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

Biamonte Salvatore
Mat. 264177

Spadafora Pierpaolo
Mat. 263722

***Abstract*—The rapid growth of user-generated content on Wikipedia poses significant challenges for maintaining information integrity. The high velocity of edits makes manual moderation unfeasible, necessitating automated real-time solutions. This report presents a microservices-based architecture designed to detect vandalism in Wikipedia streams using a hybrid approach of Graph Mining and Artificial Intelligence. We leverage Graph Data Science techniques (specifically the Leiden algorithm) to cluster pages into semantic communities, utilizing edit bursts within these clusters as heuristic triggers to optimize resource allocation. Furthermore, we implement a Retrieval-Augmented Generation (RAG) pipeline backed by a Neo4j Document Graph to provide factual context to the classifiers. We conduct a comparative analysis between a Large Language Model (GPT-OSS-20B) acting as a reasoning oracle, and lightweight Neural Network classifiers trained on synthetic adversarial data generated by a distinct model (Gemma-3-27B-it). Our results demonstrate that while Neural Classifiers achieve superior throughput and near-perfect accuracy on synthetic patterns, they lack the semantic reasoning required for subtle, context-dependent vandalism, which remains the domain of LLMs. This study highlights the trade-off between computational cost and semantic depth, proposing a tiered detection strategy.**

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 optimize the detection workflow. We construct a Semantic-Enriched Document Graph utilizing Neo4j to capture the topological structure of Wikipedia. By applying community detection algorithms (such as Leiden), we segment the graph into topical clusters. Detecting a sudden spike of edits within a specific cluster allows the system to identify potential coordinated attacks and prioritize computational resources accordingly.

Building upon this structural foundation, this work conducts a comparative analysis between two distinct detection paradigms: lightweight **Feed-Forward Neural Networks (FNN)** and thinking **Large Language Models (LLMs)**. Specifically, we perform an **Ablation Study** on the Neural classifiers to evaluate the impact of Retrieval-Augmented Generation (RAG) features, extracted from the graph, on the model's accuracy.

While generating large-scale synthetic datasets enables the training of fast neural classifiers, capturing the nuance of subtle, context-aware vandalism remains a challenge. Therefore, our analysis distinguishes between the statistical pattern recognition of Neural Networks and the semantic understanding of LLMs. 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 divergence in performance between synthetic benchmarks and manual adversarial testing.

II. Graph Construction Methodology

The construction of the graph is a multi-stage process that transforms a raw Wikipedia dump 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 primarily use two datasets provided by the Wikimedia Foundation:

- link graph dump (containing page-to-page references)
- article content dump (containing the full text of the page).

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 characters with an overlap of 50 characters). 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 embedding 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. Vandalism Detection Architectures

We implemented two opposing approaches to classification: a resource-intensive "AI Judge" based on Generative AI, and a suite of lightweight Neural Classifiers designed for high-throughput scoring.

A. The LLM Oracle (AI Judge)

The first approach utilizes **GPT-OSS-20B** as a zero-shot classifier. A crucial methodological choice was to employ a distinct architecture for the Judge compared to the synthetic data Generator (which utilizes *Gemma-3-27B-it*). This separation is designed to mitigate **self-preference bias**, preventing the Judge from overfitting on stylistic patterns produced by its own underlying model architecture.

The system constructs a prompt containing the user's edit comment, the original text, the modified text, and the "Ground Truth" retrieved via RAG from our Trusted Sources. The LLM is tasked with reasoning whether the edit is legitimate or vandalism based purely on factual consistency. While highly accurate, this approach introduces significant latency and eventual API/Local Inference costs.

B. Neural Classifiers and Ablation Study

To achieve real-time performance, we trained a set of **Feed-Forward Neural Networks** using PyTorch. The architecture consists of a multi-layer perceptron (MLP) with three hidden layers (256, 128, 64 units), employing Batch Normalization, Dropout (0.5), ReLU activation functions and a Sigmoid output for binary classification.

We conducted an **Ablation Study** to determine the importance of context (RAG). We trained five distinct variations of the model:

1) **Neural Complete:** Uses all features: embeddings of the old text, new text, user comment, and, crucially, the cosine similarity scores (RAG scores) retrieved from the Neo4j Vector Index (triangulation between Wikipedia and Trusted Sources).
2) **No RAG:** Excludes the similarity scores from the Graph, relying only on the text embeddings.
3) **No Comment:** Ignores the user's edit summary.
4) **Only New:** Analyzes only the embedding of the resulting text.
5) **Minimal (Baseline):** Uses only metadata features (text length ratio), serving as a naive baseline.

IV. 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:

1) `itwiki-latest-page.sql`,
2) `itwiki-latest-pagelinks.sql`,
3) `itwiki-latest-pages-articles.xml`).

From this snapshot, we generated a synthetic labeled dataset. We utilized **Gemma-3-27B-it** as the generative engine to craft realistic vandalistic and legitimate edits. Using a high-parameter model for generation ensures that the synthetic examples are semantically complex and challenging for the detection classifiers. 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.

V. Results and Discussion

A. Quantitative Analysis on Synthetic Data

The initial quantitative evaluation was performed on the synthetic dataset. As shown in Table I, the **AI Judge** achieved best accuracy among all the tested models, but with the obvious downside of evaluation time, making it unfeasible to use in a realistic scenario.

Table I
MODEL PERFORMANCE ON SYNTHETIC STREAM

Model	Accuracy	Avg. Time (s)
GPT-OSS-20B (AI Judge)	**84.27%**	6.357
Neural Minimal (baseline)	60.38%	0.0005
Neural No RAG	58.49%	0.263
Neural Complete (RAG)	56.60%	0.276

Note: The discrepancy in Neural performance compared to initial training suggests an overfitting on the specific synthetic generation parameters, whereas the LLM maintained consistent semantic reasoning (84.27%).

This convergence between the RAG and No-RAG neural models highlights a limitation in synthetic data generation. The generative model likely introduced latent stylistic patterns (e.g., distinct vocabulary or sentence structures in vandalized edits) that the neural classifiers learned to detect efficiently even without factual context. While this confirms the classifiers' robustness in detecting "AI-generated vandalism," it masks the value of the Graph for detecting factual inconsistencies in human-written attacks.

B. Qualitative Adversarial Stress Testing

To evaluate the system's robustness against subtle, context-aware attacks that synthetic data failed to represent, we conducted a manual adversarial testing phase using the injection tools described in Section IV-C. We crafted "semantic vandalism" scenarios: edits that are grammatically perfect and plausible but factually false.

Case Study: Polysemous Vector Ambiguity

- **Context:** Article section regarding *"Guerre romano-persiane"* (Roman-Persian Wars).
- **Edit:** Substituted the word *"persiane"* with *"saracinesche"* (resulting in *"Guerre romano-saracinesche"*).
- **Linguistic Note:** In Italian, *"persiana"* is polysemous: it denotes both the "Persian" people (demonym) and a "window shutter". The vandal replaced it with *"saracinesca"* (rolling shutter), a synonym strictly for the architectural object.

This adversarial edit exposed a critical limitation of embedding-based classifiers:

1) **Neural Classifier (No RAG):** Incorrectly classified as **Legitimate**. The embedding model placed *"persiana"* and *"saracinesca"* in close proximity within the vector space due to their strong synonymy in the architectural domain. Lacking context-aware disambiguation, the classifier perceived the substitution as semantically consistent.
2) **LLM Judge (with RAG):** Correctly classified as **Vandalism**. Leveraging the retrieved historical context, the LLM successfully disambiguated the term. It recognized that a "rolling shutter" implies an anachronistic object in the context of Roman warfare, flagging the edit despite the vector similarity.

C. Conclusion on Trade-offs

Our findings propose a tiered architecture as the optimal solution for real-world deployment. **Neural Classifiers** offer superior throughput, processing events in milliseconds, making them suitable for handling the high velocity of the stream. However, their performance on semantic vandalism dropped significantly in adversarial settings (hovering around 60% accuracy in our tests), proving they are insufficient as a standalone solution for context-dependent attacks. Consequently, the **Graph-RAG LLM Pipeline** remains indispensable. Despite the higher computational cost, it provides the necessary semantic grounding to verify facts and disambiguate complex edits that evade vector-based detection.