Real-time Vandalism Detection in Wikipedia Streams using Graph Mining and Large Language Models

Biamonte Salvatore
Mat. 264177

Spadafora Pierpaolo
Mat. 543210

***Abstract*—Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.**

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

I. Introduction

Wikipedia stands out as one of the most significant sources of crowdsourced knowledge in the digital age. Its open nature allows for the democratization of information, enabling any user to contribute to historical and scientific documentation. However, this accessibility has an obvious downside: the platform is inherently vulnerable to malicious actors. Vandalism, misinformation, and the injection of unverified facts can occur at any moment, potentially compromising the integrity of the encyclopedia.

A useful tool for monitoring such behaviour is the real-time "recentchanges" API, provided by WikiMedia, which broadcasts every modification made across the platform. While this tool offers transparency, the sheer volume and velocity of the incoming data make manual verification impossible. Human moderators cannot keep up with the flood of edits that occur every second. Consequently, there is a critical need for autonomous systems capable of filtering this stream and flagging suspicious activities in real-time.

This project addresses this challenge by leveraging graph mining techniques to enrich the context of incoming edits. We construct a simple knowledge graph using Neo4j—based on Wikipedia page dumps and link structures—to capture the relationships between entities. Building upon this structural foundation, this work conducts a comparative analysis between two distinct detection paradigms: traditional Machine Learning classifiers (such as Random Forest and Gradient Boosting) and modern Large Language Models (Gemini). The primary objective is to evaluate the trade-offs between these approaches, determining which methodology offers the most effective solution for real-time vandalism detection.

In the upcoming sections, we will detail the graph construction methodology and the experimental setup. Finally, we will describe the implemented anomaly detection architectures and discuss the analysis of our findings.

II. Graph Construction Methodology

The construction of the Knowledge Graph is a multi-stage process that transforms raw Wikipedia dumps into a rich, queryable structure within Neo4j. This process involves data cleaning, topological sampling, community detection, and semantic enrichment.

A. Data Sourcing and Sampling

We utilize two primary datasets provided by the Wikimedia Foundation: the link graph dump (containing page-to-page references) and the article content dump (containing the full text). Given the prohibitive size of the complete Wikipedia graph for in-memory graph projections, we applied a dimensionality reduction strategy. First, the raw link data was parsed into a structured CSV format $(source, destination)$, filtering out redirects to retain only direct, semantic page references.

Subsequently, to generate a manageable yet representative dataset for the Graph Data Science (GDS) engine, we employed **Snowball Sampling** technique with a depth of $K = 2$, resulting in a coherent subgraph suitable for the experiments.

B. Topological Initialization

The sampled dataset is ingested into **Neo4j**. At this stage, the graph is purely structural: nodes represent pages (identified solely by their ID) and edges represent hyperlinks.

C. Community Detection

Once the topology is established, we enrich the nodes with structural cluster information using the **Neo4j Graph Data Science (GDS)** library. We integrated three distinct community detection algorithms to capture different granularities of network structure: *Label Propagation*, *Louvain*, and *Leiden*.

This structural partitioning serves a functional purpose in the real-time pipeline: it acts as a heuristic trigger. By monitoring the frequency of edits within specific communities, the system can detect abnormal bursts of localized activity (e.g., a coordinated attack on a specific topic). If the edit rate within a single community exceeds a predefined threshold, the system activates the anomaly detection modules, thereby optimizing resource usage.

D. Semantic Enrichment

The next phase involves populating the graph with semantic content. We process the *pages-articles* XML dump to associate text with the structural nodes. Since Wikipedia stores data in MediaWiki markup, we first perform a parsing and cleaning step to extract the plain text and the title of each article. This processed data is exported to a secondary CSV file containing tuples of $(page\_id, title, content)$. Finally, these attributes are mapped to the existing nodes in Neo4j, updating the graph schema to include the textual context required for the subsequent analysis.

E. Vector Indexing and RAG Preparation

To enable the semantic search capabilities required by the LLM, we extended the graph schema to support vector operations. A key design objective was to mitigate the inherent limitations of LLMs, specifically regarding knowledge cutoffs and hallucination. We envision a deployment scenario where the platform establishes data-sharing agreements with authoritative publishers (e.g., news agencies, scientific journals). These partners provide a stream of verified articles, serving as an external ground truth to validate Wikipedia edits against real-time facts.

To support this dual-source retrieval, we implemented a uniform **chunking strategy**. The textual content is segmented into smaller overlapping blocks using a sliding window approach (e.g., chunk size of 512 tokens with an overlap of 50 tokens). This overlap is critical to preserve semantic continuity across segment boundaries.

Each text segment is encoded into a high-dimensional vector using a pre-trained Transformer model (`paraphrase-multilingual-MiniLM-L12-v2`). In Neo4j, these vectors are materialized as distinct nodes to facilitate targeted retrieval:

- `:Chunk` nodes, linked to the original Wikipedia article via a `[:HAS_CHUNK]` relationship.
- `:TrustedChunk` nodes, linked to the external authoritative source via a `[:HAS_TRUSTED_CHUNK]` relationship.

Finally, specific Vector Indexes are created on both node types using cosine similarity. This dual-indexing architecture enables the Retrieval-Augmented Generation (RAG) pipeline to efficiently fetch both the potentially vandalized content and the authoritative ground truth context during the detection phase.

III. Experimental Setup

To ensure reproducibility and manage computational overhead, we selected the **Italian Wikipedia** snapshot as our reference dataset. The experiments were conducted using a containerized microservices architecture.

A. Dataset Generation

We utilized the official Wikimedia dumps for the Italian language (specifically `itwiki-latest-page.sql`, `itwiki-latest-pagelinks.sql`, and `itwiki-latest-pages-articles.xml`).

From this snapshot, we generated a synthetic labeled dataset following the pipeline described in the previous section. To maintain a strict class balance, the dataset is partitioned as follows:

- **Training Set:** 200 samples (100 Legitimate, 100 Vandalism).
- **Testing Set:** 40 samples (20 Legitimate, 20 Vandalism).

It is important to note that the dataset size is intentionally constrained due to the high computational (and monetary) cost of the LLM generation, hence this work is explicitly framed as a Proof of Concept. The primary objective is to validate the architectural feasibility of the Graph-RAG pipeline and the integration of microservices, rather than to achieve large-scale statistical generalization.

B. System Infrastructure

The system is deployed via **Docker Compose** to ensure isolation. To handle the real-time throughput, we introduced **Apache Kafka** as a buffering layer: incoming edits are published to a specific topic, allowing the ingestion layer to operate at high velocity while the detection modules process events asynchronously. The graph backend relies on **Neo4j Community Edition** with the *Graph Data Science (GDS)* plugin enabled for topological analysis.

C. Testing Methodology

Validating against live streams is challenging due to the sporadic nature of attacks. We therefore developed two injection tools:

1) **Automated Stream Injection:** A Kafka Producer that replays the synthetic test set to measure latency and stress-test the community triggers.
2) **Manual Adversarial Testing:** A CLI tool allowing operators to inject custom, human-crafted attacks in real-time, testing the system on more subtle modification.