# Data Science Challenge

## Authors

**Name:** Margonis Phevos, Trantalidis giannis
**ID:** f3352317, f3352314
**Date:** June 9, 2024

## Table of Contents

# Introduction

## Project Description

The goal of this project is to classify Greek domain names into predefined categories . This task, known as node classification, involves predicting the category of each domain based on its attributes and relationships within a web graph and its textual content.

In the provided dataset, we have a web graph consisting of 65.208 Greek domain names. Each node in this graph represents a domain, and the edges represent the hyperlinks between these domains. Additionally, we have access to the textual content of the webpages for a subset of these domain names. The challenge is to build a model that can learn from these structures and predict the class labels of unlabeled domains.

## Tasks

- Data Cleaning
- Feature Extraction: Extract meaningful features from the textual content and the graph structure.
- Model Training: Train various machine learning models that can learn from these features and accurately predict domain categories.

- Evaluation: To evaluate the performance of the models and analyze their effectiveness in classifying the domain names using the negative log-likelihood.

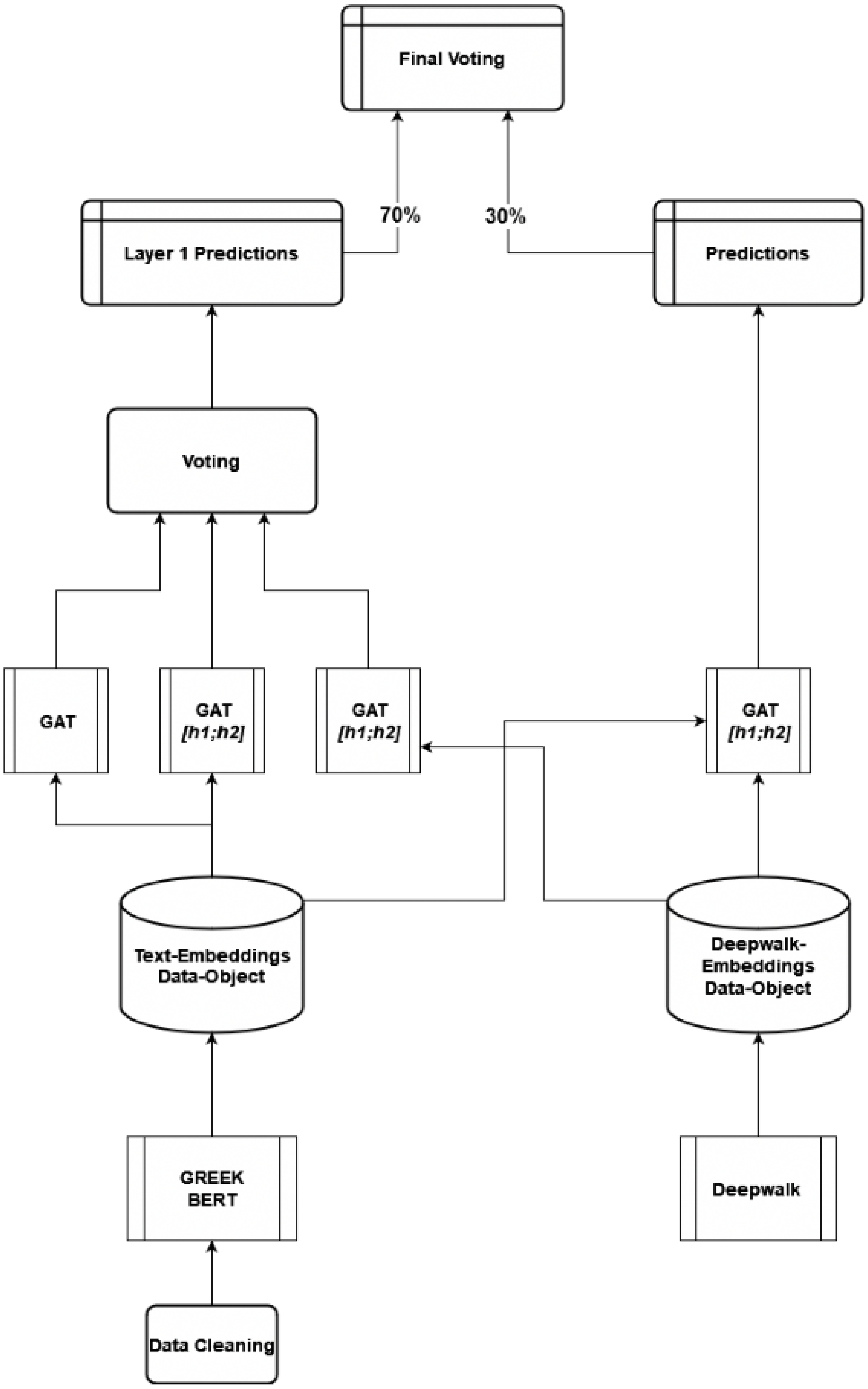

**Figure 1:** Final Model Architecture

# Methodology

## Exploratory Analysis

One of the most valuable representations for nodes can be derived from the texts associated with them. Text lengths vary significantly, ranging from zero to over twenty million characters, with a median length of 2,116 characters. However, not every node is associated with text; out of 65,208 nodes, only 40,600 possess text, leaving the remaining identifiable solely by their URLs and positions within the knowledge graph.

The dataset is divided into a labeled training set and an unlabeled test set. The training set includes 1,812 nodes, 306 of which lack text, while the test set contains 605 nodes with 98 nodes textless. The distribution of labels is consistent across both the training and test datasets.

Another crucial attribute for each node is its `in_degree`, which represents the number of incoming links from other nodes in this directed graph. The `in_degree` values span from 0 to 8,000, with a median of 6. High `in_degree` values pose certain classification challenges, as detailed in the methodology section, while a low or zero `in_degree` can negatively impact classification, particularly if the node also lacks textual content. Notably, both the training and test sets contain few nodes with nearly zero `in_degree`.

## Data Cleaning

Regarding the texts associated with each node, they can be distinguished by their `domain name` and several scraped web pages. Each page provides a complete URL path and its extracted content (`texts`). Initially, texts from each domain were cleaned by tokenizing and retaining only alphanumeric tokens. These tokens, compiled from all pages, formed the comprehensive `text` body for the domain. Additionally, URL paths and domain names were processed using regex to retain only legible words, culminating in a unique set of "url tokens" that typify each domain.

A significant challenge with URLs is their frequent use of concatenated words without clear delimiters. To address this, a unigram tokenizer using the `SentencePiece` library was trained on a custom corpus, which included the ten thousand most common English words alongside prevalent words from our URLs. Although a BPE tokenizer was also developed, comparative analysis suggested the unigram provided subjectively better results.

Another challenge was the prevalence of words in 'greeklish'. Referencing insights from the paper "Still all Greeklish to me: Greeklish to Greek Transliteration," we employed a Large Language Model (LLM) for effective transliteration. Utilizing a few-shot approach with examples from the demo of the classic transliteration tool outlined in "All Greek to me!" we engaged the OpenAI API, specifically `gpt-3.5-turbo-0125`, for transliterating `Greeklish` to Greek. A batch size of 50 was used to mitigate potential issues of AI amnesia and drift, necessitating the inclusion of node IDs in prompts to preserve context. Approximately 200 texts were inadvertently forgotten by the model, indicating challenges like AI drift. We set the temperature parameter to 0.5 to reduce the risk of generating non-factual content.

Lastly, the domain `texts` needed truncation to 512 tokens to accommodate a BERT model's limitations. Since each `text` aggregated content from multiple subpages, a simple truncation to the first 512 tokens was deemed inadequate. Therefore, a script was devised to generate "bootstrap" text samples, evenly extracting segments from each subpage. This sampling technique not only diversified the captured information but also helped avoid text repetition from dominating the extracted content.

## Feature Extraction

### Bert Finetune

To construct text representations, we utilized the Greek version of BERT provided by AUEB. We divided the available training set into two parts: a smaller training set for model fine-tuning and a validation set for performance monitoring. The model was fine-tuned over three epochs on the training subset, with the validation set employed to monitor the process and prevent overfitting.

Given that not all nodes were associated with text, we pre-appended a list of cleaned URL-derived words to each text. This strategy was designed to achieve several objectives: firstly, to generate representations for nodes lacking textual data; secondly, to transfer the insights gained from fine-tuning on text to the URLs; and thirdly, to avoid overfitting on URL data by not training simultaneously on both text and URLs.

Ultimately, we fed the concatenated URLs and texts into the fine-tuned BERT model. From each domain's output, we extracted the CLS token, which serves as the representation for that domain. This approach helped integrate the contextual relevance of both text and URLs into a unified model framework.

### Deep Walk

DeepWalk is a graph embedding technique that leverages random walks to learn representations of nodes in a graph.The essence of DeepWalk is to represent nodes in a continuous vector space, where similar nodes in terms of their graph structure are positioned closer together.

In our approach, we construct sequences of walks by initiating 50 random walks from each node in the graph. Each walk spans a length of 70 nodes. During these walks, we extend the sequence by up to 4 neighbors (if the node has 4 neighbors else its number of neighbors) when encountering a node.

Then, we leverage **Word2Vec**, a popular technique from natural language processing, to transform these sequences into meaningful embeddings. By passing these walks to Word2Vec, we learn 130-dimensional vector representations for each node in the graph. This process enables us to capture the intrinsic relationships and structural patterns present in the graph. We later use these vectors for the node classification task.

## Models

In light of the datasets we encounter—comprising primarily unlabeled data alongside a sparse set of labeled instances—it becomes clear that any effective classification algorithm must utilize both. Graph Neural Networks (GNNs) are particularly adept at this task. They excel by aggregating and transforming representations of nodes based on their local neighborhoods, utilizing existing labels to enhance this process. Crucially, Graph Attention layers refine this mechanism by assigning different weights to neighboring nodes, thereby enhancing the clarity and relevance of the aggregated information. To optimize performance, all models undergo tuning of hyperparameters, such as learning rate, weight decay, dropout rate, and the number of attention heads, using tools like Optuna. This approach ensures that each parameter is adjusted to maximize the model's effectiveness.

1. **CLS1_GAT**: We pass the CLS embeddings produced from BERT to a GAT neural network. This network consists of 2 GAT convulutional layers with 13 hidden channels. The first layer has 4 attention heads and the second 1. The second layer outputs the probabilities for the 9 classes for each node.

2. **CLS2_GAT_concat**: Again, we pass the CLS text embeddings produced from BERT to a GAT with different structure. This GAT has 2 layers. Both layers have 25 hidden channels and 2 attention heads. The output embeddings from these 2 GAT layers are **concatenated** and passed through an MLP to predict the 9 class probabilities for each node.
3. **WALK_GAT_concat**: We use a GAT model with input the embeddings produced from the Deep Walk. This model has 2 layers. Both layers have 27 hidden channels and 4 attention heads. The output embeddings from these 2 GAT layers are **concatenated** and passed through an MLP to predict the 9 class probabilities for each node.
4. **CLS + WALK_GAT_concat**: We concatanate the CLS texts embeddings with the Deep Walk embeddings into a single vector. We pass these concatanated embeddings to a GAT model with 2 layers. Both layers have 16 hidden channels and 2 attention heads. The output embeddings from these 2 GAT layers are **concatenated** and passed through an MLP to predict the 9 class probabilities for each node.

# Results

We employed a hierarchical voting scheme to optimize the balance between underfitting and overfitting in our ensemble model. Initially, we integrated predictions from models 1, 2, and 3 using a soft-voting mechanism, treating each model's output equally. This first layer of predictions aimed to leverage diverse model insights.

Subsequently, we introduced a second layer of voting. Here, we assigned a 70 percent weight to the aggregated predictions from the first layer and a 30 percent weight to predictions from a fourth, distinct model. This weighted approach helped to temper the influence of any single model, particularly if it had a propensity to deviate from the others, thereby enhancing the overall stability and accuracy of the ensemble.

This methodological refinement culminated in a log loss of 0.6691 on the public set. The effectiveness and robustness of this two-tiered voting system led us to select it as our primary strategy for the final model deployment.

| Model | Validation Loss | Public Loss | Private Loss |
|---|---|---|---|
| CLS 1 GAT | 0.7931 | 0.8343 | 0.8439 |
| CLS 2 GAT_concat | 0.7703 | - | - |
| WALK GAT_concat | 0.8067 | - | - |
| CLS+WALK GAT_concat | 0.75 | 0.7848 | 0.7631 |
| First layer voting | 0.69 | 0.6693 | 0.7255 |
| Final Voting-submission | - | 0.6691 | **0.7059** |

# Failed Attempts

This section documents various strategies we explored that did not yield optimal results, organized by increasing complexity.

1. **Graph Convolutional Networks (GCN):** We experimented with different configurations of GCN layers. Their ineffectiveness is likely due to the high variability in the number of available neighbors per node, which might have undermined the layer's ability to generalize.

2. **Neighbor Sampling with GraphSAGE:** To address the challenge of nodes within large neighborhood clusters, we utilized the `NeighborLoader` methodology, which samples a fixed number of neighbors for each node, in combination with a GraphSAGE model. However, this approach did not significantly improve performance.

3. **Text and URL Vectorization:** We tested various text vectorizers, such as Count and TF-IDF, applied to both texts and URLs, paired with a range of classification models from XGBoost to GNNs. None of these combinations met our performance benchmarks.

4. **Multilingual BERT for Embedding Extraction:** We deployed Multilingual-BERT to separately extract embeddings for URLs and texts, aiming to capture relevant attributes from the English segments of the URLs (e.g., auto/moto/sport). This method did not achieve the expected enhancement in model performance.

5. **Node2Vec for Neighborhood-Focused Embeddings:** We generated Node2Vec embeddings with the parameters ( $p < q$ ) to promote random walks primarily within neighborhoods, aiming to capture local structural information more effectively. Following the initial setup, we engaged in extensive hyperparameter optimization using Optuna to refine our embedding strategies. Node2Vec includes a straightforward `test` method, which facilitates evaluation through a downstream logistic regression classification. The most promising embeddings derived from this process were integrated into our top-performing models to assess their impact on predictive accuracy.

6. **Graph Autoencoders (GAE/VGAE):** Variational and standard graph autoencoders were used to derive latent (z) representations from combinations of text, URL, DeepWalk, or Node2Vec data. These were incorporated into our leading models, including GNNs and XGBoost, but failed to provide a competitive edge.

7. **Link Prediction with GAE/VGAE:** We trained GAE/VGAE models on various data representations to predict links, particularly aiming at enhancing connectivity for nodes with fewer than two incoming edges. Despite experimenting with adding 3 to 100 new edges per node and employing weighted edges to counteract quality degradation, this approach indicated that newly introduced edges tended to introduce additional noise rather than beneficial information, ultimately increasing classification loss.

These endeavors, while not leading to direct improvements, provided valuable insights into the limitations and potential areas of refinement for our methodologies.