ASSESSMENT OF LEARNING TECHNOLOGIES AND FRAMEWORKS FOR INTELLIGENT AND ETHICAL AI

D3.1 Initial AutoML Data Warehouse and Semantic Knowledge Graph

Deliverable No:	D3.1
Deliverable Name:	Initial AutoML Data Warehouse and Semantic Knowledge Graph
Work Package:	WP3
Task:	T3.1, T3.3 Partially: T3.2, T4.2
Dissemination Level:	PU
Deliverable Type:	R
Lead Organization:	KInIT
Submission month:	M18
Date:	31 March 2026

Funded by the European Union

Funded by the European Union under Grant Agreement 101177912. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Research Executive Agency (REA). Neither the European Union nor the granting authority can be held responsible for them.

Project description

Acronym	**ALFIE**
Title	**Assessment of Learning technologies and Frameworks for Intelligent and Ethical AI**
Coordinator	CATALINK
Reference	101177912
Type	Research and Innovation Actions (RIA)
Programme	**Horizon Europe (HORIZON)**
Topic	HORIZON-CL2-2024-TRANSFORMATIONS-01-06 Beyond the horizon: A human-friendly deployment of artificial intelligence and related technologies
Start	01.10.2024
Duration	36 months
Website	alfie-project.eu
Consortium	**Catalink Limited** (CTL)**,** Cyprus, Coordinator **University of Brighton** (UoB), United Kingdom **Centre for Research & Technology** (CERTH)**,** Greece **Distributed Analytics Solutions LTD** (DAS), United Kingdom **Edge Hill University** (EHU), United Kingdom **Kempelen Institute of Intelligent Technologies** (KINIT), Slovakia **Universitat Autònoma de Barcelona** (UAB), Spain **Robert Bosch Espana SLU** (Bosch), Spain **Eindhoven University of Technology (**TU/e), Netherlands **Diadikasia Business Consulting** (DBC), Greece

Document history

Version	Date	Beneficiary	Description
0.1	27.01.2026	KInIT	Initial ToC and structure created
0.2	28.01.2026	UAB	Described trial use case
0.3	12.02.2026	Bosch	Described trial use case
0.4	24.02.2026	CTL	Described trial use case
0.5	26.02.2026	KInIT	Input added
0.6	27.02.2026	EHU	Input added
0.7	02.03.2026	CERTH	Input added
0.8	05.03.2026	KInIT	First draft
0.9	10.03.2026	UoB	Reviewed and proposed amendments
0.10	12.03.2026	DAS	Reviewed and proposed amendments
0.11	17.03.2026	KInIT, CERTH, EHU	Resolution of review comments.
1.0	23.03.2026	CTL	Final version

Authors

Partner	Name(s)
KInIT	Peter Zigo, Viktor Košťan, Martin Tamajka, Marcel Veselý
CERTH	Sotiris Diplaris, Alexandros Vassiliades, Stamatis Samaras, Spyridon Symeonidis
EHU	Chukwudi Uwasomba, Nonso Nnamoko, Yannis Korkontzelos
UAB	Sarah Anne Mcdonagh
CTL	Zenon Lamprou
Bosch	Leonardo Abad Martin

Contributors

Partner	Contribution type	Name
UoB	Review	Ignacio Cabrera Martin
DAS	Review	Charlie Gadd

Glossary

Acronym	Definition
AI	Artificial Intelligence
API	Application Programming Interface
AutoDW	AutoML Data Warehouse
CSV	Comma-Separated Values
ETD-Hub	EthiTech Dialogue Hub
GDPR	General Data Protection Regulation
JSON	JavaScript Object Notation
ML	Machine Learning
ONNX	Open Neural Network Exchange
PKL	Pickle (Python serialization format)
RDF	Resource Description Framework
REST	Representational State Transfer
SemKG	Semantic Knowledge Graph
SPARQL	SPARQL Protocol and RDF Query Language
TUC	Trial Use Case
UI	User Interface
URI	Uniform Resource Identifier
XAI	Explainable Artificial Intelligence

Executive Summary

This deliverable provides the initial implementation of the AutoML Data Warehouse and Semantic Knowledge Graph, two central components enabling the data, storage, and semantic foundations of the ALFIE platform. Its objective is to establish a coherent and interoperable infrastructure that supports trustworthy, transparent, and ethically aligned AI development.

The work carried out includes the consolidation of all available project datasets, both public and pilot-provided, into a unified warehouse with automated metadata extraction, versioning, and orchestrated data flows. In parallel, a structured semantic pipeline was developed to transform ethical and legal discourse into an ontology-aligned knowledge graph enriched with provenance, evidence, and confidence information. Together, these components form the backbone through which data, models, bias assessments, and explainability outputs are stored, managed, and retrieved.

The results of this deliverable provide essential infrastructure for all subsequent stages of the ALFIE platform. They ensure that AI workflows can be conducted with full traceability, that ethical knowledge can be systematically leveraged in assessments, and that future extensions, such as expanded dataset integration and enhanced semantic reasoning, can build on a stable and compliant foundation.

Contents

Figures

Tables

1 Introduction

1.1 Overview

This deliverable presents the initial implementation of two central components of the ALFIE platform: the AutoML Data Warehouse (AutoDW) and the Semantic Knowledge Graph (SemKG). Additionally, the deliverable describes in detail the datasets gathered from ALFIE trial use cases as well as ethical and legal documents used for grounding of ethical assessment and recommendations.

The key outcomes are:

1. A functional, scalable data warehouse supporting heterogeneous datasets, model artifacts, and downstream AI services.
2. Complete documentation of initial trial-use-case data for website accessibility, driver-state monitoring, and compliance alert classification.
3. A validated Semantic Knowledge Graph capturing ethically relevant knowledge in a machine-interpretable and query-ready structure.
4. An event-driven orchestration mechanism enabling reliable interaction between data ingestion, bias and data drift analysis, AutoML processes, concept drift simulation, and explainability generation.

1.2 Relation to other tasks and deliverables

This deliverable is related to the following other ALFIE tasks and deliverables:

Receives inputs from:

Table 1. D3.1 Input from other tasks and deliverables

Deliverable	Due Date	Input for D3.1
D1.4	31 Mar 2025 (revised version)	Summary information for ALFIE datasets
D4.1	30 Sep 2025	Trial use cases, ALFIE architecture, specification of AutoDW and Semantic Knowledge Graph
D3.3	31 Mar 2026	Component interaction using Kafka and Task manager

Provides outputs to:

Table 2. D3.1 Output for other tasks and deliverables

Deliverable	Due Date	Output from D3.1
D3.2	30 Jun 2027	Initial datasets, metadata, versioning structure, initial SemKG and reasoning baseline
D3.3	31 Mar 2026	AutoDW information and REST API reference, datasets, ethical/legal documents, SemKG information
D3.4	30 Jun 2027	AutoDW information and REST API reference, datasets, ethical/legal documents, SemKG information
D3.6	30 Jun 2027	Data warehouse information and REST API reference
D4.2	31 Jan 2027	AutoDW information and REST API reference, datasets, orchestration events, SemKG information
D4.3	30 Apr 2026	AutoDW information and REST API reference, datasets
D4.4	31 Jul 2027	AutoDW information and REST API reference, datasets

1.3 Structure of the deliverable

The structure of the deliverable is as follows. Section 2 is dedicated to AutoML Data Warehouse (AutoDW) as a service that serves as a central storage used by all major services contained in the AutoML platform. The section briefly describes how AutoDW is connected to other services in terms of data exchange and interfaces (detailed information can be found in D4.1), how data is stored within the services, how it can be accessed and what data has been collected so far. Section 3 focuses on data collected for the three trial use cases (TUC), where we, for every TUC, describe the purpose of collected data in context of the TUC, format, annotations and other important properties of collected data. Section 4 describes ethical and legal frameworks and documents that have been identified as relevant in terms of providing ethical and legal guidance to the users when creating ML-based solutions through the AutoML platform. Section 5 describes the initial state of the Semantic knowledge graph (SemKG) implementation. The section provides details on what SemKG is, what is the intended use of SemKG in the AutoML platform and what is the relation with other services and components of the platform, how the SemKG is created, what form it has and how it can be consumed by other services. Section 6 concludes the deliverable.

2 AutoML Data Warehouse

2.1 Brief overview of the AutoDW service

The **AutoML Data Warehouse (AutoDW)** serves as the central data management and orchestration backbone for the ALFIE platform. Its primary objective is to facilitate the development of balanced and unbiased AI models by providing a robust infrastructure for the persistent storage, organization, and controlled access of heterogeneous data assets. By employing a hybrid storage architecture, utilizing **MongoDB**[1] for flexible metadata management and **MinIO**[2] for scalable object storage, AutoDW ensures the traceability and reproducibility of AI workflows. Furthermore, AutoDW integrates ethics and legal related data from the **ETD-Hub (WP2)** and coordinates with the **Agentic Core (T3.2)** to orchestrate event-driven pipelines (employing **Apache Kafka**[3] message queue logic), ensuring that data handling aligns with ALFIE's rigorous legal and ethical standards. Finally, the AutoDW provides the appropriate data storage for the results of the **Semantic Knowledge Graph service (T3.3)** utilizing the popular graph database **GraphDB**[4] to support T3.3 needs.

In summary, the AutoDW is the orchestration and storage backbone of the ALFIE platform as it manages the lifecycle of AI assets from raw data ingestion to versioned model deployment and ethical reporting. Its organization chart is presented in Figure 1 highlighting its key data storage components and capabilities.

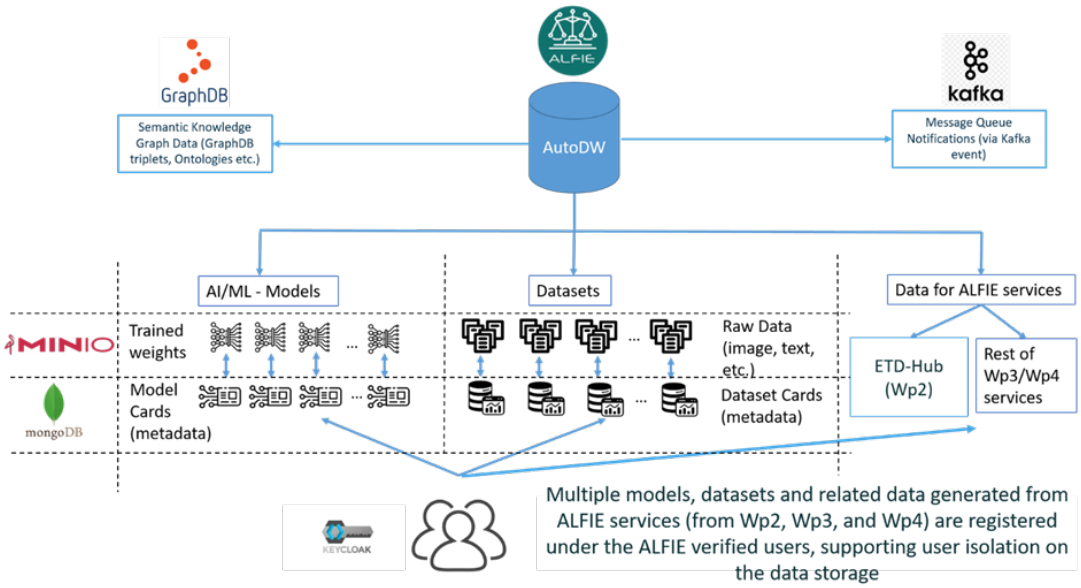

Figure 1. AutoDW organization chart and key data storage components

[1] https://www.mongodb.com/
[2] https://www.min.io/
[3] https://kafka.apache.org/
[4] https://graphdb.ontotext.com/

2.1.1 Relation to ALFIE services

AutoDW is not merely a storage repository but a dynamic service that interacts with both data producers and consumers across the ALFIE ecosystem. It bridges the gap between raw data acquisition and model deployment by managing the outputs of various downstream components, including bias detection and mitigation, explainability reports, and trained model files. Furthermore, the AutoDW is involved in ALFIE's task organization by hosting and storing all the registered task events in an executed pipeline. Finally, it connects ETD-Hub data with the Semantic Knowledge Graph service allowing the saving of structured representations of ethical and legal data and SPARQL querying on the graph.

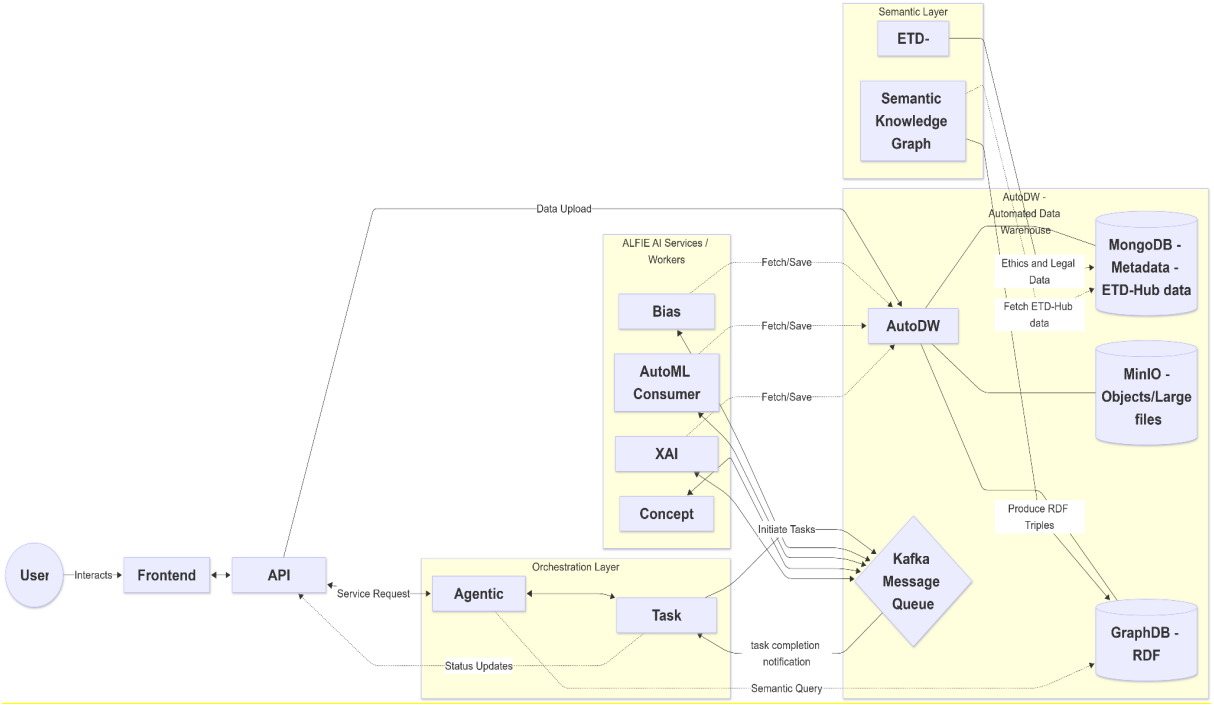

Figure 2. Updated flow diagram of AutoDW

The connections between AutoDW and the rest of ALFIE services has been explained in detail in D4.1 – "Initial AutoML platform co-designed architecture, functional requirements and operational prototype", specifically in Figure 2 of D4.1. However, due to updates since the publication of D4.1 there have been some minor changes which we present in Figure 2 for clarity. This is now the updated flow of the diagram of AutoDW reflecting its role and connection with the rest of ALFIE components. In order to assist in understanding of how AutoDW works and its role in the ALFIE ecosystem, a summary of AutoDW and its relation with the rest of ALFIE components is provided below.

Relation with ALFIE Components:

- **Data producers:**
  - **User interface / API Gateway:** Facilitates user interaction including dataset upload, download of AI models, and visualizing reports and other output artifacts.
  - **External repositories (in progress):** AutoDW establishes connections to platforms like **Hugging Face**[5] to fetch datasets and models. This functionality is not fully implemented and is planned for future work.
  - **Downstream tasks:** Services such as the **Bias Detector (T3.5), Concept Drift (T3.5),** and **XAI (T3.6)** produce reports that are persisted within AutoDW and linked to specific experiment contexts.
  - **ETD-Hub (WP2):** Provides ethics-related discussion content for policy recommendations.
- **Data consumers:**
  - **AutoML engine (T3.4):** Retrieves datasets and model artifacts for training, fine-tuning, and evaluation.
  - **Task Manager (T3.2):** Consumes event notifications from AutoDW to trigger and coordinate the execution of asynchronous services.
  - **Semantic Knowledge Graph (T3.3):** Accesses ETD-Hub data to build structured representations and relationships for SPARQL querying.

2.1.2 Service implementation and data access

The AutoDW is implemented as a high-performance **FastAPI**[6] application that serves as a unified gateway to the underlying storage layers. Access is provided through a RESTful API surface, running at http://<host>:8000, organized into domain-specific routers (e.g., `/datasets`, `/ai-models`, `/bias-reports`).

Key technical features:

- **Folder and multi-format support:** AutoDW supports both single-file and folder-based uploads (ZIP). This is critical for complex AI models (e.g., PyTorch packages, TensorFlow directories) and multi-file datasets. For folders, the system automatically handles ZIP extraction and preserves internal directory structures. More details on Table 3.

[5] https://huggingface.co/
[6] https://fastapi.tiangolo.com/

Table 3. Single file vs folder upload details

Single file upload	Folder upload (ZIP)
• **Endpoint:** POST /datasets/upload/{user_id} • **Supports:** CSV, Excel, JSON, images, videos, etc. • **Features:** - Automatic metadata extraction (columns, row count, data types) - File type detection - Encoding detection (UTF-8, Latin-1, etc.) - File hash generation for integrity	• **Endpoint:** POST /datasets/upload/folder/{user_id} • **Supports:** ZIP archives with multiple files • **Features:** - Preserve folder structure - Handle hundreds of files - Simplified metadata (file list, sizes, types) - Perfect for complex datasets

- **Automatic metadata extraction:** The AutoDW automatically extracts metadata that describe a dataset that has been uploaded. This works for both single file and folder upload and here are the details of the extracted metadata:
    - **For single files** (e.g. CSV/Excel/JSON): Column names - automatically detected, Row count - total records identified, Data types - per-column type detection, File statistics - size, encoding, hash, Memory usage - calculated for analysis.
    - **For folder uploads**: File inventory - list of all files with paths, File sizes - individual and total, File types - extensions and MIME types, Structure preservation - maintains original folder hierarchy.

  An example of the automatic metadata extraction results for a single CSV file dataset upload for the compliance alert detection technical use case is depicted on **Appendix B1**.

- **Automatic versioning:** AutoDW employs an automatic versioning system (v1, v2, v3, etc.) for every dataset and model upload. This allows users to track evolution and prevents accidental data overwrites. Furthermore, the warehouse performs an **automatic 70-15-15 split** (Train, Test, Drift), which is stored in a reserved version (v2) to support downstream concept-drift monitoring (T3.5) and AutoML tasks. The original uploaded version of the dataset is stored under v1, and any modifications are automatically stored on v2, v3, etc. Typically, if a dataset is suitable for the concept drift flow (dependent on the number of samples in the dataset) then the automatic split will be stored under v2, and any subsequent modifications (e.g. bias mitigation transformations) will be stored on v3 and so on. An example of how this automatic versioning and splitting looks like inside MinIO for the example of single CSV file compliance alerts technical use case is shown on Figure 3. In the upper part of Figure 3 there is a snapshot of AutoDW (from MinIO's UI) showing full path of single file upload and its automatic versioning options. In the middle part of the Figure 3 there is a snapshot of

AutoDW (from MinIO's UI) showing contents inside v1 (original single CSV file upload for TUC of compliance alerts). Finally, on the lower part of the Figure 3 there is a snapshot of AutoDW (from MinIO's UI) showing contents inside v2 (automatic 70-15-15 split of dataset for train/test/drift).

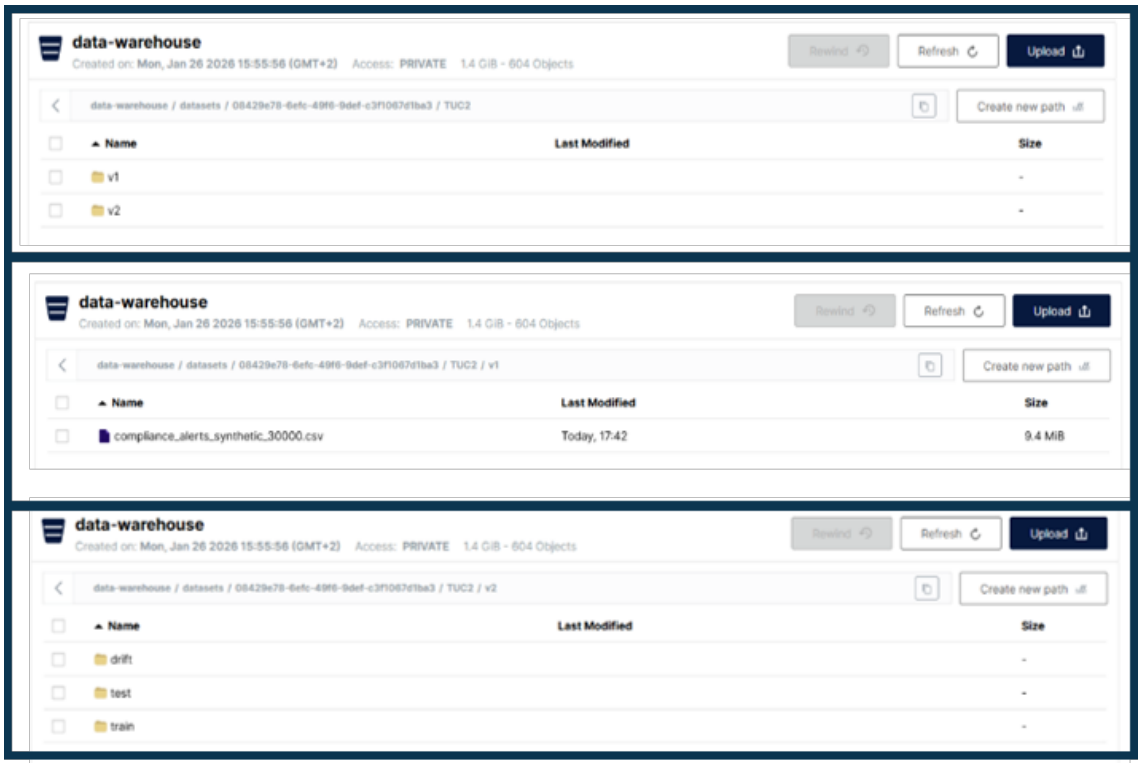

Figure 3. Snapshot of AutoDW (from MinIO's UI)

- **Unified API layer and discovery:** The API provides endpoints not only for upload/download but also for asset discovery (e.g., `GET /datasets/{user_id}/{dataset_id}/files`), allowing consumers to list and selectively download specific files from within a larger artifact.

The AutoDW API is organized into domains which are briefly presented on Table 4. Detailed technical specifications for every endpoint are provided in **Appendix A**.

Table 4. AutoDW API domains and their purpose.

API Domain	Purpose	Key Features
`/datasets`	Data ingestion	Single-file and ZIP/folder support; automatic 70-15-15 train/test/drift splitting; version-specific downloads; optional split-based download (e.g. ?split=train|test|drift).
`/ai-models`	Model management	Multi-file artifacts (e.g. .pkl + label encoders); framework-specific metadata (ONNX, PyTorch, etc.); single-file and folder (ZIP) upload; versioning and tagging.
`/bias-reports`	Fairness storage	Storage for dataset bias analysis reports; user and dataset scoped; Kafka events on save.
`/transformation-reports`	Transformation metadata	Storage for data transformation/bias mitigation reports; version-scoped (e.g. v2); flexible report payload (arbitrary JSON).
`/concept-drift-reports`	Concept drift experiments	Storage for Concept drift reports and retrained model artefacts obtained after the simulation experiment
`/xai-reports`	Explainability storage	HTML XAI reports in MinIO; model vs data explanation; beginner/expert levels; versioned by dataset and model; Kafka events on upload; download by report type and level.
`/user-files`	Supporting docs	Storage for chat attachments (PDF, MD, TXT) with user-scoped access.
`/etd-hub`	Ethical AI forum	Themes (case studies), questions, answers, votes, experts; filtering and statistics; structured for ethical AI discussion. Provides access to the ETD-Hub data to the downstream services (e.g. SemKG input).
`/graphdb`	Knowledge-graph access	SPARQL endpoint configs (stored in MongoDB); SELECT and UPDATE execution; connection testing and health; optional init from backup; repository-scoped queries.

2.2 Types and sources of data stored in AutoDW

AutoDW manages a heterogeneous set of data types, each tailored to specific roles within the ALFIE pipeline. Below is a structured description of the primary data categories.

2.2.1 Metadata and orchestration data (MongoDB and local storage)

Stored as flexible NoSQL documents, this data defines the state and context of the ALFIE ecosystem.

- **Dataset and model cards:** JSON entities containing descriptive metadata (e.g. name, ID, user_id), provenance information, and pointers to physical files in object storage.
- **Experiment contexts:** Links dataset versions to specific model versions and their subsequent evaluation results (Bias, XAI), ensuring a continuous chain of custody for every AI asset.
- **Workflow events:** Persistent logs of Kafka-driven events (e.g., `automl-trigger-events`, `bias-complete-events`), which are used by the Task Manager to track pipeline progress and handle retries.

2.2.2 Binary and large-scale artifacts (MinIO)

These represent the "heavy" data assets required for training and evaluation.

- **Datasets:** Supporting both single file (common is tabular use cases), but also supporting multi-file structures (most common on vision use cases). AutoDW handles file uploading as files meaning there is no limit to specific file types for datasets. Any file will be saved on the MinIO. AutoDW includes robust handling for various encodings (UTF-8, Latin-1, cp1252) to prevent corruption during ingestion.
- **AI models**: Multi-format model artifacts including `.pkl` (Scikit-learn), `.onnx`, `.pth` (PyTorch), and `.h5` (TensorFlow). These are often stored as versioned folders containing weights, label encoders, and configuration files.
- **Service outputs**: HTML or JSON reports generated by the **Bias Detector** and **XAI** services, preserved for auditing and user review.

2.2.3 Ethical and Semantic Knowledge (GraphDB & MongoDB)

- **ETD-Hub data:** A dedicated MongoDB collection stores discussion content from the ethics platform.
- **Knowledge Graph (RDF triplets)**: Semantic relationships and structured representations of project data optimized for SPARQL queries, enabling cross-domain reasoning over datasets and ethical policies. The AutoDW serves as the storage for the work being carried out by T3.3 in regards to generating the Semantic Knowledge Graph of ALFIE platform and so on.

2.2.4 User-supplied documentation

- **User files:** Supporting documents such as `.txt`, `.md`, or `.pdf` (e.g., chatbot attachments or technical READMEs) with a 20MB limit per file, managed via the `/user-files` endpoint.

2.3 Data availability matrix (producer - consumer mapping)

This section delineates the data lifecycle within the ALFIE AutoDW. It distinguishes between services that **produce** data (ingestion and generation) and those that **consume** it to drive the automated pipeline. Table 5 summarizes the data flow within the updated AutoDW ecosystem.

Table 5. AutoDW data flows within ALFIE

Data Type	Producer (WHO)	Consumer (WHO/WHAT)	Purpose
Raw Dataset (v1)	End User/ External	Bias Detector	Serves as the baseline for initial fairness and bias analysis.
70-15-15 Split (v2)	AutoDW (Internal)	AutoML / Concept Drift	Automates the division of data into training, testing, and drift-monitoring subsets.
Bias Report	Bias Detector	Agentic Core / User	Acts as a decision gate to determine if the data is sufficiently balanced for training.
Model Artifacts	AutoML Consumer	XAI / Predictor	Stores model weights (e.g., .pkl, .pth) and encoders required for deployment.
XAI reports	XAI Service	UI / Expert Users	Provides human-interpretable explanations for model decisions and feature importance.
Knowledge Graph	ETD-Hub / AutoDW	SPARQL Endpoint	Semantic search across the platform.
Workflow Events	AutoDW API/ rest of Wp3/Wp4 service workers	Task Manager	Orchestrates the end-to-end pipeline through event-driven triggers.
User Files	End User	Chat/ Experts	Provides supplementary context such as READMEs or legal documentation.

2.4 Event-Driven orchestration and Kafka integration

The ALFIE platform utilizes an event-driven architecture to coordinate complex AI workflows. The AutoDW acts as a central hub in this ecosystem, not only storing data but also producing and consuming Kafka events that trigger subsequent stages of the pipeline, such as bias detection, AutoML training, and concept drift analysis.

2.4.1 Workflow event lifecycle

The orchestration follows a pattern where each service listens for specific "trigger" events, performs its task, and then emits a "completion" event back to the bus. Table 6 presents all the currently available registered Kafka events and their producing and consuming services

as well as the functional trigger behind the event. Note that the Agentic Core runs tasks and receives updates through the Task Manager service. The interaction between Agentic Core, Task Manager and its workers is described in deliverable D3.3.

Table 6. Kafka events table

Event Topic	Producing Service	Consuming Service	Functional Trigger
bias-detection-trigger-events	Agentic Core / UI	Bias Detector	Triggered upon user request; starts the fairness audit.
bias-detection-complete-events	AutoDW API	Agentic Core	Produced when a bias report is saved; signals the end of the fairness audit.
automl-trigger-events	Agentic Core	AutoML Consumer	Triggered upon user request; starts model training.
automl-complete-events	AutoDW API	Agentic Core	Produced when a new model has been trained and stored.
concept-drift-trigger-events	Agentic Core	Concept Drift Service	Triggered to monitor performance on the drift data split and optionally retrain the model.
concept-drift-complete-events	Concept Drift Service	Agentic Core / AutoDW	Signals completion of drift analysis and any model retraining; can trigger downstream steps (e.g. XAI) or user notification.
xai-trigger-events	Agentic Core	XAI Consumer	Triggered upon user request; starts explainability report generation.
xai-complete-events	AutoDW API	Agentic Core	Produced when an XAI report is saved; signals the end of explainability generation; Agentic Core may notify the user.

2.4.2 Kafka event structure examples

Kafka messages in the ALFIE ecosystem are structured as JSON payloads containing the necessary context (User ID, Dataset ID, Task ID and Dataset Version) to allow consumers to fetch the relevant files from the AutoDW API.

Example 1: Bias completion event (bias-detection-complete-events**)** This event is produced by the AutoDW when a bias detector finishes its analysis. It includes critical metadata like the target column to ensure the AutoML service knows which feature to predict.

```
{
  "task_id": "bias_task_abc123",
  "event_type": "bias-detection-complete",
  "timestamp": "2025-02-10T14:30:00.000000Z",
  "output": {
          "bias_report_id": "507f1f77bcf86cd799439011",
          "dataset_id": "drowsiness_dataset",
          "dataset_version": "v2",
          "user_id": "user123",
          "mitigated_dataset_version": "v3",
          "transformation_report_id": "69a0771d27a9013b5f812fce"
  },
  "failure": null
}
```


Example 2: AutoML trigger event (automl-trigger-events**)** This event tells the AutoML consumer to begin training. It specifies if the dataset is a single file or a folder, allowing the consumer to handle ZIP extraction automatically.


```
{
  "task_id": "automl_task_xyz789",
  "event_type": "automl-trigger",
  "timestamp": "2025-02-10T14:35:00.000000Z",
  "input": {
          "user_id": "user123",
          "dataset_id": "drowsiness_dataset",
          "dataset_version": "v2",
          "task_category": "tabular",
          "task_type": "classification",
          "target_column_name": "target",
          "time_budget": "10"
  },
  "failure": null
}
```

3 Data collection from trial use cases

3.1 Trial use cases

Trial use cases are described in detail in D4.1 Initial AutoML platform architecture Section 2.2. This deliverable D3.1 describes the data, its purpose, origin, owner, accessibility, and other relevant information.

Smart website accessibility analyzer use case creates assessment of website accessibility for blind and visually impaired users. The ALFIE system checks compliance of HTML files with the latest Web Content Accessibility Guidelines.

Ethical AI tools for autonomous vehicles to ensure consistency across different drivers use case focus on improving Driver State Monitoring systems. ALFIE AutoML platform is used to develop more inclusive AI models for drowsiness detection, while mitigating biases and fairness issues.

Resolution of IT incidents and the optimization of manufacturing processes use case applies AutoML to streamline the management of Business Partner Screening (BPS) alerts, resolving the bottlenecks and inconsistencies currently caused by overwhelming manual reviews. ALFIE should help create an automated prioritization system and mitigate any identified biases.

3.2 Data collection from website accessibility (UAB use case)

This use case falls under the category of AutoML+. The objective of this use case is to analyse a website to assess its compliance with the latest Web Content Accessibility Guidelines (WCAG) (2.1 as of writing, with the framework designed to accommodate future updates to WCAG 2.2 and beyond) and other established metrics that evaluate the accessibility of a website for a diverse spectrum of users, including people with disabilities. The user provides either the source code or the URL of a webpage, and the AutoML+ engine returns an accessibility report of the given website, highlighting any accessibility issues.

The data for this Use Case will be derived from publicly available websites, creating a robust and diverse dataset that reflects real-world web development practices across different industries, technologies, and design practices. This approach ensures that the model can generate effectively across various website architecture and implementation patterns, while also respecting privacy considerations by focusing exclusively on public-facing content. The dataset may include government websites, educational institutions, news outlets, and corporate sites, providing a comprehensive coverage of different web accessibility challenges and compliance levels encountered in practice.

3.3 Data collection from IRIS use case (CTL use case)

The purpose of the data in this TUC is to train, validate, and stress-test computer vision models and possibly multimodal models that classify driver states (e.g., alert, low vigilance, drowsy) while enabling demographic-aware evaluation under equal opportunity and equalized odds constraints. In the first phase, the use case relies on publicly available datasets that have been screened for research usability and consent compliance. Based on the internal selection, the potential datasets to be used include: (i) University of Texas at

Arlington Real-Life Drowsiness Dataset (UTA-RLDD)[7], (ii) Drowsiness Dataset – University of Peloponnese (Kaggle)[8], (iii) DDD – Driver Drowsiness Detection (Kaggle)[9], and (iv) YawDD[10]. These datasets are publicly accessible for research purposes, range from several gigabytes up to ~111GB (UTA-RLDD), and include RGB driving videos recorded in real vehicles under varying lighting conditions. Data are originally owned by their respective academic creators and distributed under research-oriented or Creative Commons-style licenses. During the public-dataset phase, data will be stored in secured project repositories (e.g., AutoDW or controlled institutional storage), without replicating any personal identifiers beyond what is already anonymized in the original releases.

Across these datasets, annotations typically include categorical labels such as alert, drowsy, non-drowsy, yawning, microsleep, eye-closed, eye-open, or vigilance levels. In most cases, annotations were created by the original dataset authors via manual frame-level or segment-level labeling of recorded videos, sometimes complemented by controlled experimental protocols (e.g., induced fatigue sessions or instructed yawning). For example, UTA-RLDD contains ~30 hours of RGB recordings from 60 participants with vigilance-level annotations, while YawDD includes videos from 90 participants labeled for normal, talking/singing, and yawning behaviors. The University of Peloponnese dataset provides 130 real-car driving videos annotated for yawning and microsleep, and DDD includes drowsy/non-drowsy labels derived from curated video segments. Where available, demographic distributions (e.g., gender balance) will be documented for fairness analysis, but no additional sensitive attributes will be inferred or reconstructed. Screenshots or example frames (without revealing identities) can be included in the final deliverable to illustrate labeling granularity and camera viewpoints.

In the second phase of the trial, if the public datasets are not enough to train ethically balanced models, CTL might collect its own data to complement public datasets and address domain gaps (e.g., local driving conditions, demographic representation). The purpose of this proprietary dataset will be to validate real-world robustness, support bias auditing, and fine-tune models under operational conditions. Data origin will be project-controlled vehicle trials, with CTL acting as data owner under explicitly obtained informed consent. The dataset size will depend on trial duration but is expected to include synchronized inward-facing camera streams, vehicle telemetry, and contextual metadata. Annotations will follow a documented protocol combining manual expert labeling and semi-automated pre-annotation (e.g., eye-closure detection pipelines), with quality-control cross-checking. These data will not be publicly released; they will be securely stored in CTL servers and governed by strict access control and retention policies. Together, the phased approach (public datasets and controlled data collection) ensures methodological rigor, reproducibility, and safe scaling from research benchmarks to real-world deployment conditions.

[7] https://sites.google.com/view/utarldd/home
[8] https://www.kaggle.com/datasets/nikospetrellis/nitymed
[9] https://www.kaggle.com/datasets/ismailnasri20/driver-drowsiness-dataset-ddd
[10] https://qualinet.github.io/databases/video/yawdd_a_yawning_detection_dataset/

3.4 Data collection from IT incidents business partner screening (Bosch use case)

This TUC addresses the challenge of managing Business Partner Screening (BPS) alerts that are continuously generated to monitor third-party relationships for potential violations of the Bosch Code of Business Conduct. Compliance officers currently receive a substantial volume of these alerts, each requiring manual evaluation to determine whether flagged third-party relationships warrant further investigation or corrective action.

The dataset comes from an external provider that obtains a large collection of public sanctions to companies from different countries that collaborate with BOSCH in any way. These sanctions are published by institutional offices and regulatory agencies worldwide, covering a variety of compliance domains including financial regulations, trade restrictions, environmental violations, and other regulatory breaches. The dataset includes a description of each sanction in natural language, typically containing information about the nature of the violation, the issuing regulatory body, the date of publication, and the jurisdiction involved.

The provided dataset is not the original; it has been synthetically generated to anonymize companies and sanctions, following the statistical distributions of the original. This synthetic generation approach ensures that the dataset maintains the same patterns, class distributions, and linguistic characteristics as the real data while eliminating any possibility of identifying actual business partners or specific sanction cases. This anonymization strategy allows the dataset to be stored in the ALFIE AutoML Data Warehouse (AutoDW) without privacy or confidentiality concerns, while still serving as a valid foundation for machine learning model training.

The dataset contains 30,000 instances, with each instance representing a single BPS alert. The primary classification task involves determining whether an alert should be categorized as "Relevant" (requiring further investigation) or can be resolved through standard procedures. Historical classifications performed by Bosch compliance officers serve as the ground truth labels for supervised learning.

The following table (Table 7) shows several instances of the dataset.

Table 7. Example instances from the synthetic BPS dataset

labels	text
3	[ALERT] VendorRisk 2.1 flagged savings account TX-43886468 for round-dollar patterning. Customer: Alex Hernandez at Pioneer Systems. Amount: JPY 9,422.48 via Crypto from Sweden to Poland on 2023-04-03 07:27:07. Risk level: Medium. Escalate to L2 review. Notes: context terms — world china.
3	[ALERT] Case opened in ExpenseEye: supplier invoice TX-63538955 exhibits PEP name similarity detected. Involved party: Orion Metals represented by Jordan Garcia. Route: Italy→UK, channel Wire, value EUR 3,370.59. Risk: Low. Next step: Place account on temporary hold. Notes: context terms — information number.
3	[ALERT] VendorRisk 2.1 detected activity inconsistent with profile — transaction outside customer profile. Reference TX-20625379; product wire transfer; value CAD 2,839.92; corridor Portugal/Poland; channel ACH. Primary contact Sasha Johnson at GlobalTech Ltd. Risk: Critical. File STR/SAR draft for review. Notes: context terms — entities the.
3	[ALERT] CoreBank EU flagged current account TX-10308755 for counterparty on watchlist. Customer: Jamie Silva at Acme Logistics. Amount: USD 35,087.18 via Card from UK to Netherlands on 2025-05-27 09:21:07. Risk level: Medium. Request enhanced due diligence. Notes: context terms — further information.

3	[ALERT] VendorRisk 2.1 flagged expense claim TX-96641001 for unusual transaction velocity. Customer: Cameron Ivanov at Pioneer Systems. Amount: CAD 9,851.32 via Wire from Canada to UAE on 2023-07-14 23:50:07. Risk level: High. Create case and assign to queue. Notes: context terms — entity jan.
3	[ALERT] KYC Nexus detected activity inconsistent with profile — structuring near reporting threshold. Reference TX-34985169; product supplier invoice; value USD 34,677.94; corridor Germany/Singapore; channel ACH. Primary contact Dana Johnson at Helios Energy. Risk: Medium. Create case and assign to queue. Notes: context terms — federation jan.

4 Relevant ethical and legal frameworks and documents

4.1 Purpose of ethical and legal documents within the AutoML framework

Within the ALFIE framework, ethical and legal documents serve as a foundational mechanism for ensuring that ML-based solutions are not only technically functional but also aligned with established ethical principles and legal requirements. These documents translate high-level ethical guidelines, regulatory frameworks, and legal obligations into structured reference material that can be systematically applied during system operation, namely during the generation of ethical assessment of the user's ML use case. These documents are used to ground large language model (LLM) responses and internal reasoning processes in verifiable, external sources rather than relying solely on model priors.

By integrating ethical and legal documents directly into the reasoning loop of the Agentic Core, ALFIE ensures that compliance considerations are not treated as an external or post-hoc process, but as an integral part of decision-making and response generation throughout the system lifecycle.

4.2 Document processing

The ethical and legal documents enhance ethics report creation via the use of retrieval-augmented generation (RAG) technique. First, source documents are preprocessed, embedded into vector representations, and stored in a vector database to enable semantic similarity search.

Next, during the process of creating the ethics report, the Agentic Core queries the most relevant document segments to the description of the user's ML use case. The retrieved content is then injected into the LLM's context window as an explicit prior, constraining and informing the model's reasoning when generating ethical assessments and ensuring that outputs are grounded in the retrieved ethical and legal source material rather than solely in the model's internal knowledge.

4.3 Integration and storage

The ethical and legal documents are stored in the Agentic Core PostgresSQL database with pgvector extension that enables storing text embeddings and semantic search. The documents are ingested to the database during the installation procedure.

Currently, Agentic Core Docker image bundles a script that performs the ingestion. Using Langchain library, it loads PDF files in a specified directory in the local filesystem, splits them into text chunks, creates OpenAI embeddings of the chunks, and stores them in a specified document collection in the database. The ingestion is performed by running the Agentic Core Docker container with a specific command, bind-mapping a folder with the documents, and configuring an environment variable to the folder path in the container.

In the future, we are considering storing the documents in the AutoDW.

4.4 Identified relevant ethical and legal documents

Ethics By Design and Ethics of Use Approaches for Artificial Intelligence [1]

This document, published by the European Commission, is highly relevant for ALFIE because it directly supports the project's objective of developing fair, ethical, and legally compliant AI systems. It provides a structured and operationalized framework for implementing ethical AI across the entire system lifecycle, including design, development, and deployment phases. The framework contains principles and requirements as well as actionable tasks that can be used to identify risks in the ML use case and give useful recommendations. Additionally, the document is aimed for European legal and value context, which is another aspect relevant for ALFIE. The document builds on 'Ethics guidelines for trustworthy AI' [2].

Ethics guidelines for trustworthy AI [2]

Like the document above, this document is published by the European Commission. It is very relevant because it supports the project's objective to build ethical, trustworthy, and compliant AI systems. The document provides a detailed, structured framework of key principles and requirements for ethical AI development and deployment that align with EU values and regulations, another important aspect for ALFIE. Moreover, the document includes an operational assessment list of concrete questions that can be used to evaluate and guide ML use cases from an ethical and compliance perspective, even though the list is in a piloting phase.

5 Semantic Knowledge Graph

5.1 What is Semantic Knowledge Graph (SemKG)

The ALFIE Semantic Knowledge Graph (SemKG) is a core component that consumes discussions from the ETD-Hub and transforms them into a formally structured, machine-interpretable graph. In this graph, entities, relations, and metadata are represented using explicit semantics grounded in ontologies and standardised vocabularies. The SemKG encodes knowledge using RDF (Resource Description Framework), typed classes, formally defined properties, and logical constraints, enabling reasoning, validation, and interoperability across systems [3, 4, 15].

The SemKG is implemented as an RDF-based graph where extracted entities and relations are mapped to a controlled ontology, assigned stable IRIs, and enriched with provenance, evidence, and confidence metadata. Relations are modelled using a reified pattern (n-ary design), allowing each assertion to be associated with contextual information such as source text, extraction confidence, discourse identifiers, and expert votes. This design ensures that the graph does not merely connect nodes but encodes semantically meaningful, traceable, and auditable knowledge statements. SemKG further incorporates formal constraint validation using SHACL (Shapes Constraint Language), ensuring structural correctness, datatype consistency, and semantic integrity prior to deployment. The SemKG combines ontology-driven modelling, reified relations, and SHACL validation to ensure governed semantic representation.

5.2 SemKG vs traditional knowledge graphs

Most traditional knowledge graphs primarily model static entities and institutional artefacts, the SemKG in ALFIE formalises deliberative discourse itself as a first-class semantic structure. Through the TQAVE schema (Themes, Questions, Answers, Votes, Expert), the graph explicitly models actor participation, governance artefacts, evidential grounding, and decision-making dynamics as semantically typed constructs within an OWL-compliant RDF framework. This discourse-native modelling approach distinguishes the SemKG from traditional graph representations that typically treat such interactions as unstructured metadata rather than governed semantic objects.

5.3 Role of ontologies, semantics, and formal vocabularies

Ontologies provide the formal backbone of a Semantic Knowledge Graph by defining the classes, properties, and constraints that govern how knowledge is represented. In the ALFIE SemKG, the ontology specifies core conceptual categories (e.g., entities, relations, evidence, votes), their hierarchical organisation, and the semantics of the relationships that connect them. This formalisation ensures that extracted information is not stored as loosely connected triples but as semantically typed, logically consistent knowledge structures grounded in RDF and OWL standards [3, 4].

Formal vocabularies play a central role in ensuring interoperability and machine interpretability. The SemKG leverages standard semantic web constructs such as rdf:type, rdfs:label, owl:sameAs, and typed literals (xsd:string, xsd:integer, xsd:float, xsd:dateTime) to guarantee unambiguous interpretation across systems. External linking to authoritative

resources (e.g., Wikidata entities via owl:sameAs) strengthens semantic grounding and enables cross-graph integration. Controlled vocabularies such as SKOS are used for modelling categorical scales (e.g., severity, likelihood, risk levels), supporting structured reasoning and hierarchical inference [6].

5.4 Benefits of SemKGs for AI, automation, and explainability

A Semantic Knowledge Graph provides structural and semantic guarantees that are particularly valuable for AI-driven systems. In the SemKG pipeline, all entities and relations are represented using deterministic IRIs, formally typed classes, and validated constraints, enabling machine agents to operate over a stable and semantically governed knowledge layer. Unlike unstructured data stores, the SemKG ensures that every assertion is explicitly modelled with subject, predicate, object, and contextual metadata, thereby supporting reliable querying, reasoning, and downstream automation.

For AI and automated decision-support components, the SemKG offers three principal advantages. First, it enables structured reasoning over typed relationships through RDFS/OWL semantics and rule-based inference, allowing multi-hop knowledge expansion and semantic aggregation. Second, it embeds provenance and confidence modelling directly within reified relation structures, linking each knowledge claim to its source, evidence text, extraction metadata, and expert votes. This design supports traceability and reproducibility, which are essential in automated analytical workflows. Third, constraint validation through SHACL guarantees structural correctness prior to deployment, reducing the propagation of malformed or semantically inconsistent knowledge into AI services [5].

Explainability is further strengthened by the n-ary modelling pattern used in the SemKG. Rather than representing relations as opaque edges, each assertion is represented as a first-class node connected to evidence and confidence information. This allows AI components to justify outputs by retrieving supporting evidence and contextual metadata from the graph. Such explainable modelling aligns with emerging requirements for transparent and accountable AI systems, where traceable knowledge representation and explicit provenance are fundamental design principles.

5.5 Role of the Semantic Knowledge Graph in the AutoML platform

Within the ALFIE framework, the SemKG serves as an additional grounding source for ethical report generation by structuring and exposing domain knowledge derived from discussions in the ETD-Hub. The graph encodes concepts related to ethics, ethical issues, risks, recommendations, and their interrelations, transforming unstructured forum content into a machine-interpretable representation.

Its role in the ethics reporting process is to provide contextual, experience-based knowledge that complements other grounding mechanisms, enabling the Agentic Core to anchor ethical assessments in previously articulated concerns, patterns, and considerations discussed by domain stakeholders.

The SemKG is stored in a graph database within the AutoDW and is interfaced through AutoDW's API with SPARQL queries.

The Agentic Core interacts with the SemKG during ethics report generation to retrieve information relevant to a specific machine learning use case described by the user. This

retrieved information is used as prior context for the large language model (LLM), biasing the generation process toward known ethical risks, recommendations, and considerations associated with similar or related use cases. Two categories of retrieved information can be distinguished: (1) structured semantic knowledge stored in the SemKG, including entities, relationships, and concepts derived from ETD-Hub content; and (2) the raw ETD-Hub discussions from which this knowledge was extracted.

The retrieval algorithm applied to the SemKG, *GraphRAG*, is structured as a multi-step process. The first step of this process focuses on candidate entity selection. Given a textual description of the user's machine learning use case, semantic similarity measures or keyword-based search is used to identify and score graph entities that are most relevant to the described task. These entities act as entry points into the graph and represent ethical topics, issues, or concepts potentially applicable to the use case.

In the second step of the *GraphRAG* algorithm, the algorithm expands each selected entity by exploring its local neighborhood within the graph. For each candidate node, the algorithm traverses one or two hops to collect directly and indirectly connected entities and relations. This results in a set of subgraphs that capture contextual information surrounding the initial candidates, including related risks, mitigation strategies, and ethical considerations that may not be explicitly mentioned in the original user description.

The final step of the *GraphRAG* algorithm combines ranking and textual conversion of the extracted subgraphs. Each subgraph is transformed into a textual representation by grouping together relationships around their subject entities and serializing them into descriptive text fragments. These textual representations and in turn their respective underlying subgraphs are ranked based on semantic similarity to the user's ML use case description.

The most relevant subgraphs are then injected into the LLM's context window and used as grounding input for generating the final ethical report. In addition to serving as contextual knowledge for ethical assessment generation, each retrieved subgraph may contain metadata references to the original ETD-Hub models from which its entities and triples were derived. These references enable the Agentic Core to retrieve the corresponding ETD-Hub artifacts, such as themes, questions, and discussion answers, and provide them alongside the SemKG-derived knowledge within the LLM's context window to further inform and influence the ethical assessment generation process.

Figure 4 illustrates the complete ethical assessment generation workflow implemented as a tool the main agent can invoke to generate ethical assessment based on the user's ML task description. The component highlighted in red indicates the part of the system where the SemKG is utilized, specifically the GraphRAG process responsible for retrieving and incorporating knowledge from the semantic knowledge graph into the assessment generation pipeline.

Future iterations of this retrieval algorithm can be improved by more explicitly leveraging the SemKG's ontology and structural properties. Neighborhood exploration can be made more targeted by incorporating ontology-aware traversal rules that prioritize specific relationship types, entity classes, or ethical dimensions relevant to machine learning use cases, rather than applying a uniform hop-based expansion strategy. In addition, subgraph ranking can move beyond text-only semantic similarity by exploiting existing entities' relationships inherent to graph structure.

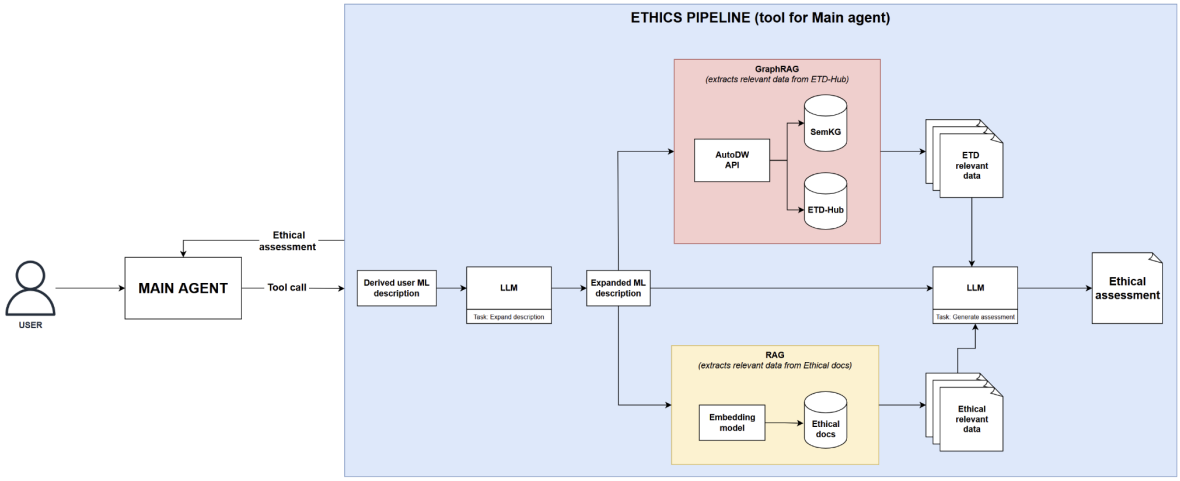

Figure 4. Implementation of ethical assessment generation workflow

5.6 Research contributions and state-of-the-art

5.6.1 Related work on Semantic Knowledge Graphs

ALFIE Semantic Knowledge Graphs (SemKGs) build upon foundational research in knowledge representation, ontology engineering, and linked data. Early work on the Semantic Web established RDF and OWL as formal frameworks for representing machine-interpretable knowledge with logical semantics [8, 3, 4]. These standards enabled knowledge graphs to move beyond simple graph connectivity toward semantically grounded, interoperable knowledge structures capable of supporting automated reasoning.

Large-scale knowledge graphs such as DBpedia [9], YAGO [10], and Wikidata [11] demonstrated the feasibility of structured, entity-centric semantic graphs at web scale. However, they typically prioritise encyclopaedic or factual knowledge and often represent relations as binary assertions without explicit modelling of contextualised discourse, evidence structures, or governance processes.

More recent research has explored knowledge graph construction pipelines combining information extraction, entity linking, and ontology alignment [12], as well as validation frameworks using SHACL to ensure graph integrity [13]. Additionally, n-ary relation patterns have been proposed to model contextualised statements in RDF when binary relations are insufficient [14]. The ALFIE SemKG aligns with this body of work by integrating automated extraction, ontology-driven typing, SHACL-based constraint validation, and reified modelling for contextualised assertions. Unlike general-purpose encyclopaedic knowledge graphs, the SemKG focuses on discourse-native modelling.

5.6.2 Advances over existing approaches

SemKG advances the state-of-the-art by shifting the focus of knowledge modelling from static, encyclopaedic entity representation toward structured modelling of deliberative processes and governance dynamics. While established semantic knowledge graphs such as DBpedia, YAGO, and Wikidata primarily capture factual assertions about entities and their relationships [9, 10, 11], the ALFIE SemKG formalises discourse itself as a first-class

semantic construct. This represents a conceptual and architectural departure from entity-centric graph paradigms.

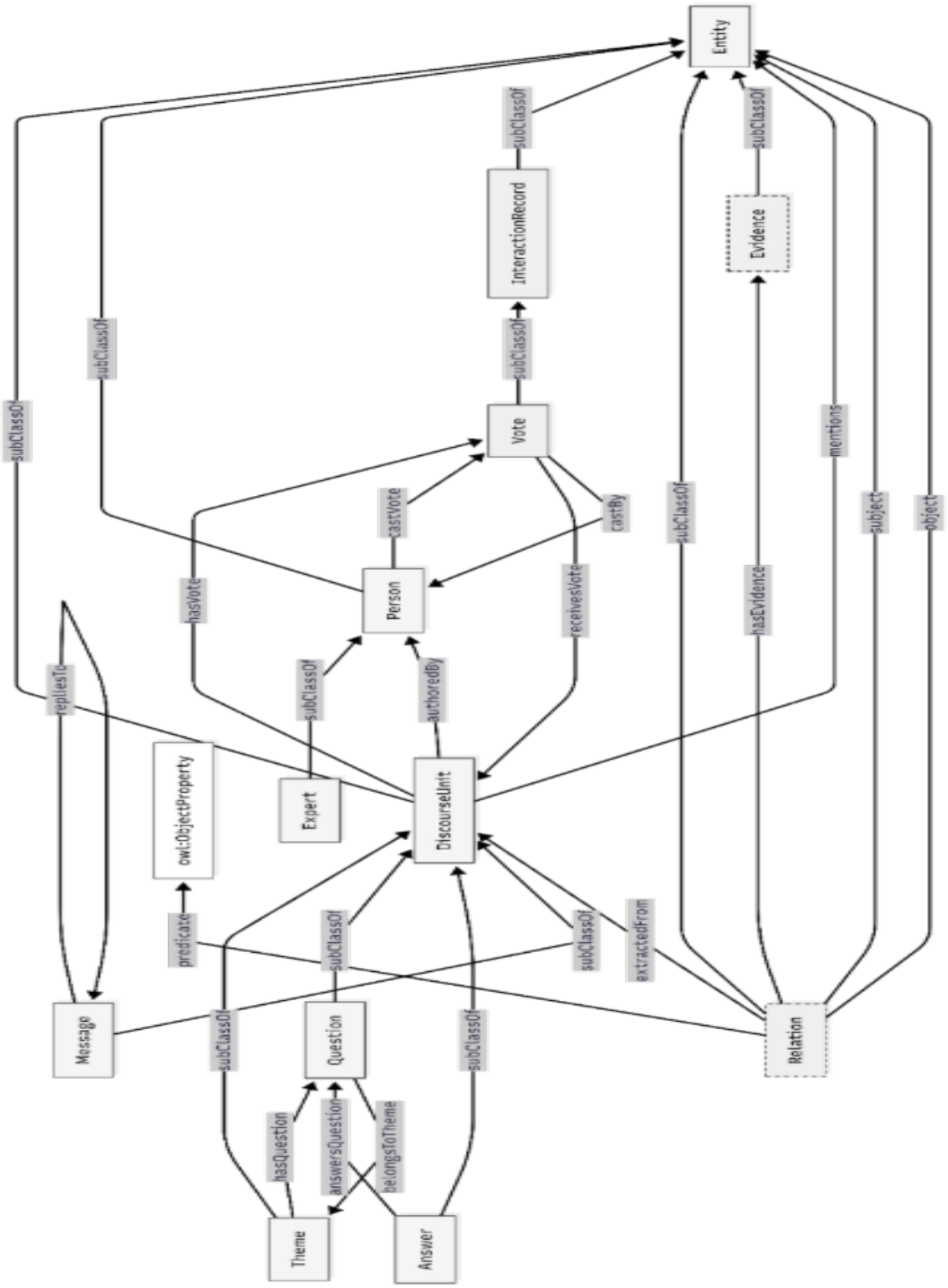

Figure 5. TQAVE high-level ontology schema.

Central to this advancement is the TQAVE (Theme, Question, Answer, Vote, Expert) schema, which models deliberative structures as semantically typed components within an OWL-compliant RDF framework (Figure 5). Rather than treating discussions, expert inputs, or voting artefacts as peripheral metadata, the SemKG explicitly encodes them as governed classes and relations. Actor participation, evidential grounding, extraction confidence, and expert voting are attached to reified relation nodes, enabling contextualised, explainable assertions. This discourse-native modelling approach operationalises governance interactions as structured semantic knowledge, rather than as loosely attached annotations.

From a methodological perspective, the SemKG integrates automated semantic extraction with ontology-driven typing, deterministic IRI generation, SHACL-based constraint validation, and reified n-ary modelling. Existing knowledge graph construction pipelines often emphasise entity linking and triple extraction [12] but do not uniformly enforce structural constraints at deployment time or model contextual provenance as a core design principle. By embedding SHACL validation into the pipeline and requiring each relation to be associated with evidence and provenance metadata, the ALFIE approach strengthens data integrity and traceability beyond many extraction-driven systems [13]. Furthermore, the SemKG is designed for explainable AI integration. Each assertion is represented as a semantically typed relation node connected to evidence, confidence, discourse identifiers, and expert votes. This design enables AI components to retrieve justification chains directly from the graph, supporting transparent reasoning and auditable decision support. In contrast to traditional knowledge graphs that prioritise large-scale data aggregation, the ALFIE SemKG prioritises governance, explainability, and deterministic reproducibility as primary design objectives.

5.7 Sources and construction of the SemKG

5.7.1 Input data sources

The Semantic Knowledge Graph (SemKG) is constructed from coherent and semi-structured discourse data supplied from AutoDW. The datasets are normalised into a JSON structure consisting of Theme, Questions, Answers, Votes, and Experts.

5.7.1.1 Theme

Theme represents structured topical domains under which questions and answers are organised. In the pipeline, themes are treated as first-class semantic entities and are assigned stable IRIs during graph construction. Their inclusion ensures that all discourse artefacts are contextualised within a controlled conceptual hierarchy. Each thematic record contains descriptive metadata that is processed by the entity extraction module and mapped to the ontology layer. Theme structures function as anchoring nodes within the graph, supporting classification, grouping, and downstream reasoning over domain-specific knowledge segments.

5.7.1.2 Questions and answers

Question–answer pairs constitute the primary source of knowledge assertions within the SemKG. Questions represent structured prompts within thematic contexts, while answers contain substantive content from which entities and relations are extracted. The semantic extraction pipeline identifies candidate entities, resolves them against known identifiers, and

derives relational assertions that are subsequently formalised as reified Relation nodes. Each extracted assertion is associated with provenance metadata, including discourse identifiers, textual offsets, and contextual evidence. This modelling ensures that every relation in the graph can be traced to its originating answer content. By structuring question–answer interactions as semantically typed constructs, the pipeline transforms conversational discourse into machine-interpretable knowledge suitable for reasoning and AI consumption.

5.7.1.3 Expert and votes

Expert interactions introduce evaluative and governance-oriented metadata into the SemKG. Votes and expert inputs are modelled as semantically typed constructs connected to reified relation nodes. Rather than storing evaluation outcomes as detached annotations, the pipeline attaches voting artefacts, confidence values, and expert identifiers directly to the knowledge assertions they assess. This design enables the graph to encode not only factual relationships but also deliberative dynamics, confidence assessments, and consensus indicators. By integrating expert participation into the semantic layer, the SemKG supports explainable AI workflows where decisions and inferences can be contextualised with evaluative provenance.

5.7.2 Semantic knowledge extraction pipeline

SemKG is constructed through a deterministic, multi-stage semantic information extraction pipeline (Figure 6). The pipeline transforms raw textual inputs into a structured RDF graph using large language models (LLMs), ontology-aware mapping, entity consolidation, and provenance modelling. The extraction workflow consists of: Semantic chunking of textual inputs, Entity extraction, Relation extraction, Normalisation and consolidation, Ontology mapping and semantic typing and RDF serialisation with SHACL validation.

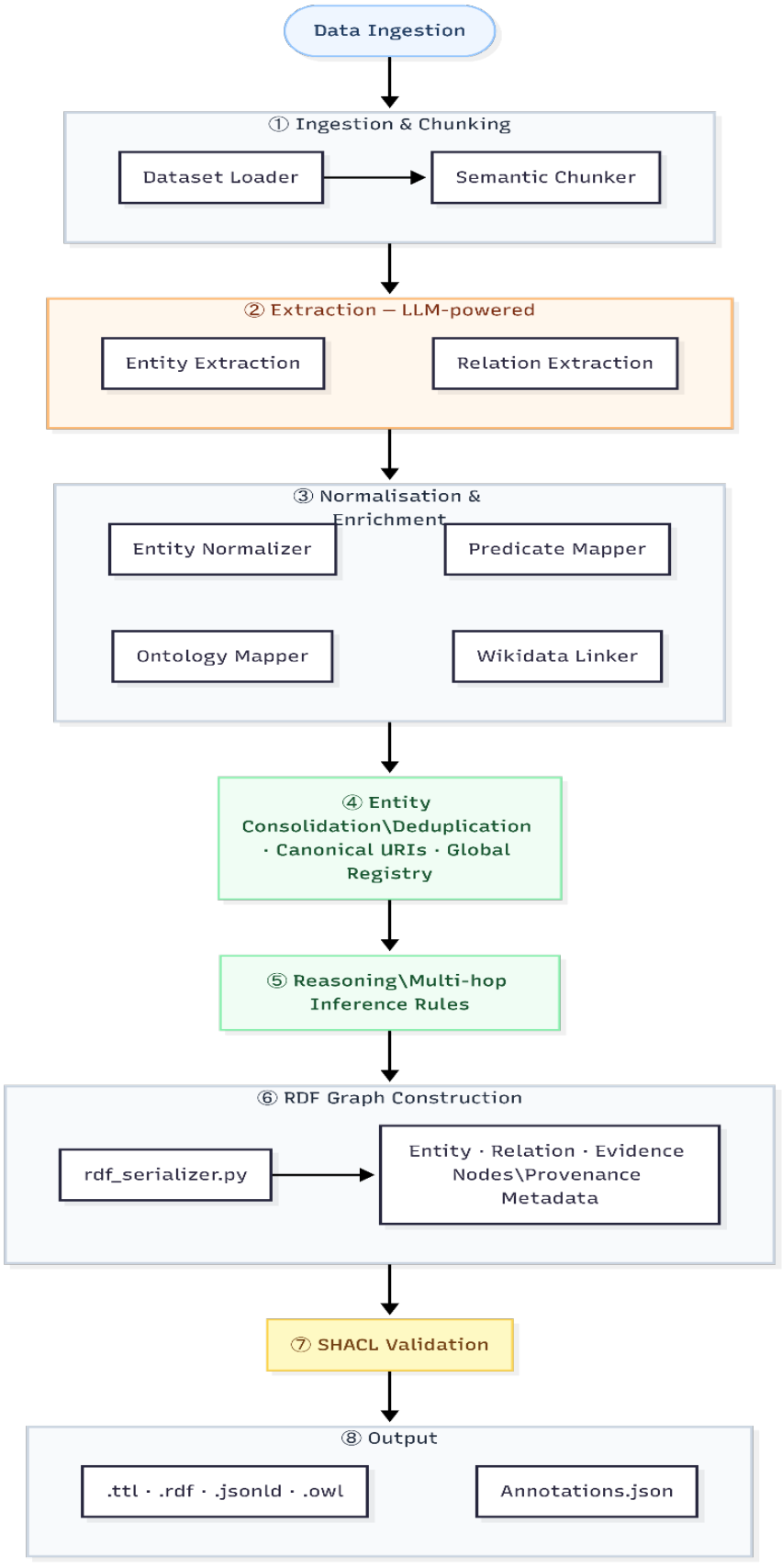

Figure 6. Semantic knowledge graph workflow

5.7.2.1 Entity extraction

Entity extraction is performed using an LLM-based structured extraction approach. The pipeline invokes a JSON-mode LLM model (gpt-4o-mini) with a constrained schema to extract entities from thematic text, questions, answers, and related discourse units. The extractor returns structured JSON objects that include the entity surface (the text span in the source content), the entity_type, a confidence score, and optional ontology hints that may support later semantic alignment. Following extraction, a post-processing stage is applied. This stage performs confidence threshold filtering to remove low-confidence results, detects span boundaries in the source text (capturing start_char and end_char offsets), mints stable URIs using deterministic hashing, and attaches metadata such as the chunk index and the extractor type used during processing.

Each extracted entity is represented in the knowledge graph as an instance of etd:Entity. The entity is assigned a canonical IRI of the form etd:<hash>, along with an rdfs:label corresponding to the surface text. Additional controlled datatype properties are attached where appropriate, and the extraction confidence value is preserved as part of the entity metadata. The extractor is designed to prevent hallucinated identifiers by avoiding model-generated IDs, enforce structured output through the constrained JSON schema, maintain deterministic identity generation via hashing, and preserve provenance information for traceability. This stage produces the foundational semantic nodes of the graph, which are subsequently used for relationship construction, ontology alignment, and higher-level reasoning processes.

5.7.2.2 Relation extraction

Relation extraction combines multiple complementary strategies to construct meaningful connections between entities in the knowledge graph. These include LLM-based structured relation extraction using the LLM model (gpt-4o-mini), lightweight rule-based extraction methods such as mentions and co-occurrence detection, and deterministic structural relation generation derived from the platform's discourse structure (for example, Question → Theme, Answer → Expert, and Vote → Answer relationships). In the LLM-based relation extraction stage, the model receives the textual chunk being processed together with a list of allowed entity URIs that were previously extracted from that context.

The model returns structured JSON specifying the source_id, target_id, relation, confidence, and evidence supporting the relation. To prevent hallucinated relations, the system strictly constrains the model so that only entity URIs present in the current extraction context may be used as relation endpoints. Each relation is represented in the graph as a reified etd:Relation node rather than as a simple RDF triple. This design allows the system to capture additional semantic metadata and supports more advanced reasoning capabilities. In particular, reification enables explicit modelling of confidence values, attachment of provenance information, tracking of textual evidence supporting the relation, and future extensibility for reasoning or validation processes.

In addition to the LLM-based extraction, the pipeline generates lightweight and structural relations deterministically. Lightweight relations include mentions, which link entities to discourse units in which they appear, and relatedTo, which captures entity co-occurrence within the same chunk of text. Structural relations encode the logical structure of the discourse and platform interactions. These include belongsToTheme, answersQuestion,

authoredBy, and endorses, the latter representing vote interactions associated with answers. All relations are stored as reified nodes with consistent metadata, ensuring uniform provenance tracking, confidence representation, and compatibility with downstream graph processing and reasoning stages.

5.7.2.3 Normalisation and consolidation

After extraction, entities undergo a normalisation and consolidation process designed to prevent duplication and reduce semantic drift across the graph. This stage ensures that semantically identical entities are consistently represented and reused throughout the pipeline. The pipeline first performs surface-form normalisation, which includes operations such as case folding and whitespace normalisation to standardise textual representations of entity labels. Following this, an identity key is constructed for each entity using a combination of the normalised surface form, the entity type, and any available ontology hints. This identity key provides a stable basis for determining whether two extracted entities represent the same underlying concept.

A registry-based consolidation mechanism is then applied across documents and processing batches. The entity registry maintains canonical mappings of identity keys to entity URIs, enabling the system to detect previously seen entities and reuse their identifiers rather than creating duplicates. When multiple entities are determined to represent the same canonical concept, URI merging is performed so that references resolve to a single consistent identifier. The entity registry enforces several important guarantees. It ensures that each semantic concept is represented by a single canonical URI, enables entity reuse across documents and discourse units, and supports stable graph growth as new data batches are processed over time. Through this consolidation process, the pipeline reduces redundancy, improves semantic coherence within the graph, and strengthens the consistency of downstream relation extraction and reasoning stages.

5.7.3 Ontology mapping and alignment

Ontology mapping is applied after extraction to align entities and relations with controlled vocabulary terms and ensure semantic consistency across the knowledge graph. This stage translates the flexible outputs produced during extraction into formally defined classes and predicates within the ETD ontology and, where appropriate, external vocabularies. The pipeline maps free-form relation labels generated by the LLM to canonical predicates within the controlled namespace (e.g., etd:<predicate>). At the same time, entity types are aligned with canonical ETD classes so that extracted entities conform to the ontology's class hierarchy. When relevant, references to external vocabularies are preserved rather than replaced, allowing the system to maintain compatibility with established semantic standards. During this process, ontology_uri metadata is attached to entities and relations to record the ontology class or predicate to which they have been aligned. The system also supports alignment with external knowledge bases. When suitable correspondences are identified, external identifiers may be linked using constructs such as owl:sameAs, enabling integration with resources such as Wikidata and other linked data ecosystems. In the most recent pipeline run, the relation alignment rate is 1.0, indicating that all relation predicates were successfully mapped to the controlled ETD namespace.

Entity alignment is partially complete, reflecting the current coexistence of canonical ETD classes alongside externally referenced class types. Ontology mapping provides several

important guarantees for the resulting graph. It ensures strict predicate control, maintains namespace consistency across the dataset, enables formal semantic interpretation of graph structures, and supports interoperability with external vocabularies and linked data resources.

5.7.4 Provenance, evidence, and confidence modelling

SemKG explicitly models provenance and evidential grounding through the use of reified evidence nodes. This design ensures that every semantic assertion within the graph can be traced back to its originating textual context and extraction process. For each extracted relation, a corresponding etd:Evidence node is created to capture the supporting information associated with that assertion. The textual evidence supporting the relation is stored using the etd:evidenceText property, while the etd:discourse_id identifies the specific source context from which the relation was derived. In addition, chunk-level metadata is preserved to maintain a link to the precise segment of text processed during extraction. The system also records the extractor type responsible for generating the relation, enabling differentiation between LLM-based, rule-based, or structural extraction methods.

Confidence values associated with relations are stored using the xsd:float datatype to enable probabilistic filtering and evaluation. Integer-based provenance fields are strictly typed using xsd:integer to maintain compliance with SHACL validation constraints and ensure consistent datatype usage across the graph. This provenance-aware modelling approach provides several important capabilities. It guarantees traceability for every semantic assertion, enabling users and systems to understand the origin of each relation. It supports explainability for downstream AI systems by preserving the textual evidence underlying graph assertions. It also enables confidence-aware filtering, allowing applications to prioritise higher-confidence knowledge, and provides a structured framework for tracking provenance throughout the pipeline.

5.8 Semantic model and ontology design

5.8.1 Core conceptual model of the SemKG

SemKG is grounded in a formally defined ontology that specifies the conceptual structure, relationships, and constraints governing the representation of discourse-derived knowledge. The ontology is implemented using RDF and OWL constructs and is programmatically instantiated through the schema generation layer of the pipeline. Its design reflects three guiding principles: semantic clarity, structural validation, and explainable modelling. The ontology provides the formal vocabulary used by the extraction and serialisation modules to map discourse artefacts into typed entities and relations. It also defines domain-specific classes, object properties, and datatype properties that support provenance modelling, expert participation, and contextualised assertions. Constraint validation through SHACL ensures that all instances adhere to the structural requirements defined by the ontology.

5.8.2 Classes and class hierarchy

The core conceptual model of the SemKG is discourse-native and revolves around semantically typed entities and contextualised assertions. Each extracted knowledge statement is represented as a reified Relation node rather than a simple binary triple. This

design enables the attachment of contextual metadata such as confidence scores, discourse identifiers, chunk-level provenance, and expert votes.

The conceptual model therefore separates:

1. The logical assertion (subject–predicate–object),
2. The contextual representation of that assertion,
3. The provenance and evaluative metadata associated with it.

This layered design supports traceability, explainability, and reasoning.

5.8.3 Object properties and datatype properties

The ontology distinguishes between object properties and datatype properties to maintain clear semantic separation between relationships that connect resources and those that attach literal values. Object properties link one resource to another resource within the graph, while datatype properties link resources to typed literals that represent attributes or metadata. Object properties are used to represent structural and semantic relationships between nodes. These include the subject, predicate, and object properties used in the reified relation modelling approach adopted by the graph. Additional object properties include hasEvidence, which links a relation to its supporting evidence node, hasVote, which connects discourse elements to vote interactions, and owl:sameAs, which is used to express alignment with equivalent entities in external knowledge bases.

Datatype properties capture scalar or textual attributes associated with entities, relations, and evidence. These include confidence values stored as xsd:float, structural indexing fields such as chunk_index, chunk_total, start_char, and end_char stored as xsd:integer, and the evidenceText property stored as xsd:string. Additional datatype properties include wikidataScore, represented as xsd:float, and ingestedAt, which is stored as xsd:dateTime when a valid ISO timestamp is available. Datatype integrity is enforced during the serialisation stage to ensure that only valid lexical forms are written to the graph. This prevents malformed literals from entering the dataset and ensures compatibility with RDF processing tools. SHACL shapes are applied as an additional validation layer, constraining property cardinality, node kinds, and datatype requirements to guarantee structural and semantic consistency across the knowledge graph.

5.8.4 Reified relations and N-ary modelling

The SemKG adopts a reified modelling strategy in which each assertion is represented as a first-class Relation node rather than as a simple binary triple alone. Instead of expressing knowledge only in the form Entity A – predicate – Entity B, the graph materialises an explicit relation resource that stores the subject, predicate, and object as properties of that node. This follows the established RDF pattern for modelling contextualised or n-ary relations, as recommended by the W3C for cases where additional information must be attached to a statement. Such a design is particularly important in the SemKG because knowledge claims are not treated as isolated facts: they carry provenance, confidence, evidence, voting context, and inference metadata. Reification therefore makes each claim individually inspectable, auditable, and governable, which is essential for explainable AI use and for any downstream service that must assess the reliability and origin of a relation [7].

5.8.5 Use of external ontologies and vocabularies

To strengthen interoperability and semantic clarity, the SemKG also incorporates established Semantic Web vocabularies rather than relying solely on project-specific terms. RDF and RDFS are used for typing and labelling, OWL supports class declarations and equivalence assertions such as owl:sameAs, SHACL provides structural validation constraints, and SKOS can be used where controlled concept schemes are appropriate. This layered reuse of standard vocabularies makes the graph easier to interpret, validate, and integrate with external semantic systems. In addition, external entity grounding is supported through owl:sameAs links to Wikidata IRIs where alignments are available, thereby improving semantic anchoring and enabling cross-graph interoperability beyond the local platform environment [11].

5.9 Initial Semantic Knowledge Graph approach

The initial SemKG approach was based on applying Named Entity Recognition to ETD-Hub discussion text and passing the annotated output to the external CASPAR system for graph generation. Although this enabled basic entity-centric graph construction, it had significant limitations: relations were not fully semantically reified, ontology alignment was limited, namespaces and external links were not tightly controlled, and there was no formal SHACL validation, deterministic IRI minting, provenance-aware modelling, or integrated reasoning support. These constraints reduced the expressiveness, auditability, and reliability of the resulting graph, and they motivated the development of the current SemKG pipeline, which adds structured LLM-based extraction, canonical ontology mapping, reified relation modelling, evidence and confidence representation, SHACL validation, deterministic URI generation, and reasoning capabilities.

5.10 Semantic reasoner and inference capabilities

The SemKG incorporates a schema-aligned semantic reasoner, implemented in reasoner.py, to extend the graph beyond directly extracted facts while preserving ontological discipline. The reasoner operates only over canonical schema predicates, normalises predicate variants before inference, filters relations by a minimum confidence threshold, and does not mint non-schema properties. Its reasoning process is iterative and bounded by a configurable maximum number of rounds, but it stops early whenever no further valid relations can be produced. Within the pipeline, this reasoning layer serves three linked purposes: multi-hop knowledge completion, governance and risk closure, and consistency checking. In practice, this means the graph can propagate relations across intermediate entities, derive governance and risk-related consequences, and support the detection of contradictory assertions, all while remaining grounded in the controlled SemKG schema.

To achieve this, the reasoner combines several complementary strategies. It applies RDFS/OWL-style reasoning such as transitivity for predicates including relatedTo, uses, trainedOn, containsSubject, partOfDomain, and includes, and symmetry for predicates such as relatedTo and associatedWith. It also supports rule-based composition, for example deriving uses from trainedOn ∘ containsSubject, exhibitsRisk from trainedOn ∘ hasRisk, and broader governance propagation such as violates ∘ relatedTo → violates, with inferred confidence bounded by rule-specific caps. In addition, schema-specific rules enable closure

over domain and risk structures, such as inferring affectedBy, partOfDomain, mitigatedBy, and further hasRisk relations from established semantic patterns.

All inferred relations are deterministically minted, annotated with reasoning metadata such as inference type and iteration, and deduplicated against both asserted and previously inferred triples to ensure reproducibility and schema compliance. Alongside inference, a conflict detection component identifies contradictory predicate pairs between the same source and target, such as compliesWith versus violates or supports versus opposes, and records these as structured conflict annotations rather than introducing new graph relations.

5.11 Infrastructure and deployment architecture

The SemKG is serialised in multiple standard RDF representations, including Turtle (.ttl), RDF/XML (.rdf), JSON-LD (.jsonld), and OWL (.owl), ensuring portability across standards-compliant semantic platforms. This multi-format export strategy allows the graph to be inspected, validated, exchanged, and deployed in environments that support Linked Data and ontology-based knowledge management. In operational terms, the SemKG is deployed to a GraphDB instance hosted within AutoDW through a configurable client module, with upload behaviour governed by the ENABLE_GRAPHDB_UPLOAD setting. This makes it possible to separate local graph generation and validation from production persistence, while also supporting controlled deployment workflows and batch-oriented upload strategies.

5.11.1 Storage layer (Triple Store / Graph Database)

Once deployed into a triple store such as GraphDB within the AutoDW environment, the knowledge graph becomes accessible through a SPARQL endpoint. This interface enables users and systems to interact with the graph using the standard RDF query language, providing a flexible mechanism for exploring and analysing the stored semantic data. Through the SPARQL endpoint, the graph supports structured semantic queries that retrieve entities, relations, and evidence based on defined patterns. It also enables pattern-based graph traversal, allowing queries to follow chains of relationships across the graph structure. In addition, SPARQL supports aggregation and analytical queries, making it possible to compute statistics, counts, or summaries over graph data. The endpoint also enables federated querying, allowing queries to combine data from the local graph with external RDF sources and linked data repositories.

The graph's structure has been designed to support SPARQL-friendly querying. All relations are modelled as reified etd:Relation nodes rather than simple triples, which allows queries to directly access relation metadata such as confidence and provenance. Within these nodes, the subject, predicate, and object are explicitly represented, enabling clear and consistent query patterns. Evidence supporting each relation is linked through the etd:hasEvidence property, allowing queries to retrieve both the relation and its supporting textual justification.

5.11.2 Reified relation query pattern

Because relations are represented as first-class etd:Relation nodes, SPARQL queries follow a consistent multi-step structure when retrieving relational information from the graph. Rather than directly querying simple subject–predicate–object triples, queries first identify the relation node and then access the properties that define the relationship. In practice, this results in a four-step query pattern. The query first matches the etd:Relation node representing the relationship of interest. It then retrieves the associated etd:subject, followed

by the etd:predicate, and finally the etd:object. This structure mirrors the traditional triple model while preserving the additional metadata attached to the reified relation.

Additional joins may be included depending on the analytical requirements of the query. For example, the etd:confidence property can be retrieved or filtered to apply confidence thresholds when selecting relations. Queries may also join to etd:hasEvidence in order to retrieve the supporting textual evidence associated with the relation. Where inference processes are applied, inference metadata such as inference_type can be queried to distinguish between relations that were directly extracted from source text and those generated through reasoning or inference mechanisms. By structuring relations as reified nodes with explicitly modelled components, the graph maintains a uniform query structure across both extracted and inferred knowledge. This consistency simplifies SPARQL query design and supports reliable analytical and reasoning workflows.

5.11.3 Confidence and evidence-aware querying

The Semantic Knowledge Graph (SemKG) supports confidence-aware filtering directly at query time. Because confidence values are stored using the xsd:float datatype, agents and analytical services can apply threshold constraints within SPARQL queries using FILTER expressions. This allows consumers of the graph to selectively retrieve relations that meet a specified confidence level, enabling more reliable downstream reasoning, analytics, or decision-support workflows. Evidence nodes linked through the etd:hasEvidence property provide access to the contextual grounding of each relation. Through these links, queries can retrieve the supporting evidence text, the associated discourse identifiers that reference the source context, and chunk-level provenance metadata describing the segment of text from which the relation was extracted. This design enables query results to be accompanied by contextual justification, allowing users and automated systems to inspect the supporting evidence behind each semantic assertion. As a result, analytical outputs derived from the graph can remain transparent, traceable, and explainable The SemKG supports confidence-aware filtering directly at query time. Since confidence values are stored as xsd:float, agents and analytical services can apply threshold constraints using SPARQL FILTER expressions.

5.11.4 Inference visibility

Inferred relations are stored using the same schema as extracted relations and are annotated with inference-specific metadata. This unified representation ensures that inferred knowledge integrates seamlessly with extracted knowledge within the graph structure. Through this design, SPARQL queries can distinguish between inferred and extracted assertions by inspecting the associated inference metadata. Queries may also filter results based on the specific type of inference that generated the relation, enabling analysts or systems to focus on particular reasoning processes. When intermediate reasoning chains are recorded, queries can further inspect these structures to understand how an inferred relation was derived from existing knowledge within the graph. Because inferred triples conform to the same schema as extracted relations and are validated prior to persistence, they remain fully compatible with the standard query interface. As a result, both inferred and extracted knowledge can be accessed, filtered, and analysed using the same SPARQL patterns and query mechanisms.

5.11.5 Controlled namespace and predicate governance

All predicates in the Semantic Knowledge Graph are restricted to the controlled ETD namespace following the ontology mapping and normalisation stages. During these stages, any free-form or model-generated relation labels are aligned with canonical predicates defined within the ETD ontology, ensuring that the graph uses a consistent and governed vocabulary. This restriction ensures that uncontrolled or ad hoc predicates do not appear in query results. By enforcing a controlled namespace, the system maintains stable query formulation across different versions of the graph, preventing schema drift that could otherwise break existing queries. It also guarantees deterministic predicate usage, meaning that the same semantic relationship is always represented by the same canonical predicate. This governance model provides an important layer of stability for services interacting with the SPARQL endpoint. By maintaining strict predicate control, platform APIs and analytical services can rely on consistent graph semantics over time, improving long-term compatibility, maintainability, and reliability of applications built on top of the knowledge graph.

5.12 Consumption of the SemKG by platform services

The SemKG is designed to be consumable by downstream platform services, particularly agent-based components that require stable identifiers, inspectable semantics, and traceable provenance. Each entity is assigned a deterministic IRI generated through stable hashing, allowing agents to reference the same real-world concept consistently across repeated pipeline runs. This gives the graph a dependable identity layer for retrieval, linking, caching, and orchestration. In addition, relations are modelled as reified etd:Relation nodes rather than simple subject–predicate–object assertions. This enables services to inspect relation-level metadata such as confidence scores, extractor provenance, and whether a fact is asserted or inferred, thereby supporting more selective and context-aware graph consumption.

6 Conclusions

This deliverable established the initial technical foundations for the ALFIE platform across four core areas: data management, trial use case data, ethical grounding, and semantic representation.

First, the AutoML Data Warehouse was implemented as the platform's central storage and orchestration layer. It now supports versioned dataset and model management, automated metadata extraction, dataset splitting, and event driven communication with downstream services. These capabilities provide the traceability and data handling required for later AutoML, fairness, explainability, and drift workflows.

Second, the initial datasets for the three trial use cases were collected and documented. For each case, the deliverable described the data origin, purpose, structure, and annotation scheme. This provides a clear baseline for running the first full training and evaluation cycles in ALFIE.

Third, the project identified and integrated relevant ethical and legal documents. These materials were embedded and stored to enable retrieval during ethics assessment generation, ensuring that the platform can ground its outputs in established European guidance on trustworthy AI.

Fourth, SemKG constitutes the formal semantic backbone of the ALFIE platform. It transforms heterogeneous ETD-Hub discussion data, comprising themes, questions, answers, expert interactions, and votes into a structured, ontology-aligned, machine-readable representation. Through deterministic URI minting, reified relation modelling, SHACL validation, and evidence-backed assertions, the SemKG ensures that all semantic statements are structurally compliant, ontology-controlled, Confidence-scored, Provenance-aware and Reproducible. It supports agentic capabilities within ALFIE by ensuring that AI components operate over a stable and interpretable semantic vocabulary.

Together, these results deliver a coherent and operational starting point for the next development stages of WP3, enabling reliable data workflows, grounded ethical assessments, and consistent semantic interpretation across the ALFIE platform.

6.1 Future work

6.1.1 External dataset support

Currently, the ALFIE platform lacks external dataset ingestion pipeline. This functionality will be a significant usability improvement, because the user would not have to download the dataset from an external platform and then upload it to ALFIE. Furthermore, it will improve traceability and data lineage, because the dataset source will be registered in AutoDW, whereas currently the information is lost after downloading and uploading the dataset.

A preliminary work has been done to support HuggingFace [5] datasets, but the solution has not been integrated with AutoDW yet. The solution consists of a REST API service, which downloads information about datasets and models from HuggingFace[5] and ingest them to Elasticsearch service. This enables keyword, semantic, and hybrid search options.

6.1.2 Semantic Knowledge Graph

The current SemKG implementation provides a structurally validated, ontology-aligned, evidence-backed semantic layer for the ALFIE platform. While the system performs well, several extensions are planned to enhance semantic depth, reasoning capability, alignment coverage and agent integration.

References

[1] European Commission, Ethics by Design and Ethics of Use Approaches for Artificial Intelligence, Brussels, Belgium: European Commission, 2021. [Online]. Available: https://ec.europa.eu/info/funding-tenders/opportunities/docs/2021-2027/horizon/guidance/ethics-by-design-and-ethics-of-use-approaches-for-artificial-intelligence_he_en.pdf

[2] High-Level Expert Group on Artificial Intelligence (AI HLEG), Ethics Guidelines for Trustworthy AI, Brussels, Belgium: European Commission, 2019. [Online]. Available: https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

[3] W3C. (2014). *RDF 1.1 Concepts and Abstract Syntax*.

[4] W3C. (2012). *OWL 2 Web Ontology Language Document Overview*.

[5] W3C. (2017). *SHACL – Shapes Constraint Language*.

[6] W3C. (2009). *SKOS Simple Knowledge Organization System Reference*.

[7] W3C. (2006). *Defining N-ary Relations on the Semantic Web*.

[8] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). *The Semantic Web*. Scientific American.

[9] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). DBpedia: A nucleus for a web of open data. *ISWC 2007*.

[10] Suchanek, F. M., Kasneci, G., & Weikum, G. (2007). YAGO: A core of semantic knowledge. *WWW 2007*.

[11] Vrandečić, D., & Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10).

[12] Paulheim, H. (2017). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web Journal*.

[13] Knublauch, H., & Kontokostas, D. (2017). Shapes Constraint Language (SHACL). W3C Recommendation.

[14] W3C. (2006). *Defining N-ary Relations on the Semantic Web*.

[15] Uwasomba, C.F., Lee, Y., Yusoff, Z. and Chin, T.M. (2022). Ontology-based methodology for knowledge acquisition from groupware. Applied Sciences, 12(3), p.1448.

Appendices

Appendix A: AutoDW technical API reference

This appendix provides a comprehensive reference for the AutoDW REST API. To support multi-tenancy and auditability, all endpoints utilize the {user_id} path parameter and implement automatic semantic versioning.

A.1 Dataset Management (`/datasets`)

Table 8 - A.1. This domain handles the ingestion, versioning, and automated splitting of datasets.

Method	Endpoint	Description	Key Parameters
`POST`	`/datasets/upload/{user_id}`	Uploads a single dataset file. Automatic versioning and metadata extraction.	`dataset_id`, `name`, `file`
`POST`	`/datasets/upload/folder/{user_id}`	Uploads a ZIP archive of a dataset folder. Extracts and preserves structure.	`dataset_id`, `name`, `zip_file`
`GET`	`/datasets/search/{user_id}`	Search datasets with various filters.	`Query(search in name, description, and dataset_id)`, `tags`, `file_type`
`GET`	`/datasets/search/{user_id}/{dataset_id}`	Get latest dataset metadata (from MongoDB) by `user_id` and `dataset_id`.	`user_id`, `dataset_id`
`PUT`	`/datasets/{user_id}/{dataset_id}`	Update dataset metadata.	`user_id`, `dataset_id`
`GET`	`/datasets/{user_id}`	Get all datasets metadata for a user with pagination.	`user_id`, `skip`, `limit`
`GET`	`/datasets/{user_id}/{dataset_id}/download`	Fetches the latest version. For folders, returns a ZIP unless `filename` is specified.	`filename` (optional), `split` (train/test/drift)
`GET`	`/datasets/{user_id}/{dataset_id}/files`	Lists all files contained within a specific dataset version.	`version` (optional)
`DELETE`	`/datasets/{user_id}/{dataset_id}`	Removes a dataset and its associated physical files from MinIO.	`version` (optional)

A.2 AI Model Management (/ai-models)

Table 9 - A.2. Manages the lifecycle of trained AI models, including support for multi-file model packages (e.g., weights + encoders).

Method	Endpoint	Description	Key Parameters
POST	/ai-models/upload/single/{user_id}	Uploads a single model binary (e.g., .pkl, .onnx).	model_id, name, framework
POST	/ai-models/upload/folder/{user_id}	Uploads a model folder as a ZIP. Critical for PyTorch/TensorFlow artifacts.	model_id, name, zip_file
GET	/ai-models/search/{user_id}	Search AI models with various filters.	Query(search in name, description, and model_id), framework, model_type,tags
GET	/ai-models/{user_id}/{model_id}/metadata	Retrieves the Model Card (JSON) including training metrics and framework info.	version (optional)
GET	/ai-models/{user_id}/{model_id}	Get AI model metadata (from MongoDB).	user_id, model_id
PUT	/ai-models/{user_id}/{model_id}	Update AI model mdetadata (from MongoDB).	user_id, dataset_id
GET	/ai-models/{user_id}/{model_id}/download	Downloads the model artifact.	version (optional)
DELETE	/ai-models/{user_id}/{model_id}	Delete an AI model and all its files.	version (optional)
GET	/ai-models/{user_id}	Get all AI models for a user with optional filtering.	framework, model_type,tags

A.3 Bias, mitigation transformations, concept drift and XAI Reporting (`/bias-reports`, `/transformation-reports`, `/concept-drift-reports`, `/xai-reports`)

Table 10 - A.3.1 Endpoints for storing results from downstream ALFIE services (dataset bias reports T3.5).

Method	Endpoint	Description	Key Parameters
POST	/bias-reports/	Create or update a bias report for a dataset. Sends Kafka completion event when X-Task-ID header is present.	Body: user_id, dataset_id, dataset_version, report (JSON). Optional: target_column_name, task_type, is_folder, file_count, transformation_report_id. Header: X-Task-ID (optional).
GET	/bias-reports/{user_id}/{dataset_id}	Get bias report for a dataset (latest or specific version).	user_id, dataset_id, version (query, optional).
GET	/bias-reports/{user_id}/{dataset_id}/all	Get all bias reports for a dataset (all versions).	user_id, dataset_id.
DELETE	/bias-reports/{user_id}/{dataset_id}	Delete bias report(s) for a dataset.	user_id, dataset_id, version (query, optional; omit to delete all).

Table 11 - A.3.2 Endpoints for storing results from downstream ALFIE services (dataset bias mitigation (transformations) reports T3.5).

Method	Endpoint	Description	Key Parameters
POST	/transformation-reports/	Create or update a transformation report (e.g. bias mitigation) for a dataset version.	Body: user_id, dataset_id, version, report (JSON).
GET	/transformation-reports/{user_id}/{dataset_id}/{version}	Get transformation report for a specific dataset version.	user_id, dataset_id, version.
DELETE	/transformation-reports/{user_id}/{dataset_id}/{version}	Delete transformation report for a dataset version.	user_id, dataset_id, version.

Table 12 - A.3.3 Endpoints for storing results from downstream ALFIE services (concept drift reports T3.5).

Method	Endpoint	Description	Key Parameters
POST	/concept-drift-reports/	Create or update a concept drift report (JSON).	Body: user_id, dataset_id, dataset_version, model_id, model_version (JSON).
GET	/concept-drift-reports/{user_id}/{dataset_id}/{model_id}	Get concept drift report for a dataset/model (specific versions or latest).	user_id, dataset_id, model_id. Optional version
DELETE	/concept-drift-reports/{user_id}/{dataset_id}/{model_id}	Delete concept drift report(s). Omit version params to delete all reports for this dataset/model.	user_id, dataset_id, model_id. Optional version

Table 13 - A.3.4 Endpoints for storing results from downstream ALFIE services (XAI reports T3.6).

Method	Endpoint	Description	Key Parameters
POST	/xai-reports/upload/{user_id}	Upload an XAI report HTML file. Sends Kafka completion event when X-Task-ID header is present.	user_id (path). Form: file (HTML), dataset_id, dataset_version, model_id, model_version, report_type (model_explanation | data_explanation), level (beginner | expert). Header: X-Task-ID (optional).
GET	/xai-reports/{user_id}/{dataset_id}/{model_id}	Get all XAI reports for a dataset and model, optionally filtered by versions.	user_id, dataset_id, model_id. Query: dataset_version, model_version (optional).
GET	/xai-reports/{user_id}/{dataset_id}/{model_id}/{report_type}/{level}	Get metadata for a specific XAI report (by type and expertise level).	user_id, dataset_id, model_id, report_type, level (path). Query: dataset_version, model_version (optional; default latest).
GET	/xai-reports/{user_id}/{dataset_id}/{model_id}/{report_type}/	Return the report HTML for viewing in the browser.	user_id, dataset_id, model_id, report_type, level (path). Query: dataset_version, model_version (optional; default

	{level}/view		latest).
DELETE	/xai-reports/{user_id}/{dataset_id}/{model_id}/{report_type}/{level}	Delete a specific XAI report (specific versions or latest).	user_id, dataset_id, model_id, report_type, level (path). Query: dataset_version, model_version (optional; omit to delete latest).

A.4 User Files (/user-files)

Table 14 - A.4 Handles auxiliary project documentation.

Method	Endpoint	Description	Key Parameters
POST	/user-files/upload/{user_id}	Upload a file (e.g. chat attachment). Returns metadata including file_id. Allowed types: TXT, MD, CSV, JSON, XML, PDF, DOC, DOCX, ODT, RTF (max 20 MB).	user_id (path). Form: file (required), name, description, project_id (optional).
GET	/user-files/{user_id}	List files for a user with pagination. Optional filter by project/conversation.	user_id (path). Query: project_id, skip, limit.
GET	/user-files/{user_id}/{file_id}	Get metadata for a single file (no file content).	user_id, file_id (path).
GET	/user-files/{user_id}/{file_id}/download	Download file content (e.g. for parsing or summarization by other services).	user_id, file_id (path).
DELETE	/user-files/{user_id}/{file_id}	Delete a file and its metadata.	user_id, file_id (path).

A.5 GraphDB data (/graphdb)

Table 15 - A.5 Controls for Graphdb (create, update, delete configs on specified Semantic Knowledge Graph) execute SPARQL queries.

Method	Endpoint	Description	Key Parameters
POST	/graphdb/configs	Create a new GraphDB configuration (SPARQL endpoints, credentials, repository). Configs stored in MongoDB.	Query: created_by. Body: name, description, select_endpoint, update_endpoint, username, password, repository_name, timeout, max_retries (optional).
GET	/graphdb/configs	List all GraphDB configurations with	Query: skip, limit.

		pagination.	
GET	/graphdb/configs/{config_id}	Get a single GraphDB configuration by ID (metadata only; password not returned).	config_id (path).
PUT	/graphdb/configs/{config_id}	Update a GraphDB configuration. Partial updates supported.	config_id (path). Body: any of name, description, select_endpoint, update_endpoint, username, password, repository_name, status, timeout, max_retries (all optional).
DELETE	/graphdb/configs/{config_id}	Delete a GraphDB configuration.	config_id (path).
POST	/graphdb/configs/{config_id}/test	Test connection to the GraphDB repository (endpoints, auth, repository existence).	config_id (path).
POST	/graphdb/query	Execute a SPARQL query against the repository identified by a config.	Body: config_id, query, query_type (SELECT, INSERT, DELETE, UPDATE), timeout (optional).
GET	/graphdb/configs/{config_id}/query-examples	Get example SPARQL queries (SELECT and UPDATE) for the configured repository.	config_id (path).

A.6 ETD-Hub data (/ETD-Hub)

Table 16 - A.6 Integration with ETD-Hub data (saving, retrieving and deleting ETD-Hub entities).

Method	Endpoint	Description	Key Parameters
POST	/etd-hub/themes	Save a new theme (case study / ethical AI topic).	Body: theme_id, name, description, expert_id, problem_category, model_category, domain_category.
GET	/etd-hub/themes	List themes with optional filtering by category.	Query: skip, limit, problem_category, model_category, domain_category.
GET	/etd-hub/themes/{theme_id}	Get a theme by ID. Increments view count.	theme_id (path).
GET	/etd-hub/themes/{theme_id}/with-questions	Get a theme with its associated questions.	theme_id (path).

PUT	/etd-hub/themes/{theme_id}	Update a theme.	theme_id (path). Body: name, description, problem_category, model_category, domain_category (all optional).
DELETE	/etd-hub/themes/{theme_id}	Delete a theme.	theme_id (path).
POST	/etd-hub/questions	Save a new question.	Body: question_id, title, body, expert_id, theme_id.
GET	/etd-hub/questions	List questions with optional filtering.	Query: skip, limit, theme_id, expert_id.
GET	/etd-hub/questions/{question_id}	Get a question by ID. Increments view count.	question_id (path).
GET	/etd-hub/questions/{question_id}/with-answers	Get a question with its answers and vote counts.	question_id (path).
PUT	/etd-hub/questions/{question_id}	Update a question (metadata only in current implementation).	question_id (path). Body: title, body (optional).
DELETE	/etd-hub/questions/{question_id}	Delete a question.	question_id (path).
POST	/etd-hub/answers	Create a new answer (or reply when parent_id is set).	Body: answer_id, description, question_id, expert_id, parent_id (optional).
GET	/etd-hub/answers	List answers with optional filtering.	Query: skip, limit, question_id, expert_id, parent_id.
GET	/etd-hub/answers/{answer_id}	Get an answer with vote statistics (upvotes, downvotes).	answer_id (path).
DELETE	/etd-hub/answers/{answer_id}	Delete an answer.	answer_id (path).
POST	/etd-hub/votes	Save or update a vote (up/down) on an answer by an expert.	Body: vote_id, expert_id, answer_id, vote_value.
GET	/etd-hub/votes/{exp	Get a vote by expert and answer.	expert_id, answer_id (path).

	ert_id}/{answe r_id}		
DELETE	/etd-hub/votes/{exp ert_id}/{answe r_id}	Delete a vote.	expert_id, answer_id (path).
POST	/etd-hub/experts	Create a new expert profile.	Body: expert_id, user_id, area_of_expertise, bio, profile_picture (optional).
GET	/etd-hub/experts	List experts with optional filtering.	Query: skip, limit, area_of_expertise, is_deleted.
GET	/etd-hub/experts/{e xpert_id}	Get an expert by ID.	expert_id (path).
GET	/etd-hub/experts/{e xpert_id}/with -stats	Get an expert with statistics (e.g. questions, answers).	expert_id (path).
DELETE	/etd-hub/experts/{e xpert_id}	Delete an expert.	expert_id (path).
POST	/etd-hub/documents	Save a new document (file/metadata linked to theme or expert).	Body: document_id, title, description, expert_id, theme_id (optional).
GET	/etd-hub/documents	List documents with optional filtering.	Query: skip, limit, theme_id, expert_id.
GET	/etd-hub/documents/ {document_id}	Get a document by ID.	document_id (path).
GET	/etd-hub/stats/over view	Get overview counts (themes, questions, answers, experts, documents, votes).	None.

Appendix B: AutoDW data examples

The following JSON file shows the result of the AutoDW automatic metadata extraction service when a single CSV file dataset is uploaded. This example is from the compliance alert technical use case.


```
{
  "dataset_id": "TUC2",
  "user_id": "08429e78-6efc-49f6-9def-c3f1067d1ba3",
  "name": "Synthetic compliance alerts",
  "description": "The dataset includes a description of each sanction in natural language, typically containing information about the nature of the violation, the issuing regulatory body, the date of publication, and the jurisdiction involved.",
  "version": "v2",
  "file_type": "csv",
  "file_size": 9819850,
  "original_filename": "compliance_alerts_synthetic_30000.csv",
  "files": [
    {
      "filename": "compliance_alerts_synthetic_30000.csv",
      "file_path": "datasets/08429e78-6efc-49f6-9def-c3f1067d1ba3/TUC2/v2/train/compliance_alerts_synthetic_30000.csv",
      "file_size": 6873489,
      "file_type": "csv",
      "file_hash": "20c153d85ce4f989cc021413607df3bd",
      "content_type": "text/csv"
    },
    {
      "filename": "compliance_alerts_synthetic_30000.csv",
      "file_path": "datasets/08429e78-6efc-49f6-9def-c3f1067d1ba3/TUC2/v2/test/compliance_alerts_synthetic_30000.csv",
      "file_size": 1473402,
      "file_type": "csv",
      "file_hash": "5946bf7001da5a512540b6cc4b71c4d8",
      "content_type": "text/csv"
    },
    {
      "filename": "compliance_alerts_synthetic_30000.csv",
      "file_path": "datasets/08429e78-6efc-49f6-9def-c3f1067d1ba3/TUC2/v2/drift/compliance_alerts_synthetic_30000.csv",
      "file_size": 1472959,
      "file_type": "csv",
      "file_hash": "051862811031f0c163f0590ec7ac8afd",
      "content_type": "text/csv"
    }
```

```
  ],
  "is_folder": true,
  "columns": [
    "labels",
    "text"
  ],
  "row_count": 30000,
  "data_types": {
    "labels": "int64",
    "text": "object"
  },
  "tags": [],
  "custom_metadata": {
    "split": {
      "train": 21000,
      "test": 4500,
      "drift": 4500
    }
  },
  "created_at": "2026-03-01T15:42:49.255000",
  "updated_at": "2026-03-01T15:42:49.255000",
  "file_hash": null
}
```