LingoMap - An Interactive Semantic Augmentation Workbench

> Submission for the 2025 OMG Semantic Augmentation Challenge

Author: TsaiChen LO
Live Demo: https://lingomap.streamlit.app/
GitHub Repository: https://github.com/tsaichen1o/LingoMap/tree/main

---

1. Project Vision & The OMG Challenge

The 2025 OMG Semantic Augmentation Challenge highlights a critical business need: a vast amount of the world's data is stored in formats like CSV, which lack the rich semantic context necessary for reliable data integration, advanced analytics, and AI applications. This "semantic gap" makes it difficult, time-consuming, and expensive for organizations to unlock the true value of their data assets.

LingoMap was conceived and built to directly address this challenge. It is an intelligent, interactive workbench designed to bridge this semantic gap. Its core mission is to empower users by transforming opaque, tabular data into a rich, interconnected, and machine-readable knowledge graph, following established standards like RDF and ontologies such as FIBO, GeoSPARQL and Schema.org.

2. Core Philosophy: The AI-Powered Trinity

Unlike traditional, manual mapping tools, LingoMap's innovation lies in its synergistic use of three core technologies to replicate and accelerate an expert's workflow:

1. **Automated Data Profiling**: The system first performs a deep statistical analysis of each data column to understand its fundamental characteristics (data types, uniqueness, value distribution).
2. **Retrieval-Augmented Generation (RAG)**: LingoMap queries a pre-built vector knowledge base containing multiple standard ontologies. This provides the AI with a dynamically retrieved, context-aware list of relevant semantic terms for any given column or concept.
3. **Large Language Model (LLM) Reasoning**: An advanced LLM (Google's Gemini) acts as the "brain" of the operation. It synthesizes the data profiles and RAG results to perform complex reasoning tasks, such as conceptualizing entities, defining relationships, and recommending precise property mappings.

3. System Architecture & Technologies

LingoMap is a full-stack Python application built with a modern, modular architecture.

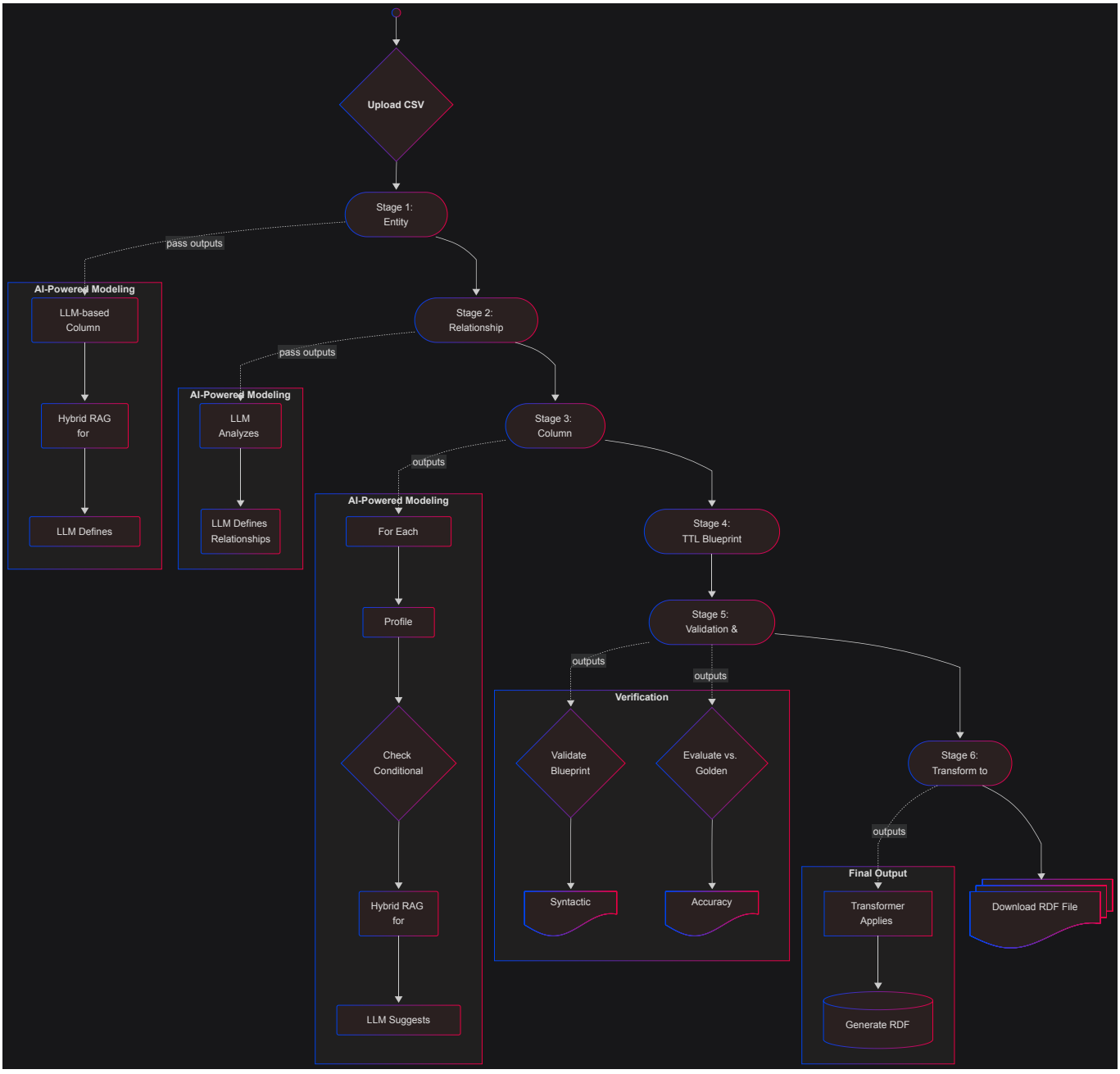
- **Frontend**: Streamlit, providing a reactive, web-based user interface for an interactive workflow.
- **Backend & Orchestration**: A custom `CoreMappingEngine` orchestrates the multi-stage pipeline.
- **Data Analysis**: Pandas for data manipulation and a custom `Data Profiler` module.
- **LLM Engine**: Google Gemini (via the `google-generativeai` library) is used for all generative and reasoning tasks.

- **Vector Database (RAG)**: ChromaDB, configured to run as a service, hosts the vocabulary embeddings. The `rag_search.py` module provides a dedicated interface for querying.
- **Knowledge Representation**: RDFLib is used for generating and handling RDF graphs, with Turtle (`.ttl`) being the primary serialization format.

4. The LingoMap Workflow: A Six-Stage Journey

The user experience is designed as a guided, six-stage journey from a raw CSV file to a fully semanticized RDF graph.

Workflow Diagram



- **Stage 1: Entity Conception**: Upon uploading a CSV, the user clicks "Generate Core Entities". The `CoreMappingEngine` invokes the `column_clusterer`, which now uses an LLM to group columns based on their descriptions and names. Then, for each cluster, `entity_conception.py` uses the **Hybrid RAG** approach (detailed below) to ask the LLM to define a core business entity, like "Financial Institution" or "Physical Address".

- **Stage 2: Relationship Definition**: The user clicks "Define Entity Relationships". The `relationship_definition.py` module takes the list of entities from Stage 1, provides them to the LLM along with a curated list of common connecting properties (like `schema:location` or `fibo-fnd-rel-rel:isSubunitOf`), and asks the LLM to act as a knowledge architect, defining the links between the entities.
- **Stage 3: Detailed Column Mapping**: The user can now analyze columns within each entity group. For each column, the `CoreMappingEngine` orchestrates a deep analysis:
    1. A data profile is generated (`data_profiler.py`).
    2. A check is run to see if it's a "flag" column suitable for a conditional rule (`rule_generator.py`).
    3. If it's regular data, a rich query is created and sent to the `rag_searcher` to find relevant properties from the knowledge base.
    4. The `column_mapping.py` module then uses the **Hybrid RAG** approach to generate a final, highly-contextualized property mapping suggestion from the LLM.
- **Stage 4: TTL Blueprint Generation**: After all mappings are decided, the `ttl_generator.py` module compiles the entities, relationships, and column mappings from the previous stages into a complete, well-structured Turtle (`.ttl`) file. This file is the "semantic blueprint" for the data.
- **Stage 5: Validation & Evaluation**: To ensure quality and rigor, the user can:
    1. **Validate the Blueprint**: The `validate_mapping.py` script checks the generated TTL file for syntactic correctness and structural integrity.
    2. **Evaluate AI Performance**: The `run_evaluation.py` script compares the AI's suggestions against a pre-defined "golden standard" file, calculating and displaying performance metrics like accuracy.
- **Stage 6: Transform CSV to RDF**: The final step. The user clicks "Transform Data to RDF". The generic, rule-driven `ttl2rdf_transformer.py` script is invoked. It parses the AI-generated TTL blueprint from Stage 4 and uses it as an instruction manual to process the original CSV file row-by-row, generating the final, linked RDF graph data, which is then made available for download.

5. Deep Dive: The Hybrid RAG & Prior Knowledge Strategy

A key innovation in LingoMap is its use of a **Hybrid RAG** approach, which combines dynamic, real-time search with static, expert-curated knowledge.

1. **Knowledge Base Creation (Offline Process)**: The process starts with `vocabulary_processor.py`. This script reads multiple standard ontology files (like FIBO, Schema.org, etc.) in formats such as Turtle. For each meaningful class and property, it generates a descriptive text, creates a vector embedding using a `SentenceTransformer` model, and stores this embedding along with its URI and other metadata in a ChromaDB vector database. This creates a rich, searchable knowledge base.
2. **The Hybrid RAG in Action (Online Process)**: During the interactive workflow (e.g., in `entity_conception.py` and `column_mapping.py`), the following happens:
    - **Dynamic Search**: The `rag_search.py` module queries the ChromaDB to retrieve a list of the most semantically relevant terms based on the current context (e.g., column profile).
    - **Static "Prior Knowledge"**: The code also maintains a hard-coded Python list of `RELEVANT_ENTITY_CLASSES` or `RELEVANT_DATA_PROPERTIES`. This list represents a "golden set" of preferred, high-quality terms.

- **Combined Prompting**: The LLM prompt is constructed with **two distinct sections**: one for the dynamic RAG results and one for the static "Highly-Recommended" list. The LLM is explicitly instructed to use the RAG results for context but to give preference to the curated list for its final decision.

This hybrid strategy provides the best of both worlds: the **relevance** of a dynamic search and the **stability and quality** of an expert-curated knowledge list.

6. Conclusion

LingoMap successfully demonstrates a complete, end-to-end workflow for semantic augmentation. By intelligently orchestrating data profiling, a hybrid RAG system, and LLM-powered reasoning, it transforms the complex, manual task of data semanticization into a guided, semi-automated, and verifiable process. It directly addresses the OMG Challenge by providing a powerful, repeatable, and user-friendly solution to bridge the semantic gap, turning raw data into actionable knowledge.