata and capability-level agent access: existing standards primarily describe datasets and their structure, whereas our focus is on exposing and discovering the actionable capabilities that agents can provide on top of these datasets.

Combining services with autonomous agents has been studied for nearly two decades. JIAC V [12] is an SOA-compliant agent framework; broader lines propose MAS-based SOA for enterprise integration and supply-chain management [24]. SoMAS integrate service offer/request, registries, and composition with autonomous agents for industrial control [11], while the Agents-and-Artifacts (A&A) programming model [18] treats agents and their environment as first-class abstractions over Web Services. These efforts established enduring design principles, including service directories [8], typed capability descriptors [13], service discovery, loose coupling, and capability composition, which remain relevant as service composition challenges re-emerge in the era of LLM agents. Rather than introducing a new paradigm, AEDS builds upon these foundations and adapts them to the context of data-centric ecosystems, extending capability discovery from software services to data-oriented agents responsible for publishing, describing, and exploiting FAIR and OGD resources.

Contemporary agentic AI practice modernizes these ideas on top of microservices backends, with LLM-based agents acting as orchestrators over services, events, and tools. Two architectural patterns recur: agent as intelligent façade, where an agent front-ends microservices and decides which to call; and microservice as reliable tool, where an LLM-based planner is central and microservices handle transactional or structured operations. Tool-augmented LMs such as ReAct [27], Toolformer [19], and ReAcTable [30] provide the execution substrate the Orchestrator invokes. Emerging standards such as the Model Context Protocol [3] bring the structure long present in FIPA/OASIS into the LLM-agent era. However, most current agentic AI architectures focus on tool invocation and orchestration, often assuming a predefined set of available tools and services; recent surveys identify tool retrieval and selection as key open challenges for LLM-based agents operating over large and evolving capability repositories [26]. While emerging standards improve interoperability between agents and tools, comparatively less attention has been paid to the discovery, publication, and exploitation of data-centric capabilities in open and decentralized data ecosystems. AEDS addresses this gap by extending capability discovery and orchestration beyond software tools to FAIR and OGD resources, where datasets, APIs, analytical services, and domain-specialized agents evolve independently.

RAG grounds generation in retrieved context [1]; hybrid dense-sparse retrieval is now standard for robust recall [21, 9]. Chunking [28, 16] and granularity choice [5] materially affect evidence quality. Although RAG systems retrieve textual passages to support answer generation, the underlying retrieval process can also be viewed broadly as the discovery of relevant external resources: in open-data environments, these resources may correspond to datasets, APIs, or specialized data services rather than document fragments. AEDS applies RAG at the metadata level for catalogue discovery and dataset selection; rather than directly generating answers from retrieved passages, it introduces a second reasoning stage within each selected service, enabling dataset-specific exploration, analysis, and tool invocation. This two-level process combines retrieval-based grounding with service-level reasoning over heterogeneous data resources.

Query decomposition improves coverage for complex questions [25]. Dataset search, surveyed by Chapman et al. [4], has until recently treated datasets as opaque units to rank. Across these research areas, answering complex information needs often requires combining evidence from multiple heterogeneous resources, yet existing approaches primarily emphasize the retrieval of relevant documents, entities, or datasets, leaving coordination and execution largely implicit. Recent surveys [29] instead characterize LLM-based agents as planners, coordinators, and tool users, providing a natural foundation for making these coordination mechanisms explicit; in AGORA, DSPy [10] provides the modular abstraction used to implement these agent roles. Accordingly, AEDS moves beyond dataset retrieval toward dataset orchestration, treating each dataset as a callable service with its own reasoning capabilities and enabling decomposition-driven coordination across heterogeneous data resources.

3 AEDS Conceptual Architecture

From retrieval to service composition. Let Q be a natural-language user query and $\mathcal{D}$ the set of datasets published in an OGD catalogue. The classical view models access as a ranking problem

$$Q \mapsto \text{rank}(\mathcal{D} \mid Q),$$

returning an ordered list of datasets for user interpretation. AEDS instead formulates access as a capability-discovery and service-composition problem

$$Q \mapsto \text{execute}(\text{compose}(A_1, \dots, A_k \mid Q)),$$

where each A_i is an agent-enhanced service exposing both a dataset (or dataset family) and the reasoning capabilities required to exploit it. This formulation extends service-oriented multi-agent principles to data-centric capabilities [11, 18].

Figure 1 shows the three AEDS components and their actor flows. Publisher Agent and Agent Registry represent the forward-looking supply layer, while the

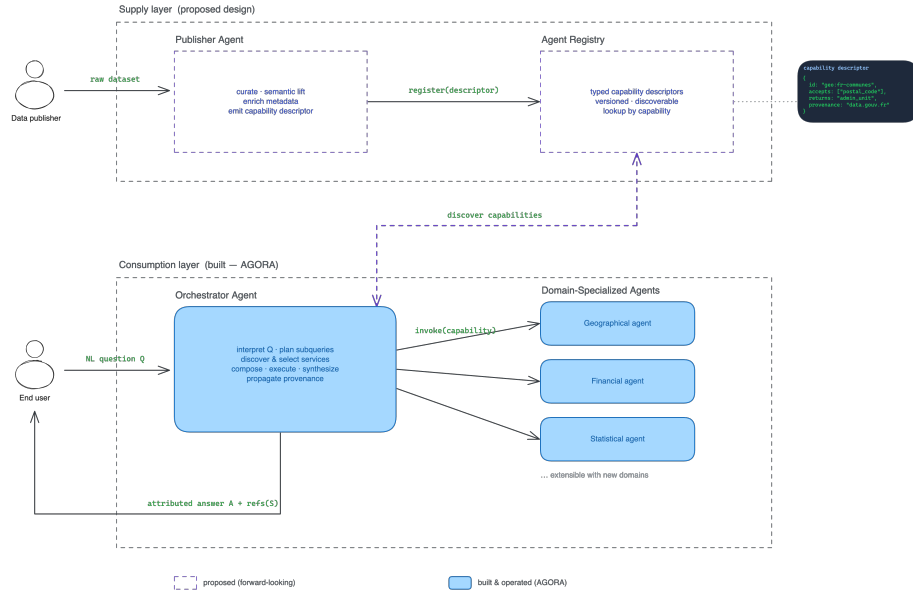


Fig. 1. AEDS conceptual architecture.

Orchestrator and Domain-Specialized Agents constitute the realized consumption layer implemented in AGORA, presented in Section 4.

The Agent Registry forms the backbone of the ecosystem. Publishers register domain-specialized agents, while consumers (orchestrators or other agents) discover them by capability rather than by string matching. The Registry stores typed capability descriptors, manages versioning and deprecation, and acts as a capability directory analogous to UDDI-style service registries. Unlike conventional metadata catalogues, Registry entries are callable: each advertises an execution contract rather than descriptive metadata alone. The consumption layer consists of Reasoning Agents. The Orchestrator Agent interprets natural-language queries, plans subtasks, selects and composes services, and synthesizes the final answer. Domain-Specialized Agents (e.g., geographical or financial agents) encapsulate domain-specific reasoning and can be extended as new publisher communities and application domains emerge. The Publisher Agent constitutes the supply layer. It accepts raw dataset uploads, curates and semantically enriches them, and generates the capability descriptors consumed by the Registry. This shifts part of the reasoning effort upstream, allowing datasets to become directly discoverable and callable.

Across these components, transparency is ensured through end-to-end propagation of provenance, execution traces, routing decisions, and attribution metadata. Such traceability is essential in public-sector settings, where explanation and accountability are as important as the generated answer.

These architectural choices embody four design principles: *(i)* capability over resource, whereby datasets are exposed as executable capabilities rather than static resources; *(ii)* separation of concerns, which decouples publication from consumption; *(iii)* explicit composability, whereby complex information needs are addressed through coordinated service composition; and *(iv)* transparency by design, ensuring end-to-end traceability throughout the execution workflow.

To realize these principles, each AEDS service exposes a common typed contract $A_i = \langle \text{descriptor}, \text{input-schema}, \text{output-schema}, \text{provenance}, \text{execution-hints} \rangle$. Unlike traditional dataset catalogues, which primarily expose descriptive metadata, AEDS contracts advertise operational capabilities. Discovery is therefore driven not only by topical relevance but also by a service’s ability to contribute to a reasoning workflow. Capability descriptors are expressed in machine-readable form, drawing inspiration from FIPA service directories [8] and OASIS service descriptions [13]. Input and output schemas define invocation and composition interfaces, provenance is propagated by construction to preserve attribution across composition steps, and execution hints specify latency, computational cost, and invocation constraints, enabling the Orchestrator to compare alternative execution plans. The resulting contract serves both as the publication artifact consumed by the Registry and as the coordination artifact used during service composition.

At publication time, the Publisher Agent curates an incoming dataset, derives its capability descriptor, and registers the resulting service in the Registry. At query time, the Orchestrator decomposes the user request into subgoals, discovers candidate services, selects an execution plan, invokes the selected services, and synthesizes a provenance-aware answer. The Registry provides the coordination interface between publication and consumption, enabling both sides to evolve independently while remaining loosely coupled. Overall, AEDS transforms open-data access from a retrieve-and-interpret paradigm into a discover-compose-execute paradigm, where answers emerge from the coordinated execution of data-centric services rather than manual inspection of datasets.

4 AGORA: Consumption-Layer Instantiation

AGORA (Agent-based Government Open-data Reasoning Assistant) is the realized consumption layer of AEDS, operating over `data.gouv.fr` (~73k datasets). In AEDS terms, AGORA plays the role of the Orchestrator Agent and instantiates the Domain-Specialized Agent pattern through per-dataset routed reasoning. In the absence of a populated Agent Registry, AGORA treats the `data.gouv.fr` catalogue itself as a fallback capability directory and derives per-dataset reasoning modes at query time.

Given the catalogue datasets $\mathcal{D}$ and their indexed metadata representation $\mathcal{I}$ (title, description, tags, organization, and hybrid-search features), AGORA implements a two-level retrieval and reasoning process. Given a user query Q , the first level identifies a subset of relevant datasets $S \subseteq \mathcal{D}$ from $\mathcal{I}$, while the second level extracts evidence from each selected dataset and performs dataset-specific

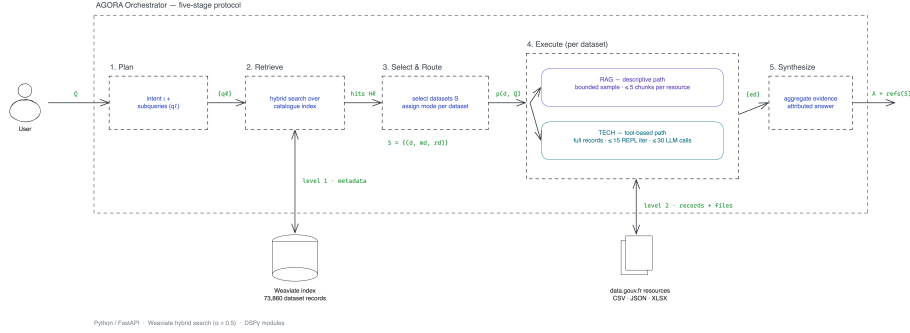


Fig. 2. AGORA runtime pipeline.

reasoning before synthesizing an attributed answer. In the AEDS perspective, these two stages correspond respectively to service discovery and service invocation/composition, separating global catalogue search from local dataset reasoning and reducing unnecessary file-level processing.

For each selected dataset d , a routing function $\rho(d, Q) \in \{\text{RAG}, \text{TECH}\}$ selects the most appropriate reasoning strategy: RAG (descriptive path), performing passage-style extraction and synthesis when representative text evidence is sufficient, or TECH (tool-based path), performing structured parsing, lookup, and light computation when the question requires precise operations over records/tables.

Routing acts as a local planner executed independently for every selected dataset. Consequently, the same dataset may follow different reasoning paths depending on the user query. Descriptive questions generally activate the RAG path, whereas analytical, lookup, aggregation, or filtering tasks are routed to the TECH path. The two modes intentionally operate under different execution budgets: RAG samples only a bounded subset of records sufficient for descriptive evidence, whereas TECH processes the complete dataset whenever exhaustive computation is required. The global objective is to maximize grounded answer quality while minimizing unnecessary processing: $A = \pi_{\text{syn}}(Q, \bigoplus_{d \in S^\downarrow} e_d, \text{refs}(S))$, where e_d denotes the evidence produced for dataset d , $\bigoplus$ aggregates evidence across datasets, and $S^\downarrow$ denotes a deterministic ordering ensuring reproducibility. The synthesis function combines the user query, aggregated evidence, and provenance references into the final attributed answer.

The routing decisions are executed through the orchestration protocol illustrated in Figures 2 and 3. Given a query Q , the orchestration protocol executes five successive operators, each producing the artifact consumed by the next stage:

1. **Plan:** $\pi_{\text{plan}}(Q) \rightarrow P = (\iota, \{q_\ell\}_{\ell=1}^k)$, where ι is the inferred intent and $\{q_\ell\}$ the generated subqueries.
2. **Retrieve:** $H_\ell = \text{search}_{\mathcal{I}}(q_\ell)$ for each subquery, where H_ℓ is the ranked set of candidate datasets and $\mathcal{I}$ the catalogue index.

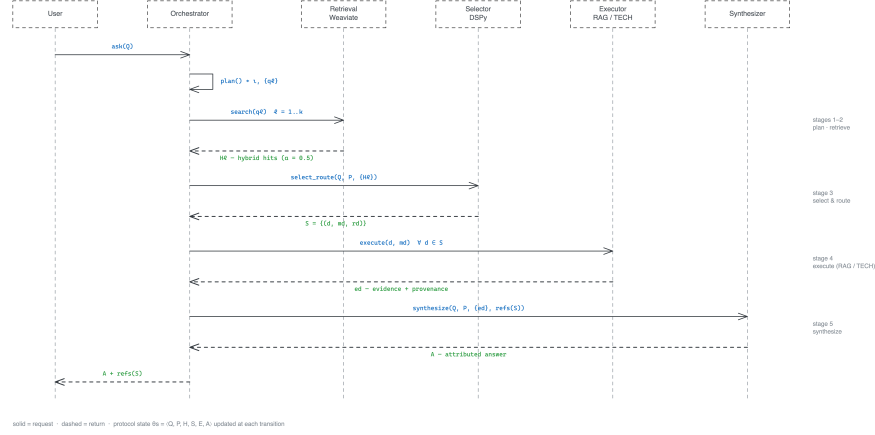


Fig. 3. Interaction sequence across orchestration stages.

3. **Select/Route:** $\pi_{\text{sel}}(Q, P, \{H_\ell\}) \rightarrow S = \{(d, m_d, r_d)\}$, where $m_d \in \{\text{RAG}, \text{TECH}\}$ is the assigned reasoning mode and r_d its rationale.
4. **Execute:** $e_d = \text{Ex}_{m_d}(d, Q)$ for each selected dataset via the RAG or TECH path, where e_d is the evidence block returned by the chosen execution strategy.
5. **Synthesize:** $\pi_{\text{syn}}(Q, P, \{e_d\}, \text{refs}(S)) \rightarrow A$. The synthesis stage combines the user query, execution plan, evidence blocks, and provenance references to generate the final attributed answer A , reconciling potentially conflicting evidence and preserving traceability to underlying sources.

The protocol state $\theta_s = \langle Q, P, H, S, E, A \rangle$ is updated at each transition; only orchestrator-controlled transitions are allowed, preventing ad hoc agent loops and preserving a clear provenance chain. The Orchestrator serves as the unique control authority throughout execution. It decomposes user queries, coordinates retrieval and routing, removes duplicate results across subqueries, enforces protocol ordering, manages execution budgets, maintains the protocol state, and propagates provenance information to the synthesis stage. Fallback policies are likewise orchestrator-controlled: if routing fails, all selected datasets are processed through the RAG path; if reasoning services become unavailable, execution degrades gracefully to direct LLM inference. The above orchestration protocol is realized by the implementation described below.

4.1 Implementation

AGORA is implemented in Python/FastAPI [17]. Hybrid metadata retrieval relies on Weaviate [22], while DSPy [10] implements the planner, selector, execution agents, and synthesis modules. Dataset metadata are harvested from the `data.gouv.fr` API [6], embedded for semantic retrieval, and indexed for hybrid

dense-sparse search. The reference implementation and live demo are publicly available¹.

Selected datasets are processed through one of two execution paths according to the routing decision. The RAG path downloads resources on demand, parses and chunks textual content when necessary [28, 16, 5], retrieves the most relevant passages, and generates grounded evidence. The TECH path parses structured resources (e.g., CSV, JSON, and XLSX) into executable representations supporting lookup, filtering, aggregation, and lightweight computation. When code execution is unavailable, a constrained prompt-based fallback preserves system functionality. Resources requiring preprocessing (e.g., compressed archives or spreadsheets) are transparently normalized before execution.

To ensure scalable execution, AGORA combines dense and sparse retrieval for dataset discovery, applies bounded evidence extraction to avoid over-representing individual resources, and enforces deterministic evidence ordering together with resource budgets to control latency and computational cost. Execution progress, routing decisions, and source provenance are propagated throughout the pipeline and exposed in the user interface, allowing users to inspect how datasets are selected and processed. Such transparency is particularly important in public-sector settings, where evidence attribution and explainability are as important as the generated answer. The modular architecture also provides stage-level cost observability by tracking LLM token usage across the planner, selector, RAG execution, TECH execution, and synthesis stages, making it possible to identify where computation concentrates and to target efficiency optimizations that monolithic QA pipelines do not readily support. The implementation described above is illustrated through the following end-to-end execution example.

4.2 Running Example

We trace a single question through all five stages to illustrate how each component contributes. The question is:

“Quels jeux de données sont disponibles sur la qualité de l’air en France ?” (Which datasets are available on air quality in France?)

Stage 1: Plan. The Planner (DSPy `ChainOfThought`) decomposes the question into five retrieval-oriented subqueries: (i) hourly measurements from official monitoring networks (Atmo France/Airparif), (ii) historical time series by pollutant (PM_{2.5}, NO₂, O₃), (iii) national emission inventories by sector (CITEPA), (iv) forecast and modelling outputs (daily ATMO indices), (v) citizen-sensor and low-cost network data. This decomposition steers hybrid retrieval towards complementary facets of the topic rather than returning near-duplicate results.

¹ Code available at <https://github.com/salmanyoussef/agora>. Live demo available at <https://agora-demo.jollycoast-9eb8a291.eastus2.azurecontainerapps.io>. The interface lets users select *General* mode (descriptive RAG only) or *Technical* mode, in which the selector routes each dataset to either RAG or tool-based reasoning over structured resources. Intermediate pipeline stages are streamed live, making the orchestration protocol directly observable.

Stage 2: Retrieve. For each subquery, AGORA queries Weaviate with hybrid search ($\alpha = 0.5$, $k = 5$); the five searches together surface 25 candidate dataset records from the 73,860-entry catalogue.

Stage 3: Select / Route. The Selector agent evaluates the 25 candidates and selects 22 relevant datasets. Since the question asks for an overview of available datasets rather than a specific computation, the Selector assigns RAG mode to all 22.

Stage 4: Execute (RAG path). For each selected dataset, AGORA downloads available resources, extracts text, chunks into 800-character windows (100-character overlap), and retrieves the top-15 chunks most similar to the active subquery. A DSPy `Predict` module produces a self-contained evidence block attributed to the source dataset and URL. For datasets hosted on defunct regional portals (e.g., `datacat.datalocale.fr`), the pipeline catches the download error, logs it transparently, and continues without aborting.

Stage 5: Synthesise. The Synthesis agent receives all evidence blocks and produces a single structured French-language answer organised by dataset type: official monitoring networks (Airparif hourly and monthly series, Paris annual averages), regional emission inventories (Pays de la Loire, AtmoSud, Atmo Auvergne-Rhône-Alpes), the national ATMO daily index, and citizen-sensor corpora (Côtes-d’Armor low-cost sensors, IRISA mobile campaign). Every cited dataset is linked by its `data.gouv.fr` URL, and the response notes explicitly where file downloads failed and why. The full pipeline consumed approximately \$0.10 in LLM calls and completed in under 19 minutes on a consumer laptop, a latency attributable to serial on-demand resource downloads across 22 datasets rather than to LLM inference time.

5 Preliminary Evaluation

We conducted two parallel evaluations of the AGORA consumption layer over the full `data.gouv.fr` catalogue (73,860 datasets, June 2026 snapshot), using the same eight benchmark questions. The goal is not to claim state-of-the-art performance (formal benchmarking with ground-truth annotations is future work) but to characterise system behaviour across question types and domains and to identify how LLM quality affects each pipeline stage.

5.1 Setup

We defined eight evaluation questions spanning four thematic domains and two question types (Table 1). *Discovery* questions ask the system to identify relevant datasets on a topic; they typically trigger broad query decomposition and RAG-mode execution. *Analytical* questions require filtering, counting, or ranking values within datasets; they are expected to trigger TECH-mode routing. The

Table 1. Pipeline outcomes for GPT-5-mini and LLaMA-3.3-70B-Instruct on the eight evaluation queries. “Sub.” = planner subqueries; “Sel.” = datasets selected; “RAG/Tech” = routing counts; “Tech” = technical-mode datasets (LLaMA never uses RAG); “Ans.” = qualitative outcome (✓ informative, ∼ partial, ✗ unanswered). LLaMA consistently generates one subquery (column omitted).

ID	Domain	Type	GPT-5-mini ($k=5$)				LLaMA-3.3-70B ($k=3$)		
			Sub.	Sel.	RAG/Tech	Ans.	Sel.	Tech	Ans.
Q1	Environment	Discovery	6	25	16 / 9	✓	3	3	∼
Q2	Transport	Discovery	6	14	7 / 7	✓	3	3	∼
Q3	Transport	Analytical	1	2	0 / 2	✓	0	0	✗
Q4	Finance	Discovery	5	13	8 / 5	✓	1	1	∼
Q5	Demographics	Analytical	1	3	0 / 3	✓	0	0	✗
Q6	Education	Discovery	7	16	9 / 7	✓	3	3	∼
Q7	Energy	Discovery	5	11	4 / 7	✓	0	0	✗
Q8	Energy	Analytical	1	2	0 / 2	✓	1	1	✗

first run used GPT-5-mini ($k=5$ candidates per subquery), the default AGORA configuration. The second run used LLaMA-3.3-70B-Instruct accessed through the same Azure OpenAI-compatible endpoint with $k=3$; only the underlying LLM was swapped, keeping all pipeline stages and prompts identical.

5.2 Results and Observations

Routing behaviour. The Selector routes per dataset independently of question type, using dataset descriptions to judge whether a random passage sample (RAG) or structured code execution (TECH) is more appropriate. Q1 (air quality, discovery) selected 25 datasets across 6 subqueries with a 16/9 RAG/TECH split; descriptive datasets were assigned RAG while those hosting structured measurement tables were assigned TECH. Q2 (road accidents, discovery) selected 14 datasets with an even 7 / 7 RAG/TECH split, reflecting the heterogeneous nature of accident data: some tabular files require field-level extraction while others are described well by metadata alone. Q3 (analytical: count Vélib’ stations in Paris) generated a single targeted subquery, selected 2 datasets, and routed both to TECH mode; the pipeline returned the precise count of 1,517 stations by executing Python against the official Vélib’ catalogue download, illustrating that TECH routing enables provably grounded answers to exact lookup and count queries. Q5 (analytical: French communes with population >100k) similarly routed all three selected datasets to TECH and, after recovering from a primary-URL 404 via fallback, produced a complete ranked list of 50 communes directly from the INSEE population XLSX preview in 123 seconds. Q8 (analytical: EV charging points in France) routed both selected datasets to TECH and returned a data-backed total of 2,950,882 charging points (Enedis, 2026 Q1), broken down by category (public: 192,008; private residential: 1,667,790; corporate: 1,091,084). Discovery queries (Q2, Q4, Q6, Q7) consistently produced mixed RAG/TECH splits: Q6 (schools) reached 9 / 7 across 16 datasets; Q7

(renewable energy) reached 4 / 7 across 11 datasets, with TECH favoured for tabular time-series and RAG for descriptive reports and forecasts.

Query decomposition. The Planner consistently decomposes broad discovery questions into semantically distinct subqueries rather than issuing a single retrieval call. For Q1 (air quality), six subqueries probe official monitoring networks, historical time series, emission inventories, forecast models, citizen sensors, and air-quality indices separately, recovering 25 relevant datasets that a single query would not have surfaced with comparable precision. Q2 generated six subqueries targeting BAAC national accident tables, annual mortality statistics, vulnerable-road-user data, radar locations, traffic-count datasets, and circumstantial fields (weather, alcohol, speed), recovering 14 relevant datasets across all six facets. Q4 (municipal budgets) generated five subqueries targeting primitive budgets, expenditure breakdowns by nature and function, fiscal transfers and DGF dotations, multi-year time series, and cross-source join keys, selecting 13 datasets from five distinct facets of public finance data; Q6 (school infrastructure) generated seven subqueries and selected 16 datasets spanning national registries, pupil counts, baccalauréat indicators, staffing ratios, catchment boundaries, and boarding/sports facilities. Overall, the results in Table 1 indicate a clear relationship between query decomposition and dataset selection. Discovery questions consistently generate broader execution plans, producing between five and seven planner subqueries and leading to the selection of 11 to 25 datasets, whereas analytical questions are addressed through targeted plans of a single subquery and two to three selected datasets. This behavior shows that AGORA dynamically adapts the scope of dataset exploration to the underlying information need.

Data quality robustness. Several older datasets (pre-2015) are hosted on regional portals that are no longer reachable. AGORA handles these gracefully: download failures are caught, logged, and reported transparently in the final answer, while execution continues with the remaining evidence. Q5 illustrates how the pipeline recovers from hosting failures: the primary resource URL returned 404 at query time. A URL-fallback mechanism retries alternative resource URLs from the same dataset metadata record, successfully downloading the XLSX file via a secondary URL. Additionally, when code execution via a sandboxed REPL is unavailable, the Technical Agent falls back to reading a compact sorted preview table directly. Combining these two mechanisms, Q5 completed at a cost of \$0.028.

LLM comparison. Comparing the two columns of Table 1 shows that the planning stage is the primary differentiator. GPT-5-mini generates 1 to 7 subqueries per question (average 4.5), while LLaMA-3.3-70B consistently generates exactly one. Since retrieval coverage scales with subquery count, GPT selects 86 datasets in total versus 11 for LLaMA (a 7.8x gap). LLaMA selects zero datasets for Q3, Q5, and Q7, where its single subquery misses the thematic specificity that

GPT’s decomposition recovers. For the five questions where LLaMA does select datasets, it routes exclusively to TECH mode ($\text{RAG} = 0$), in contrast to GPT’s mixed allocation. Analytical questions (Q3, Q5, Q8) are answered precisely by GPT but not by LLaMA. The trade-off is latency: LLaMA completes each query in 8 to 153 seconds versus 2 to 54 minutes for GPT, a roughly 40x difference attributable to GPT’s broader dataset selection and multi-dataset execution. These results suggest that LLM quality affects AGORA primarily through planning depth, not routing calibration; improving subquery generation for open-weight models is therefore the highest-priority direction for reducing the performance gap.

Limitations. Without ground-truth annotated answers, we cannot compute precision, recall, or factual accuracy metrics. Full empirical evaluation, including human judgment of answer quality, ablations of the planner and routing components, and latency profiling, is identified as the primary remaining contribution for the next iteration of this work. The LLM comparison is exploratory: a single run per model, with differing k values, shows behavioral trends rather than statistical conclusions.

6 Discussion and Limitations

The distinction between Level 1 dataset discovery and Level 2 dataset reasoning provides a useful diagnostic framework for analyzing system behavior. Failures may arise during planning (π_{plan}), dataset discovery, selection and routing (π_{sel}), execution (Ex_{m_d}), or synthesis (π_{syn}). By isolating these stages, the architecture enables targeted improvements to individual protocol components rather than relying on monolithic prompt engineering.

Routing uncertainty. The dynamic routing mechanism is effective for heterogeneous data, but it introduces dependency on selector quality. Incorrect mode assignment can either miss analytical requirements (false RAG) or incur avoidable cost (false TECH). Better calibration and uncertainty-aware routing are natural next steps. In the current instantiation, routing decisions are made from a 500-character description window, an explicitly approximate step that trades routing accuracy for low overhead. Richer dataset profiling at selection time is a concrete improvement direction.

Generalization beyond data.gouv.fr. AGORA depends on a catalogue API of sufficient quality (bulk listing, structured metadata, stable resource URIs). Porting to a second portal is tractable but not free: ingestion, hybrid-search tuning, and selector prompts each require calibration. Nevertheless, the underlying orchestration protocol, including discovery, routing, and evidence synthesis, is largely independent of the target portal and provides a foundation for broader deployment.

Future evolution of AEDS. AGORA demonstrates the feasibility of the consumption layer of AEDS. Extending the framework with Publisher Agents and an Agent Registry represents a natural next step toward a complete ecosystem in which datasets expose machine-actionable capabilities. This requires further work on semantic enrichment, capability publication, and governance.

7 Conclusion

We reframed open-data access as service composition over specialized capabilities through Agent-Enhanced Data Services (AEDS). Its consumption layer is realized as AGORA, an orchestrator over `data.gouv.fr` combining two-level retrieval with per-dataset RAG/TECH routing. An evaluation over eight benchmark questions shows that planning depth, rather than routing calibration, is the primary determinant of system performance. These results provide initial evidence supporting a transition from passive OGD portals toward agent-enhanced data ecosystems. Future work will establish a comprehensive benchmark for AGORA before extending AEDS with Publisher Agents and an Agent Registry and evaluating the complete framework across multiple open-data portals.

Declaration on Generative AI. During the preparation of this work, the authors used Claude (Anthropic) to support language editing and to assist with debugging implementation code. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of this publication.

References

1. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)
2. Akhtar, M., Benjelloun, O., Conforti, C., et al.: Croissant: A metadata format for ML-ready datasets. In: Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning (DEEM) (2024)
3. Anthropic: Model context protocol (MCP) specification. <https://modelcontextprotocol.io> (2024)
4. Chapman, A., Simperl, E., Koesten, L., et al.: Dataset search: A survey. The VLDB Journal **29**, 251–272 (2020)
5. Chen, T., Wang, H., Chen, S., et al.: Dense X retrieval: What retrieval granularity should we use? In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2024)
6. Etalab: data.gouv.fr: The french open data portal. <https://www.data.gouv.fr> (2024)
7. European Commission, SEMIC: DCAT application profile for data portals in Europe (DCAT-AP). W3C-aligned specification, <https://semiceu.github.io/DCAT-AP/> (2023)
8. Foundation for Intelligent Physical Agents: FIPA agent management specification. Tech. Rep. SC00023K, FIPA (2002)

9. Gao, L., Ma, X., Lin, J., Callan, J.: Precise zero-shot dense retrieval without relevance labels. In: ACL (2023)
10. Khattab, O., Singhvi, A., Maheshwari, P., et al.: DSPy: Compiling declarative language model calls into self-improving pipelines. arXiv preprint arXiv:2310.03714 (2023)
11. Leitão, P., Marik, V., Vrba, P.: Past, present, and future of industrial agent applications. IEEE Transactions on Industrial Informatics **9**(4), 2360–2372 (2013)
12. Lützenberger, M., Küster, T., Konnerth, T., et al.: JIAC V: A MAS framework for industrial applications. In: Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2013)
13. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Reference model for service oriented architecture 1.0. Tech. rep., OASIS (2006)
14. Nikiforova, A., McBride, K.: Open government data portal usability: A user-centred usability analysis of 41 open government data portals. Telematics and Informatics **58**, 101539 (2021)
15. Open Knowledge Foundation: CKAN: The open source data management system. <https://ckan.org> (2024)
16. Qu, R., Tu, R., Bao, F.: Is semantic chunking worth the computational cost? arXiv preprint arXiv:2410.13070 (2024)
17. Ramírez, S.: Fastapi. <https://fastapi.tiangolo.com> (2018)
18. Ricci, A., Buda, C., Zaghini, N.: An agent-oriented programming model for SOA. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN) (2007)
19. Schick, T., Dwivedi-Yu, J., Dessì, R., et al.: Toolformer: Language models can teach themselves to use tools. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)
20. Wang, F., et al.: Paths to open government data reuse: A three-dimensional framework of information need, data and government preparation. Information & Management **60**(8), 103879 (2023)
21. Wang, X., Wei, Z., Xu, M., et al.: Searching for best practices in retrieval-augmented generation. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2024)
22. Weaviate B.V.: Weaviate: Open-source vector database. <https://weaviate.io>
23. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., et al.: The FAIR guiding principles for scientific data management and stewardship. Scientific Data **3** (2016)
24. Wooldridge, M., Ciancarini, P.: Agent-oriented software engineering: The state of the art. In: First International Workshop on Agent-Oriented Software Engineering (AOSE) (2000)
25. Xiong, W., Li, X.L., Iyer, S., et al.: Answering complex open-domain questions with multi-hop dense retrieval. In: ICLR (2021)
26. Xu, W., Huang, C., Gao, S., Shang, S.: Llm-based agents for tool learning: A survey. Data Science and Engineering (2025)
27. Yao, S., Zhao, J., Yu, D., et al.: ReAct: Synergizing reasoning and acting in language models. In: ICLR (2023)
28. Yepes, A.J., et al.: Financial report chunking for effective retrieval augmented generation. arXiv preprint arXiv:2402.05131 (2024)
29. Yu, C., Cheng, Z., Cui, H., et al.: A survey on agent workflow – status and future. arXiv preprint arXiv:2508.01186 (2025)
30. Zhang, Y., Henkel, J., Floratou, A., et al.: ReAcTable: Enhancing ReAct for table question answering. Proceedings of the VLDB Endowment **17**(8), 1981–1994 (2024)