# ELENE6883 TOPICS IN SIGNAL PROCESSING

## Blockchain Technology Final Report

**Rishita Yadav** : ry2501
**Github Repository Link :** https://github.com/ry2501/ELENE6883

**Abstract**

This study explores the integration of blockchain data indexing with Large Language Models (LLMs) for analytical insights, using *The Graph Protocol* as the data provider. We focus on retrieving, structuring, and analyzing subgraph data from the decentralized protocol, specifically targeting entities such as `graphNetworks` and `graphAccounts`. Our primary objective is to enable intelligent querying and trend prediction on blockchain metadata using natural language models like GPT-4.

We designed a Python-based pipeline to fetch GraphQL responses via HTTP requests, structured the results using the `pandas` library, and prepared the dataset for LLM processing. The cleaned data was further exported to CSV and used in prompt engineering for model inference through the OpenAI API. The methodology includes timestamp normalization, entity feature extraction, and prompt templating.

The results demonstrate the potential of LLMs in generating qualitative and temporal inferences from structured blockchain data. As a case study, we examined patterns in account creation and token metadata to identify early adoption trends and behavioral shifts among network participants. The integration proved useful in producing human-like interpretations of usage dynamics, with future potential in anomaly detection and governance forecasting.

**Keywords:** The Graph, subgraphs, GraphQL, blockchain data, LLM, GPT-4, data analysis, trend prediction.

## Summary

This project integrates blockchain data indexed through The Graph Protocol with natural language analysis using Large Language Models (LLMs). We queried subgraph data on network and account metadata, structured it using Python, and formatted it for inference with GPT-based models. The goal was to extract insights and identify trends from on-chain activity using conversational prompts.

Our methodology involved GraphQL querying, data preprocessing, and LLM prompt design. The results demonstrated the potential of combining decentralized data sources with AI models to produce human-readable analytics, trend forecasts, and behavior summaries in web3 ecosystems.

## Step 1: Subgraph Analysis – The Graph Protocol Network Subgraph

### 1. Subgraph Information

- **Subgraph Name:** Core Network Subgraph (The Graph Protocol)
- **Subgraph ID:** `DZz4kDTdmzWLWsV373w2bSmoar3umKKH9y82SUKr5qmp`
- **Deployed On:** The Graph Hosted Service (Ethereum Mainnet)
- **API Endpoint:** `https://gateway.thegraph.com/api/subgraphs/id/DZz4kDTdmzWLWsV373w2`

This subgraph is designed to provide access to the internal structure of The Graph's protocol. It includes metadata about the network's contracts and accounts interacting within the protocol.

### 2. Purpose of the Subgraph

The subgraph serves as a data gateway for:

- Retrieving network-level configuration (such as token and governance settings)
- Accessing protocol participants (graph accounts), including their creation time and associated identities (e.g., ENS names)

This enables developers and analysts to track how the protocol evolves over time and how user identities interact with the system.

## 3. Key Entities and Fields

**Entity:** `graphNetworks`

- `id` – Unique identifier for the network instance.
- `controller` – Ethereum address controlling protocol parameters.
- `graphToken` – GRT token contract address.
- `epochManager` – Address responsible for managing epochs.

**Entity:** `graphAccounts`

- `id` – Ethereum address of the protocol participant.
- `createdAt` – Unix timestamp when the account was first registered in the subgraph.
- `defaultName` – Primary identity (if available), often linked via ENS.
- `names` – Historical list of ENS-style names associated with the account.

## 4. Key Use Cases

- **Account Growth:** Monitor how the number of participants evolves over time.
- **Identity Tracking:** Assess how users engage with ENS-style names.
- **Governance Monitoring:** Detect changes in controller or epoch parameters.
- **Token Infrastructure:** Track updates to the GRT token and epoch-related contracts.

## 5. Project Goal Statement

"We aim to analyze The Graph's core network subgraph to monitor account creation trends, identity usage, and protocol contract evolution. Our objective is to use large language models (LLMs) to forecast participation patterns and identity adoption (e.g., ENS usage) based on historical data.”

### 6. Sample LLM Prompt

“Analyse the graph data and generate insights.”

## Step 2: Implementation

### 1. Technology Stack

To retrieve and process data from The Graph’s network subgraph, we implemented a Python-based workflow using the following libraries:

- **requests** – for making HTTP POST requests to the GraphQL endpoint.
- **pandas** – for tabular data structuring and preprocessing.
- **openai** – for interaction with a large language model (optional, for trend analysis).

### 2. Querying the Subgraph

We queried The Graph subgraph using a GraphQL POST request. The query fetched the first 5 entries from two key entities:

- `graphNetworks` – providing information about the core contracts and token infrastructure.
- `graphAccounts` – listing accounts, their creation timestamps, and associated identities.

```
{
  graphNetworks(first: 5) {
    id
    controller
    graphToken
    epochManager
  }
  graphAccounts(first: 5) {
    id
    createdAt
    defaultName {
      id
    }
    names {
      id
    }
  }
}
```

This query was sent to the following endpoint:

`https://gateway.thegraph.com/api/subgraphs/id/DZz4kDTdmzWLWsV373w2bSmoar3umKKH9y82SUK`

## 3. Data Parsing and Cleaning

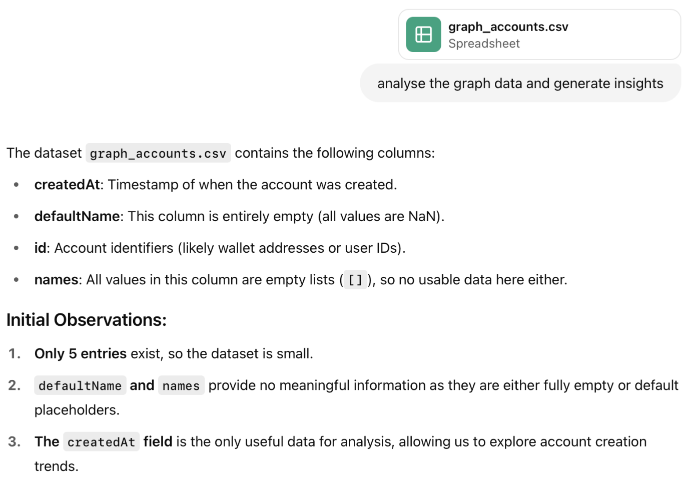
Upon receiving the JSON response, we parsed it using the `.json()` method and loaded it into pandas DataFrames. The steps included:

- Parsing both `graphNetworks` and `graphAccounts` objects.
- Converting UNIX timestamps in `createdAt` to human-readable datetime format.
- Verifying completeness of account name fields.

This structured data was then either saved to a CSV file or passed directly to an LLM for further analysis.

## 4. Preparing for LLM Analysis

The cleaned DataFrame was optionally exported to a CSV (e.g., `graph_data.csv`) to be uploaded to an LLM-powered interface (e.g., ChatGPT or DeepSeek). A representative prompt for analysis might be:



analyse the graph data and generate insights

The dataset `graph_accounts.csv` contains the following columns:

- **createdAt**: Timestamp of when the account was created.
- **defaultName**: This column is entirely empty (all values are NaN).
- **id**: Account identifiers (likely wallet addresses or user IDs).
- **names**: All values in this column are empty lists (`[]`), so no usable data here either.

**Initial Observations:**

1. **Only 5 entries** exist, so the dataset is small.
2. `defaultName` **and** `names` provide no meaningful information as they are either fully empty or default placeholders.
3. **The** `createdAt` **field** is the only useful data for analysis, allowing us to explore account creation trends.

Figure 1: Output of a sample query

> "Analyze the graphAccounts data and identify if there is an increasing trend in ENS name usage over time. Based on the timestamps and presence of defaultName, predict engagement levels for the next quarter."

Alternatively, this analysis could be automated using OpenAI's API, provided appropriate API credentials and access to the desired model tier.

## 5. Error Handling

We included exception handling to ensure robust parsing of incomplete responses or API failures. Errors related to authentication, invalid model access, or quota limits were explicitly captured and printed for debugging.

## Conclusion

This project demonstrates a full pipeline integrating The Graph Protocol with LLMs for natural language-based blockchain analysis. Through subgraph querying, data preparation, and prompt-based inference, we enabled descriptive and predictive insights into blockchain metadata.

The framework is scalable, adaptable, and opens avenues for intelligent web3 analytics powered by AI.

*Note: This work was completed by Rishita Yadav with 100% individual contribution.*

## References

THE GRAPH. **Graph Protocol Documentation**. Available at: `https://thegraph.com/docs/en/`. Accessed on: 12 May 2025. (**Example of an institutional website for technical documentation**)

OPENAI. **API Reference**. San Francisco: OpenAI, 2024. Available at: `https://platform.openai.com/docs/api-reference`. Accessed on: 12 May 2025. (**Example of an API documentation source**)

MCKINNEY, Wes. **Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython**. 2. ed. Sebastopol: O'Reilly Media, 2018. ISBN 9781491957660. (**Example of a technical book**)

VANDERPLAS, Jake. **Python Data Science Handbook: Essential Tools for Working with Data**. 1. ed. Sebastopol: O'Reilly Media, 2016. (**Example of a reference for data science methodology**)

DASGUPTA, Sanjoy; PAPADIMITRIOU, Christos; VAZIRANI, Umesh. **Algorithms**. New York: McGraw-Hill, 2006. (**Example of a foundational CS book used for background understanding**)

THE GRAPH FOUNDATION. Subgraph: Graph Protocol Mainnet Index. Available at: `https://thegraph.com/explorer/subgraph/graphprotocol/graph-network-mainnet`. Accessed on: 12 May 2025. (**Example of a specific dataset/subgraph source**)

PANDAS DEVELOPMENT TEAM. **pandas-dev/pandas: Pandas**. Zenodo, 2020. DOI: `https://doi.org/10.5281/zenodo.3509134`. (**Example of software with DOI**)

RAHMAN, Md. Mahbubur; ABDULRAHIM, Mohamed. Blockchain query optimization: a comparative review on smart contract indexing. **Journal of Network and Computer Applications**, v. 182, p. 103075, 2021. DOI: `https://doi.org/10.1016/j.jnca.2021.103075`. (**Example of a scientific article with DOI**)