University Of St Andrews

CS4099 Senior Honours Project

Dynamic of Wikipedia Subnetworks in the field of Computer Science

Author:
Xiangli Li

Supervisor:
Christopher Stone

May 2022

Abstract

Encyclopedias are collections of information that covers a wide range of topics. They are traditionally carefully curated into a glossary ordered alphabetically for ease of access. Nowadays, encyclopedias can be found online, with Wikipedia being the largest encyclopedia to ever exist, and it looks like it will continue to be for a long time in this digital age.

One important trait of Wikipedia's is that its articles have intrinsic structures to them. Articles contain hyperlinks to related articles which gives rise to an underlying hyperlink graph structure, and articles are also assigned to categories which gives rise to an underlying categorisation tree.

The underlying structure of Wikipedia gives us an entry point into understanding how certain subjects are perceived in the minds of its readers. Alongside Wikipedia's hyperlink and categorisation structures, the contents of Wikipedia articles also convey information on how correlated they are to other articles. A **graph structure** can be constructed from this, describing dynamics within the subnetworks of Wikipedia. The goal of this project is to inspect the relationship between categories in **Computer Science** in particular through examining Wikipedia articles in this field, the subcategories they belong to, and their place in the overall graph structure.

Declaration

Acknowledgements

I am grateful to my supervisor Christopher Stone for his continuous support throughout this academic year, especially in introducing me to ideas that I was not previously familiar with. His insight and feedback have been greatly valued.

Contents

1 Introduction

In Wikipedia's own words, it is a multilingual, free, online encyclopedia that open to anyone who desires to contribute their knowledge. It is the largest and most read reference work in history. As of April 2022, the English language Wikipedia alone contains over six and a half million articles[14].

Wikipedia's extent of coverage of topics is one of the reasons it is so successful. Traditional encyclopedias like Encyclopædia Britannica are written by experts in the corresponding areas, which means traditional encyclopedias are prided on their accuracy in content. In 2005, the science journal Nature concluded that there exist few differences in accuracy between Wikipedia and Britannica[11]. Britannica however disputed this. Wikipedia's commitment to maintaining reliable and accurate content has improved over the years. This along with its widespread usage means even the scientific community has to be aware of the information published on Wikipedia. An example of this is Wikipedia's partnership with the World Health Organization (WHO) in regards to misinformation during the COVID-19 pandemic[12]. With that being said, as a consequence of being open, Wikipedia state that it does not promise the validity of its content. Whatever the reader's opinions on Wikipedia are, it cannot be contended that Wikipedia is widely used by the general population, and appears often in popular culture, which marks its significance in today's world. Understanding Wikipedia can be seen as a way into understanding how topics are perceived, and this is the underlying principle of this project.

2 Project Objectives

Below is a list of original objectives of the project as documented in the Description, Objectives, Ethics & Resources (DOER) document from the first semester.

Primary:

- Expand on the work of Volodymyr Miz et al to come up with a way of generating a graph structure used for analysis. Each topic will be a cluster of node in the graph structure and the "progress" of a given topic can then be evaluated in context of the whole graph structure
- Each aspect (represented by a node) of a topic (represented by a cluster of nodes) should have a suitable number of other nodes connected to it so its relationships with them can be tracked, investigate what this optimal number of nodes should be
- Come up with metrics for evaluating the progress of a given topic. These need to be measurable. Metrics could include size of node, position of node in the graph, direction, new nodes
- Create a webpage that displays this information. The webpage should allow the user to explore different topics as they wish

Secondary:

- Guesstimate how the users navigate from one topic to the next (or which topics are viewed in the same sitting/session) by comparing user activity and time-series of the topics
- Display topics that are viewed in conjunction with each other in an interesting manner on the web page

- Further develop the webpage to allow the user to view a time-lapse of a topic of their choosing

Tertiary:

- Investigate how the public's perception of AI related topics have changed over time by tracking the number of positive/negative words in the edit history

3 Software Engineering Process

This project can be seen as a series of mini-projects: choosing an appropriate way to retrieve/model the graph structure, gathering appropriate data from Wikipedia, preparing the data, process the data, then lastly visualise the data. Due to the vast amount of content that Wikipedia holds, and the volume of libraries out there that help with handling this data, I did not follow a stringent process right from the beginning. Instead, an incremental development process was followed. I thought it would be fitting to combine linear and iterative methodologies: treat each step as a mini-waterfall then combine them to ensure they work together.

The waterfall process includes design, implementation, testing, and integration stages. The integration stage requires special attention to what kind of communication is required between the different components. For example, not everything can be displayed on the frontend, so we need to decide on what is relevant and significant enough to warrant communication from the backend. Bearing each component's position in the entire project in mind, when one step is complete, move on to the next part.

3.1 Choice of Languages and Technologies

A crucial aspect in development is the choosing the most fitting languages and technologies. Some were chosen at the outset and some were experimented with and compared due to alterations in development. The tools that were decided on from the beginning are outlined in this section.

3.1.1 Python

Python is used often in data analysis/data science, and this is largely due to its simplistic syntax and its smooth learning curve. Its common usage gives rise to many libraries that extend its basic functionality, particularly in the area of data processing and data analysis. I decided on using Python for preprocessing data over other high-level languages like Java due to its widespread usage and extensive community support.
python

3.1.2 SQLite

Making use of a database makes managing and reviewing the information much simpler. The tables are searchable and sortable, and being able to execute SQL commands in the console is useful in making sure that the processed data is as we expect. SQLite is a self-contained SQL database, it comes bundled with Python so it seemed natural to go for SQLite.

However, SQLite is slow compared to alternatives because it makes zero effort at indexing. To counter this, indexes are manually created to allow for faster retrieval of records.
sqlite

3.1.3 React JS

React JS is a JavaScript library built by Facebook. As we want an interactive visualisation on the frontend, React JS seemed like a good choice due to its key benefits being its speed and flexibility. react

3.1.4 PlotlyJS

PlotlyJS is a charting library that provided many chart types. It is used in this project for plotting the final graph structure. Another visualisation library, D3, was also considered. However, D3 has a reputation of having a steep learning curve. The frontend requirements of the project are simple, we want an interactive scatter plot that will display relevant information when a point is clicked. I decided that an easy to use library would suffice for this.

The three main components to consider when using Plotly are data, layout, and figure. The data object defines what we want to be displayed (i.e. the article nodes); The layout object defines features that are not directly related to the data, like title, axis title etc; The figure object creates the final object to be plotted and contains the previous two elements.
plotly

4 Ethics

This project uses Wikipedia data, which is publicly available to download from Wikimedia data dumps. However, this data includes editor usernames that are not fully anonymised and is potentially revealing. An ethical application was submitted to the School of Computer Science Ethics Committee and this was approved.

5 Context Survey

In this chapter, we will review the literature relevant to this project including articles on collective memory, corpora usage, and natural language processing.

5.1 Hebbian Learning Theory

The basic idea of the Hebbian Learning Theory was introduced by Donald Hebb in 1949[5]. This theory attempts to describe the learning process as neurons being fired or activated with other neurons, forming a neural network. Connections between neurons are initially weak, but as they are activated together repeated, the connections between them grow stronger and our actions then in turn become more intuitive. According to Hebb, this is the underlying process to how humans learn.

To demonstrate this, think about how when we are just starting out in something, every action is deliberate. As we grow more familiar with the task at hand, we get to relax a bit more. For example, when we are learning a new language, repeatedly saying or writing a word helps us remember it; When we are learning to ride a bike, going out and getting on the bike helps us learn to not fall.

This is, to some extent, analogous to how Wikipedia operates. Each Wikipedia article can be seen as a neuron, and the connection between articles is the idea of *association* or *correlation* between the articles[9]. For example, when a user of Wikipedia is reading an article on artificial intelligence, they might think of Alan Turing who proposed the Turing test. If we were to visualise a Wikipedia graph structure being composed of article nodes, this association in the reader's mind between artificial intelligence and Alan Turing will strength the connection (e.g. edge) between the two article nodes.

However, the Wikipedia graph structure does not simply exist in *one* reader's mind. Millions of readers will access Wikipedia everyday, read various articles, and make their own unique associations. This is where the notion of *collective memory* comes into play.

5.2 Collective Memory

Memory is an interesting thing, for both individuals and groups. Our brains will forget things by design so we are not overwhelmed by useless memories. However, sometimes even we ourselves are surprised by what we remember - we have all been in the situation when a past memory pops into our head seemingly out of nowhere. The point is, memory is complex and cannot be explained in a simple manner.

The makes the idea of collective memory more fascinating. The notion of collective memory was first introduced by Halbwachs[4]. It can be thought of as a common image of the past of a community. Collective memory is ever changing, the now may affect the past and the future may affect the now. The existence of Wikipedia allows dynamics of collective memory to be observed, and new insights on a community's common image of events to be gathered. If we know what triggers memories of a past event, then we may be able to predict related events that may be remembered by the reader. Wikipedia contains many features that make analysis of collective memory possible. For example, the editing activities of Wikipedia may be a good indication of the collective memory construction process.

Wikipedia can be seen as a source of collective memory for global events, like natural disasters. The dynamics of Wikipedia as a global memory repository for high significance events can be investigated. If we think about events in our life, there are certain events that will cause us to relive other past events. For Wikipedia, a similar process can occur - we can investigate the catalysts for reviving memories, e.g. what are the triggers of revisiting behavior of event pages?

The content and usage of Wikipedia are both important sources of information when it comes to analysing the construction of collective memory. The content includes editing history which gives an indication of how readers' perception of an event has changed, and view logs patterns may be used for estimating the *quantity* of public remembering of events.

Individual memory can falter according to the forgetting curve, as proposed by Ebbinghaus[3]. Episodic memory which is responsible for memorising details of events may be quickly forgotten due to the realisation of newer events. However, there are factors that can boost human memory of an event. First of all, the more frequency and recent an activation of a memory is, the more likely it will be remembered. If we treat collective memory as the sum of individual remembrances, then we can conclude that the 2011 nuclear catastrophe in Fukushima reminded readers' of the Chernoly incident a few decades prior[7].

There are experiments that use Wikipedia pageview data to gain insights into how event memory is triggered as part of collective memory. This means asking the question, how likely is it to trigger the remembering of another event, given a current event? Events are compared against two metrics, temporal similarity and location similarity. The first is how close in they are, e.g. a week between the two events. The second is the locations of the events, e.g. London and Manchester. Collection memory are affected by both of these metrics. However, the correlation is not strong. In general, location and time do not make significant contributions to remembering scores (i.e. a measure of how well remembered an event is, given a triggering event) for events, but events with high scores have clear location and time similarities with the triggering event.

Kanhabua et al's main motivation was to identify catalysts for triggering remembrance of past events, based on Wikipedia pageview stats. Identifying patterns from past memory triggers is difficult due to the noise in stats, and the various event types, unique circumstances surrounding each event, and the multitude of possible reasons for a reader visiting a page. Despite these reasons, the researchers still found some interesting patterns and concluded that recent events in the same region are likely to be remembered (looking at earthquakes and aviation accidents). Additionally, semantic similarities (particularly in reference to terrorist attacks) also influence which events are remembered well.

A final note from the authors describe high impact events to be more likely to be remembered though they are not always strongly correlated in time or place with the currently considered event. A high impact event could be one where lots of casualties or financial loss occurred.

5.3 Natural Language Processing - Wikipedia as a Corpus

Natural language processing is the study of interactions between machines and natural (i.e. human) language. Natural language processing models can be constructed for the purpose of analysing natural languages. The corpus used to train a natural language processing model is important as it serves as the foundation and its quality will significantly affect the quality of the final model.

A corpus is defined as a large set of text which represents a language and is used to analyse it. In natural language processing, the corpus will provide evidence for usage of the vocabulary that lets the model learn how this language is utilised.

Corpora can be of many types[17]. First of all, it can be monolingual or multilingual. Another aspect to consider is the time frame at which it is collected, e.g. wartime articles will have a totally different undertone from articles published during peace time. Lastly, is the text from a specific domain, such as medicine or science? The Penn treebank for example, is a commonly used corpus, but it is geared towards financial English which may not be suitable for all applications.

The success of a natural language processing technique relies heavily on the corpora used and their quality. In recent years, Wikipedia has been gathering attention and interest in the field of NLP for this reason. Wikipedia's ever-growing nature, wide coverage, recency and high quality of content seem inviting to NLP researchers. Many have used Wikipedia as a structure semantic knowledge base.

The English Wikipedia alone contains almost four billion words, and is continuing to grow at a rapid pace[16]. The internet in general provides a large, free and easily accessible corpus, but Wikipedia is especially suited to natural language processing applications. Consider the difference between a tweet and a Wikipedia article. Tweets are informal, ungrammatical, and casual, while Wikipedia articles are well-formed, grammatical and often written for educational purposes. Additionally, Wikipedia is a good representation of the population's interest distribution in the world.

6 Design and Implementation

In this chapter, important design decisions are discussed. The components of this project are relatively independent and they are discussed in each subsection. What I envision as the outcome of this project is an interactive graph structure the user can play around with: when they click on a certain node (i.e. article), a side panel will appear with information about that particular node. To achieve this, both a frontend and a backend are needed. The backend will handle the the pre-processing of relevant information, while the frontend will enable interactivity. Backend tasks include extracting articles in the field of Computer Science and generating the coordinates for these articles. Frontend tasks on the other hand, will include displaying the relationships the articles have with each other and plotting the graph structure given coordinates from the backend.

See below for a mockup diagram of how I imagine the final frontend to look.

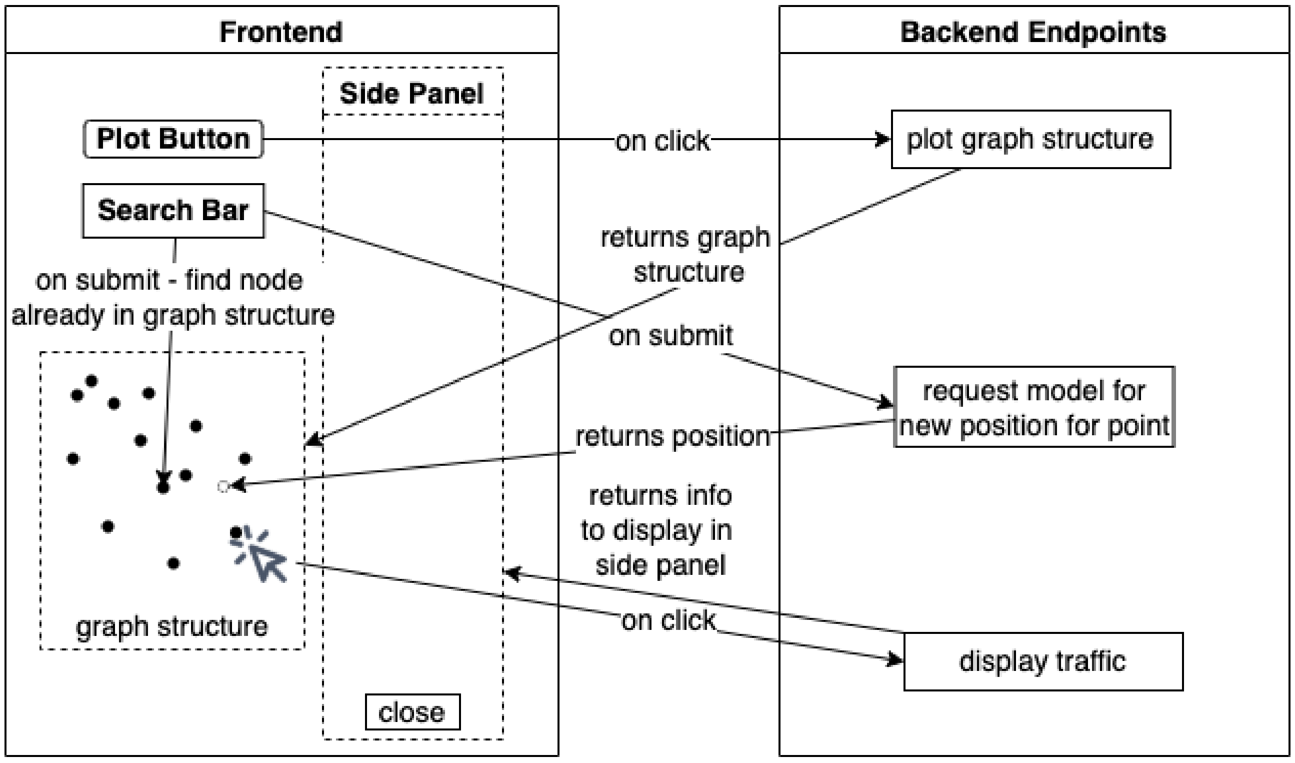


Figure 1: Mockup

See below for a diagram of the backend pipeline.

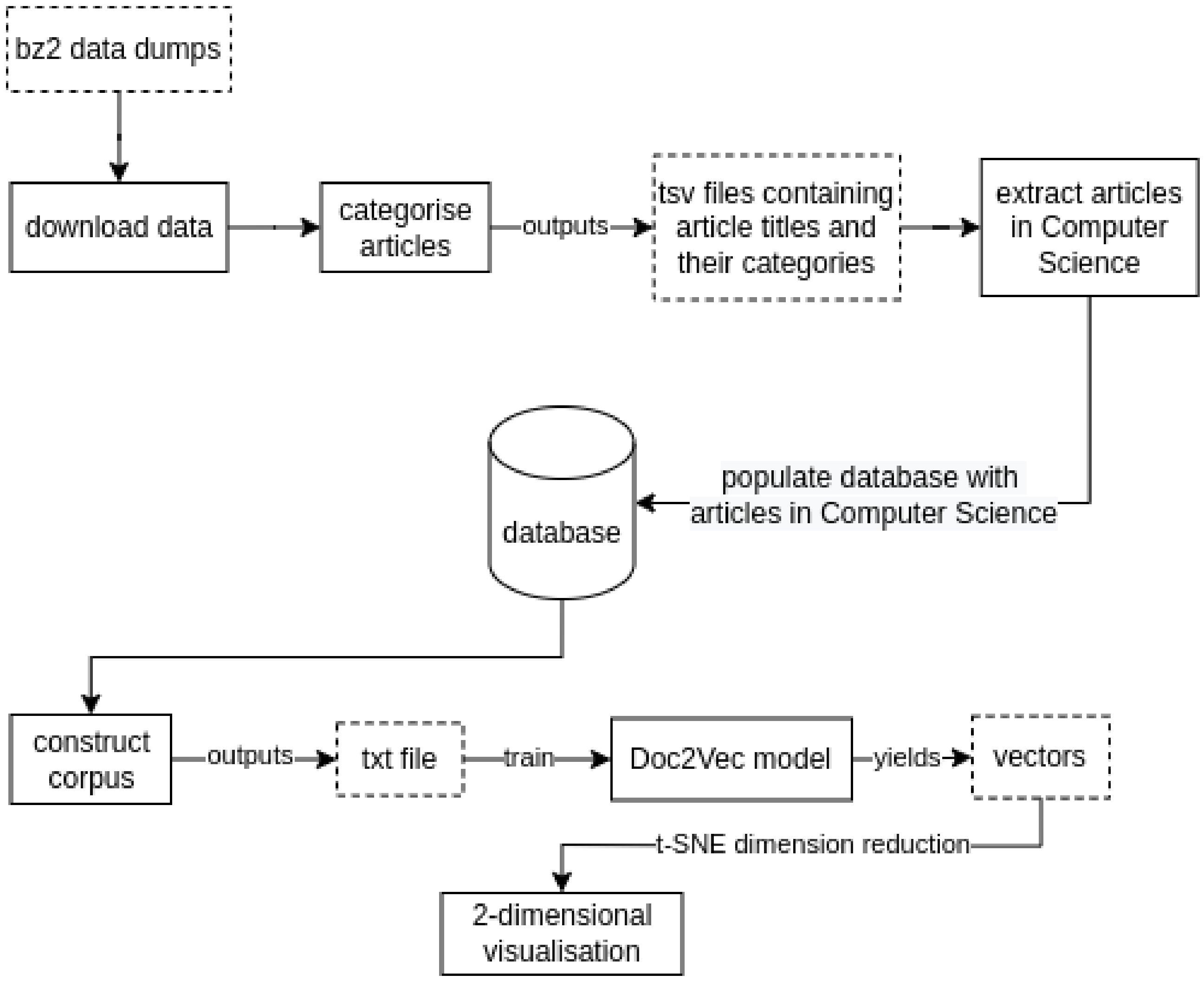

bz2 data dumps
download data
categorise articles
outputs
tsv files containing article titles and their categories
extract articles in Computer Science
populate database with articles in Computer Science
database
construct corpus
outputs
txt file
train
Doc2Vec model
yields
vectors
t-SNE dimension reduction
2-dimensional visualisation

Figure 2: Pipeline

6.1 Backend-Frontend Connection

Requests from the frontend to the backend are routed to URL endpoints. Communication from the backend to the frontend is completed through JSON responses. For visualising the graph structure, the frontend will request the database for nodes to plot and the backend will send back a list of coordinates (x and y). The frontend will then plot this graph using PlotlyJS. Each of the points are also assigned a colour, which represents the category the article node belongs to.

Yarn and Flask are used to aid the development process. Yarn is a project manager type application and Flask contains functionalities that make web development easier including routes from URL to functions. See below for a diagram of the functionality of the backend endpoints which the frontend may send HTTP GET requests to.

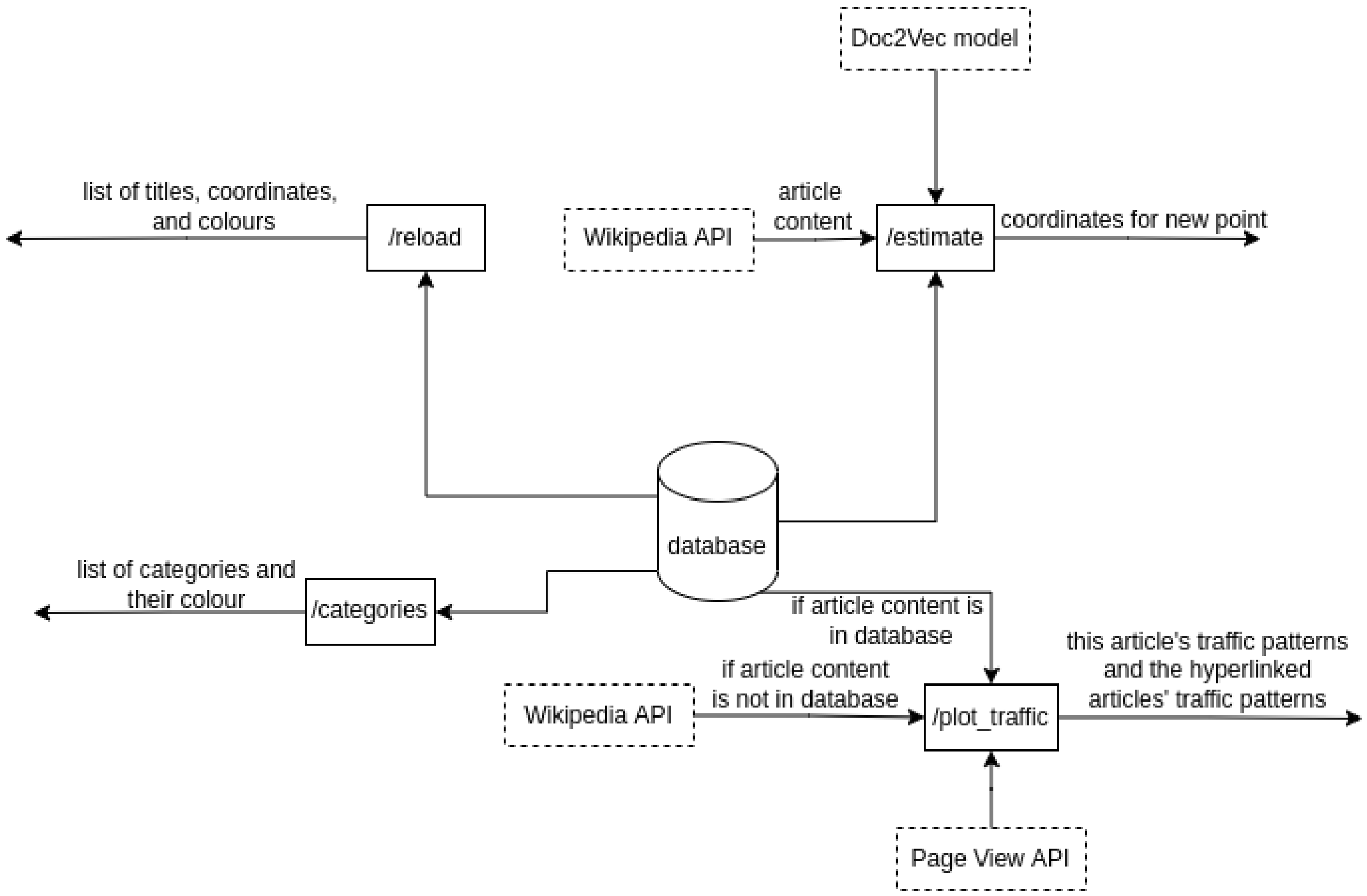

Doc2Vec model
list of titles, coordinates, and colours
/reload
Wikipedia API
article content
/estimate
coordinates for new point
database
list of categories and their colour
/categories
if article content is in database
this article's traffic patterns and the hyperlinked articles' traffic patterns
if article content is not in database
Wikipedia API
/plot_traffic
Page View API

Figure 3: Endpoints

6.2 Discussion of the Wikipedia Category Graph

I would like to first discuss the relationship between the content and tagged category of Wikipedia articles. As mentioned in previous sections, Wikipedia has its own underlying hyperlink structure, which involves articles being tagged with labels describing their categories. The purpose of categorisation is to make it easier for readers to browse related articles. This is captured by the idea of **folksnomy** - articles can have tags assigned to them by end users for future usage. In the case of Wikipedia, its articles are tagged by the editors into a category by including a special link to an article that describes the category. Each article can be tagged to multiple categories, and these categories may be further categorised forming a hierarchical category graph[15].

Therefore, Wikipedia can be seen as a knowledge graph that has a human authored categorisation system. However, this human touch introduces more problems than it solves. Wikipedia imposes no strict rules on the form of this hierarchy, permitting multiple categorisation schemes to co-exist thus also allowing loops. In practice, due to large amounts of data that Wikipedia holds, as well as its collaborative, granular, and ever-evolving nature, the category hierarchy is sparse and noisy. As an estimate, there are 1.5 million categories on 6 million Wikipedia articles[1]. It contains redundancies, inaccuracies, and mistakes.

From a semantic point of view, the category graph is inherently fuzzy because there are no objective methods to classification. To conclude, the sheer volume of categories mean they are not very informative. It is better to think of the category graph as an ever-changing thesaurus that contains overlays from different hierarchies, rather than a neat, directed, acyclic graph.

Furthermore, the relationship between the contents and categories of Wikipedia is somewhat paradoxical. We cannot both construct the category structure from article content and enrich information on articles using the category structure (which at this point does not exist). All these factors make Wikipedia's category structure difficult to use directly.

6.2.1 Solution - Categorisation using Centrality

Conti et al[1] presented a method for solving the paradoxical relationship between the contents and the categories of Wikipedia articles. This method will only use information contained in the category pseudo-forest. A pseudo-forest is an undirected graph where every connected node has at most one cycle. In the case of the Wikipedia category structure, using a pseudo-tree eliminates the possibility of a subcategory being a parent of its parent category.

Before looking into the algorithm, we should first discuss the **centrality** measure. In graph theory, centrality is the idea of assigning ranks to nodes within a graph according to their position in the structure. Centrality measures include geometric measures (i.e. closeness), path-based measures, and spectral measures (i.e. projection-based). Centrality measures are used to assess the position of the categories in the pseudo-forest. Categories of Wikipedia can be ranked according to their centrality in the category pseudo-forest forming a new hierarchy.

We can guarantee the resulting hierarchy is acyclic if, given two categories x and y, we conclude that x is a subcategory of y if it was marked as a subcategory in the original pseudo-forest (keep in mind that no loops are allowed) and the centrality score of x is less than that of y. The core principle of this process is that only categories that are fundamental/internal to the category hierarchy itself are selected as base categories (called milestones). This is the first step of Conti et al's method. Harmonic centrality is then used to generate the rankings of the categories.

Next, every other category is mapped to the closest reachable milestone. This means each category is mapped to its most specific (i.e. closest) generalisation (i.e. milestone) that has been preserved. Categories which have no milestones are orphans and therefore dropped.

Now that there is a new acyclic category hierarchy, each Wikipedia article is reassigned accordingly. If an article is tagged categories C1 and C2, it will now be mapped to milestone categories of C1 and C2. If the categories an article belongs to are all orphans, then the article itself will not be mapped to anything either.

6.2.2 Using the Categoriser

To utilise the categoriser, we follow Paolo Boldi and Corrado Monti's README file[2]. The three basic requirements are Java 8, Ant (a general purpose build tool), and Ivy (a package manager). A **Vagrant** box using Ubuntu trusty64 is set up, and the required packages are installed via terminal commands.

Some issues were encountered here. One issue was that Oracle changed its licensing program therefore the machine failed to fetch some Java packages. This was fixed through some research and the *apt-get install openjdk-8-jdk* command was used instead. Another issue was that Ivy dependencies are routed through HTTP, and they could not be resolved as HTTPS was required. This was fixed through adding a resolver that enforces HTTPS as the root.

With the virtual machine environment set up, we can start downloading Wikipedia data. Bz2 files are downloaded from **Wikimedia**. The datadump from **November 2021** is used for this project, and takes up around 17.8GB in total. I am using the segmented Wikipedia datadumps over the datadump which contains all of English Wikipedia in its entirety. This is because a smaller datadump does not take long to process while the full 17.8GB datadump takes upwards of 30 hours to process (Boldi's code displays the estimated completion time). If an error were to occur at a late stage it would result in a lot of wasted time. Additionally, the results of the categorisation process can be examined fairly quickly when performed on smaller files.

The program yields several output files but we are only interested in the file that contains the mapping between articles and their categories. The virtual machine has a directory named *vagrant*, whose contents are synced to the host machine's (i.e. my home PC's) directory. When the categorisation process is complete, the mapping files are moved to the *vagrant* directory for access from the host machine. These mapping files are also backed up on a flash drive.

6.3 Extracting Articles in Computer Science

Selecting the right articles is an important step. The selected articles will make up the nodes in the graph structure. The collection of chosen articles serve as the backbone of this project and care should be taken when determining whether an article is considered an article in Computer Science. We can use category mappings of articles from the last stage to filter and preserve only articles in Computer Science. The point is that if an article is mapped to any category that is considered a Computer Science category, then the article itself should be considered an article of Computer Science.

Every Computer Scientist will have their own biases when it comes to categorisation, and thus will have in mind a different set of notable categories from someone else. Furthermore, there are certain areas that have taken on a life of their own that may not be considered 'core' Computer Science by specialists. A good example of this is artificial intelligence. I personally would consider artificial intelligence a subset of Computer science, in the sense that the everything in artificial intelligence stems from ideas in Computer science.

Pmernyei et al[13] expanded on Conti's method, and like us, with a particular focus on Computer Science categories and articles. They inspected 10,000 categories selected by the categoriser and identified **ten** important areas in Computer Science. These include computer file systems, programming language topics, computing architecture. I though this was a good starting point.

I also came across a 'Map of Computer Science'[8] created by Dominic Walliman, an educational YouTuber who is keen on data visualisation and has produced various visualisations on STEM topics. I believe his map captures the ideas in Computer Science in a concise and elegant manner, so I decided to combine categories from his map with categories identified by Pmernyei et al.

The full list of categories include:

- Computer file systems
- Databases
- Internet protocols
- Computer security
- Web technology
- Computational linguistics
- Programming language topics
- Computer architecture
- Operating systems
- Distributed computing architecture
- Computational complexity theory
- Information theory
- Cryptography
- Algorithms
- Computational theory
- Supercomputing
- Theoretical computer science
- Programming languages
- Computational science
- Augmented reality
- Human-computer interaction
- Internet of things
- Big data
- Hacking
- Machine learning
- Computer vision
- Image processing
- Artificial intelligence

- Natural language processing
- Robotics
- Graph theory
- Quantum computing
- Data structures
- Computer engineering
- Compilers
- Notable figures

Each of these base categories contain subcategories. For example, for the category *algorithms*, we have:

- Analysis of algorithms
- Optimization algorithms and methods
- Lossless compression algorithms
- Computer arithmetic algorithms
- Cryptographic algorithms
- Parsing algorithms
- Concurrency control algorithms
- Distributed algorithms
- Graph algorithms
- Sorting algorithms
- Classification algorithms
- Networking algorithms

These can be found in the Appendix under Areas of Computer Science.

The categoriser yields TSV files, with the first element being the title of the article, and the remaining elements being the categories the article is assigned to.

6.3.1 Storing Filtered Articles and their Contents

The 'store_mapping.py' file handles adding relevant mapping data (i.e. each article and the categories they are assigned to) into a table named **MAPPING**. Given a directory of mapping files, the code will iterate through them and check for articles that contain a listed subcategory. When this is satisfied, the article title and its list of mapped categories are inserted into the MAPPING table.

Once the article mapping information is stored, we can use it for the next step which is to store the contents of the articles identified to be in the field of Computer Science. The 'store_article_content.py' file handles this. The mwxml library mwxml is used to parse the XML datadumps that are decompressed from the bz2 files. The code checks that the current article is an article in the field of Computer Science (i.e. checks that its mapped categories include one from the list of identified Computer Science categories) and if so then its page ID, page title, revision information, and text are inserted as a record into the **ARTICLE_CONTENT** table.

It should be noted at this stage that some articles have the same page title but contain different text content. They will also have different page IDs, meaning one should not simply replace the other. When duplicate titles are encountered, they are all added to the database but care should be taken when retrieving content information for analysis later on.

6.4 Using Extracted Articles to Construct a Graph Structure

At this stage, we have identified articles in the field of Computer Science, and we also have the categories of Computer Science to which they are assigned to. In order to construct a graph structure from these articles which will serve as nodes, we need a way of relating their contents to their position in the overall graph structure.

Word2Vec is a technique for understanding natural language. Essentially, word2vec algorithms transforms words into numerical vectors, which capture the characteristics of the words. Doc2Vec is an extension of word2vec - it transforms a document into a vector. We are dealing with Wikipedia articles so the Doc2Vec approach is appropriate. The Doc2Vec model will serve as the base for which the graph is built on. It will generate a unique vector for each of the articles, describing their position in the overall graph structure.

6.4.1 Constructing a Corpus from Identified Articles

I believe a specialised corpus is appropriate for the Doc2Vec model as specialised corpora are often used when specialised language is expected. Articles in Computer Science will contain many technical terms and mathematical notations.

The 'build_corpus.py' file describes the process of extracting readable text from each article's content, and adding that to a corpus. Note that article content is in a markup template which contains many headings and meta data which may affect the quality of the corpus, so only the readable text (e.g. what we see when we view a Wikipedia page) is preserved.

The **Gensim** library gensim includes methods for filtering out Wikipedia markup leaving the readable text. Then, the text is converted to all lowercase, and line breaks are removed so the text of each article fits onto a single line. Punctuation marks except for commas and full stops are also removed through Regex.

The resulting corpus is simply a txt file. Each line represents an article and will be used later as an instance for training or testing. The tag is placed at the start of the line, this is the title of the article. A separator (I used | in this project) succeeds the tag, which is then followed by the filtered text content. The separator must not appear in any of the article titles nor their contents, if it did then an error would occur in splitting a line into its title and contents.

6.4.2 Training the Doc2Vec Model

The training and testing of the Doc2Vec model take place in the 'doc2vec_model.py' file. The corpus is split into a training set and a test set. I used an 80-20 split as this seems to be the convention for model training. The The *simple_preprocess* method of the Gensim library is called on the corpus text, and this removes tokens that are too long (¿15) or too short (¡2). The corpus is shuffled before it is split so that variance is reduced and the model remains general. This is mainly due to the fact that I'm not sure how Wikipedia order's their articles, but so far the records in the database are listed according to Wikipedia's ordering of articles.

The Gensim library is fairly easy to use. First, a Doc2Vec class which specifies parameters is initialised. Then, a vocabulary is built from the sequence of articles in the training set. Finally, the model's neural weights are learned through iteration. The resulting model can be used to produce a vector for any article describing its position in the vector space. Some experiments on fine-tuning parameters are shown below.

6.5 Dimension Reduction

It is difficult for humans to visualise and understand data of more than three dimensions. A vector space like this one which has many dimensions is hard to work with when it comes to visualisation. This calls for dimension reduction. High dimensional data is often reduced into 2-dimensional or 3-dimensional for the sake of information visualisation.

Additionally, there is often redundant information in the vectors, including related or duplicated features. Dimension reduction attempts to eliminate these by using the features to construct a lower dimensional space to offset the redundancy in the original set of features. All in all, dimension reduction will map important information from the original feature set to a smaller subset of features that are more concise.

There are many methods of dimension reduction. A common method is called Principal component analysis (PCA). It reduces the dimensionality of highly correlated data by converting the original set to a new set while preserving the global structure. T-distributed stochastic neighbourhood embedding (t-SNE) is another method of dimension reduction but it does not preserve global structure but rather local structure. It operates by embedding nodes from higher dimensions to a lower dimension while preserving the *neighbourhood* of that node. The Kullback-Leibler divergence records the difference between two probability distributions over the same variable. T-SNE tries to minimise this divergence value, which is how it preserves local structure.

T-SNE implementations often contain hyperparameters (e.g. perplexity, learning rate) which can be tuned for the application at hand whereas PCA is more stringent. T-SNE also handles outliers better. Taking these factors into consideration, I decided to use t-SNE for dimension reduction.

Dimension reduction is implemented in 'visualisation.py' and this process uses **OpenTSNE**'s implementation of t-SNE. The dimensionally reduced plot is displayed using the **matplotlib** library matplotlib. Note that t-SNE is a randomised algorithm so a fixed seed (i.e. 7) is used for reproducible results.

6.5.1 OpenTSNE

One feature that I personally was keen on having right from the start is to allow the user to look up any article and the Doc2Vec model will compute its vector in the Computer Science based graph structure. Initially, I used **Scikit-learn's** sklearn implementation of t-SNE. This is a transductive learner, meaning the model is only applicable to data points it was trained on. The reason is that sklearn's implementation of t-SNE does *not* learn a mapping function from a higher dimensional space into a lower dimensional space, it instead runs an iterative process attempting to reach an equilibrium where the loss is minimised.

In order to map a given point (with high number of vectors) that is not in the training set into a lower dimension, another method is needed. Through some research, I came across **OpenTSNE** opentsne. This is a Python implementation of t-SNE which allows the addition of new data points to existing embeddings, allowing predictions of a point's position to be made.

7 Experiments

In this chapter, we will review some experiments that were performed on various components of the project.

7.1 Fine-Tuning Parameters for the Doc2Vec Model

These experiment results are visualised through t-SNE (perplexity=50) dimension reduction into 2-dimensions.

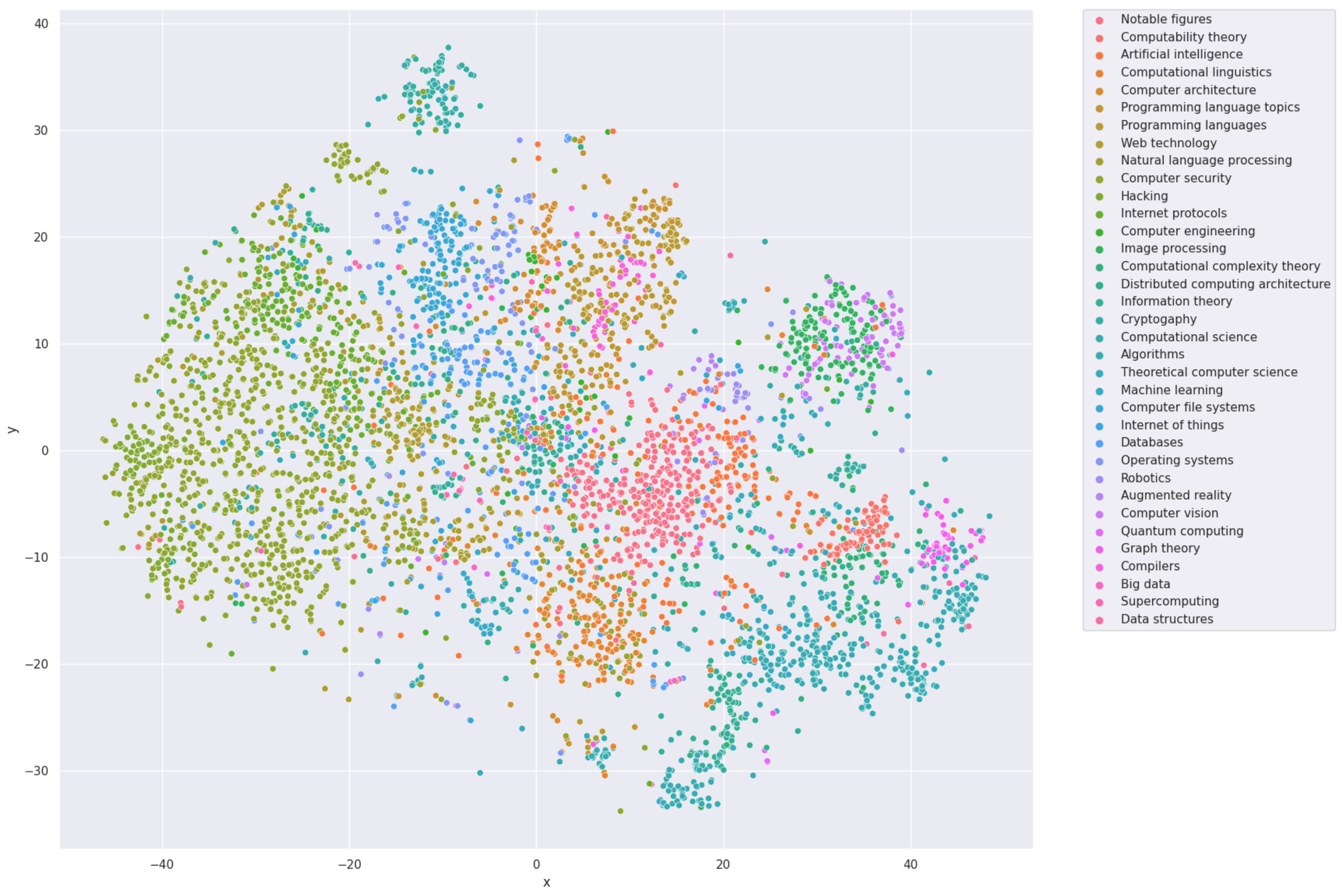


Figure 4: 30 Dimensions, 20 Epochs

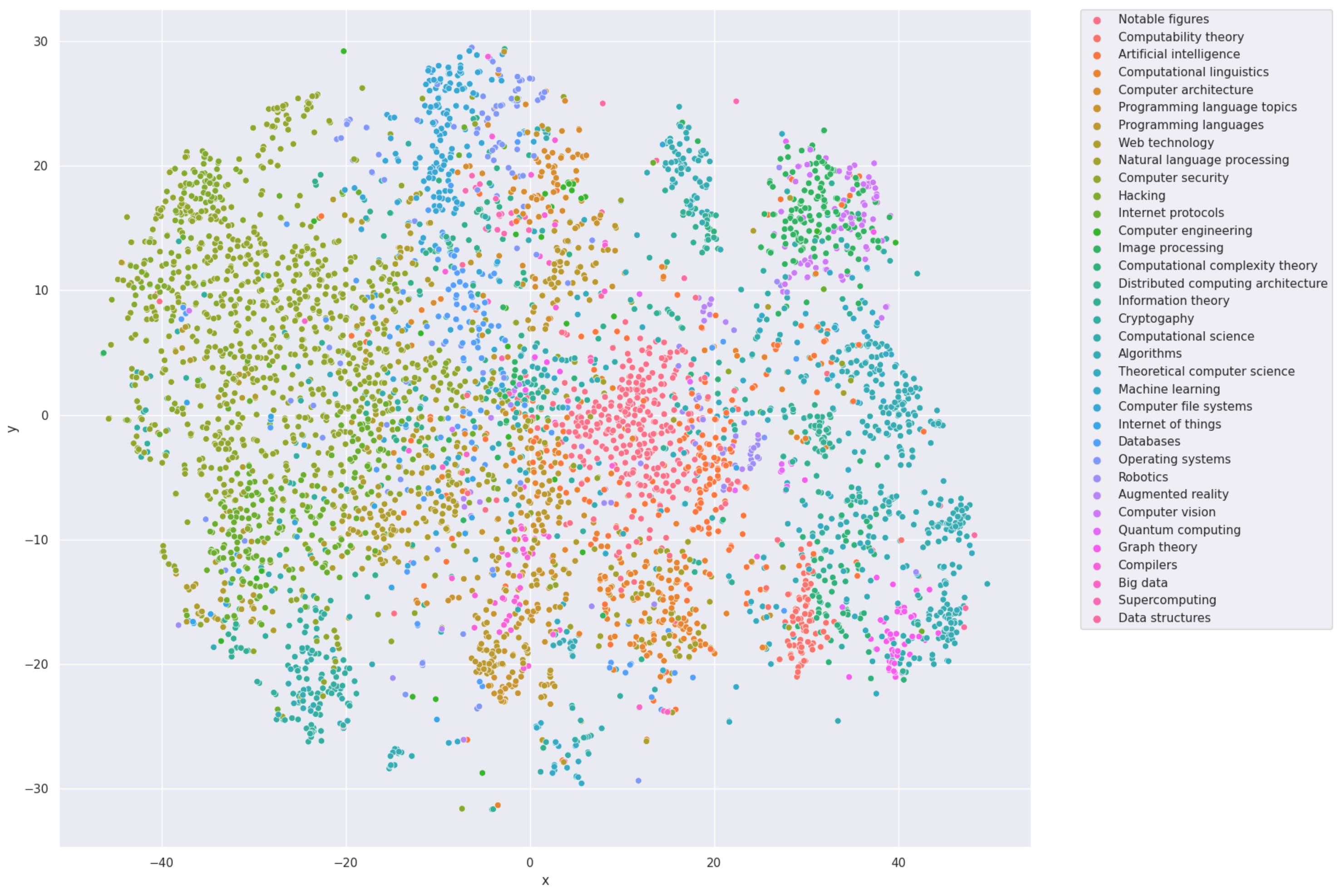


Figure 5: 30 Dimensions, 30 Epochs

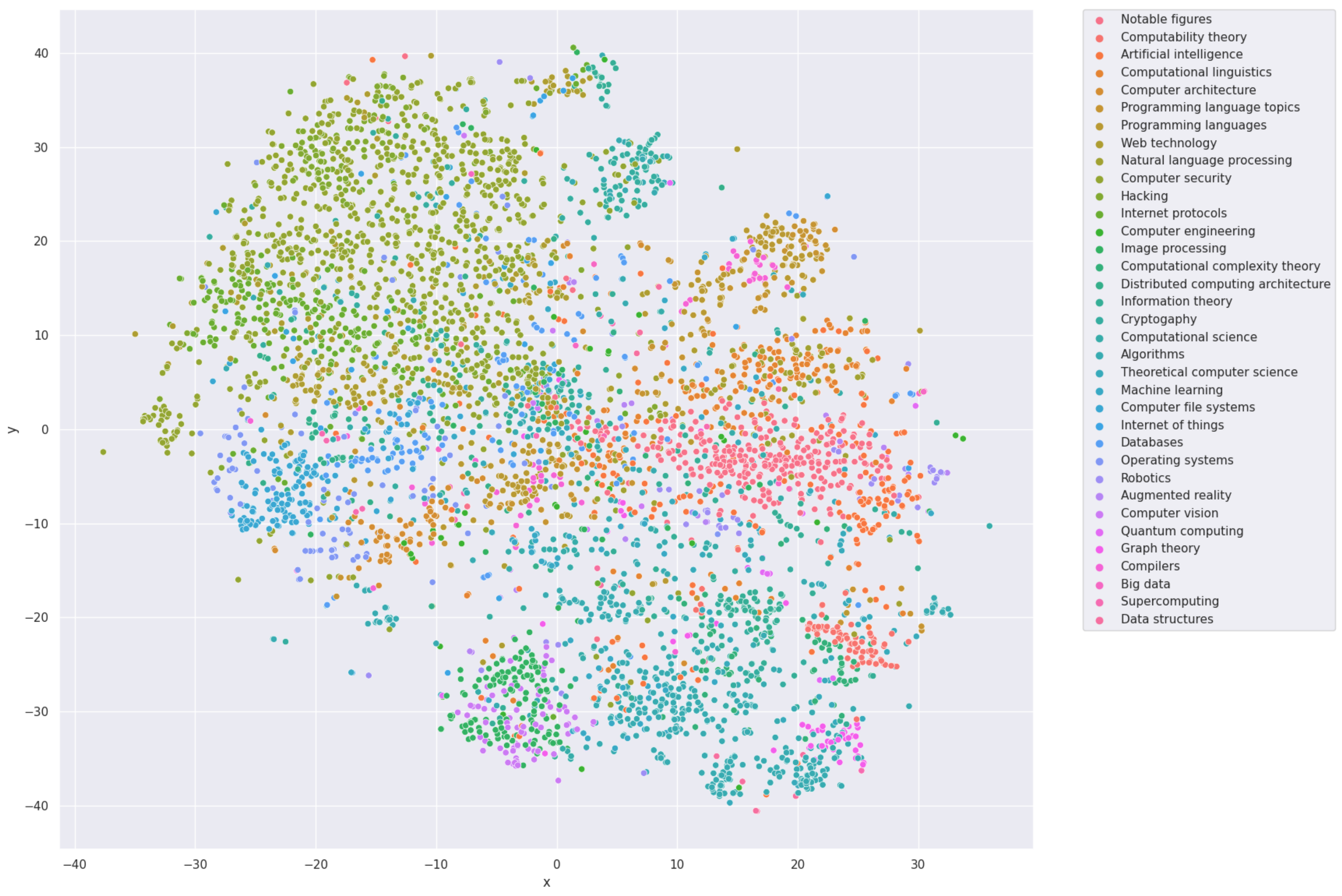


Figure 6: 40 Dimensions, 10 Epochs

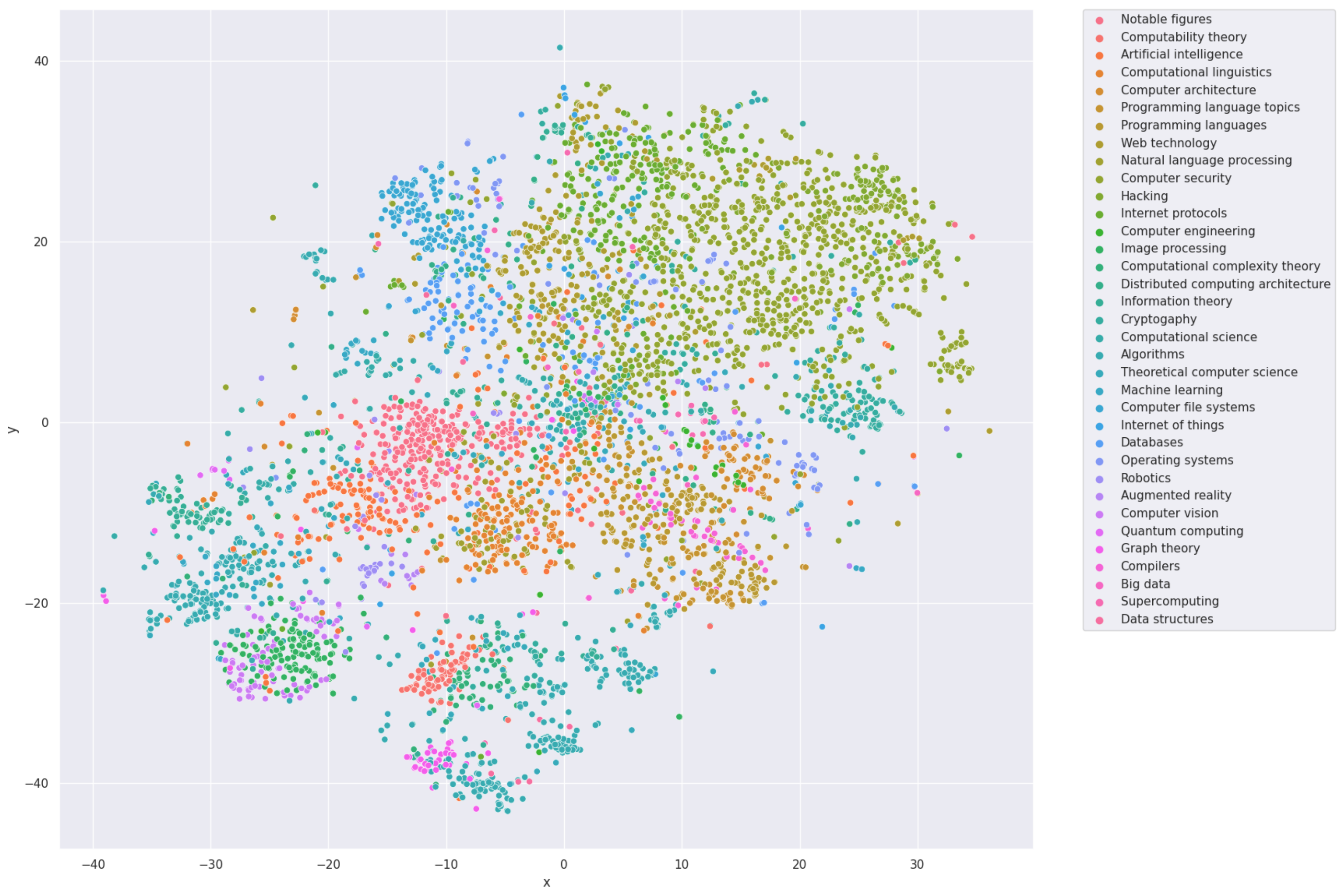


Figure 7: 40 Dimensions, 20 Epochs

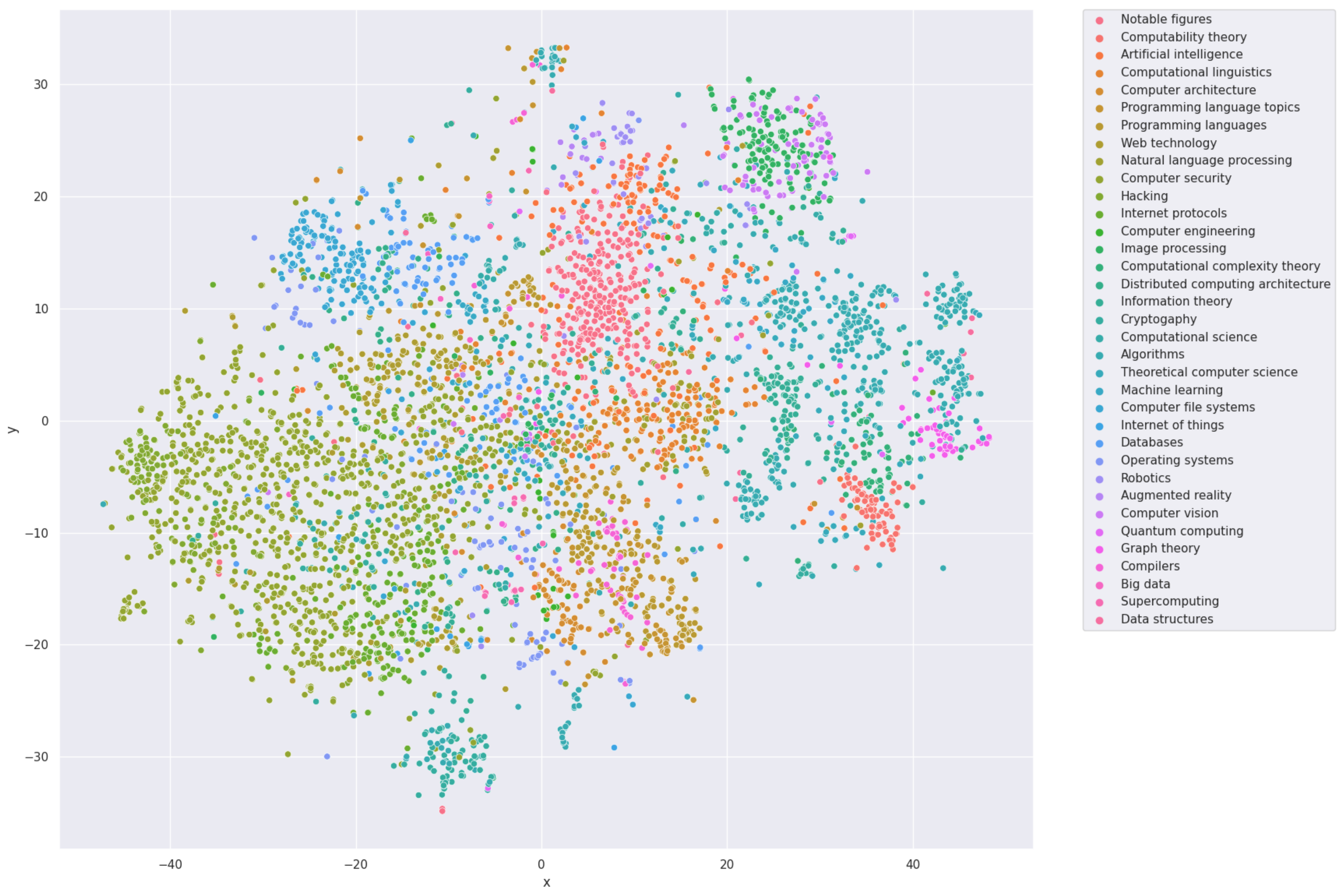


Figure 8: 40 Dimensions, 30 Epochs

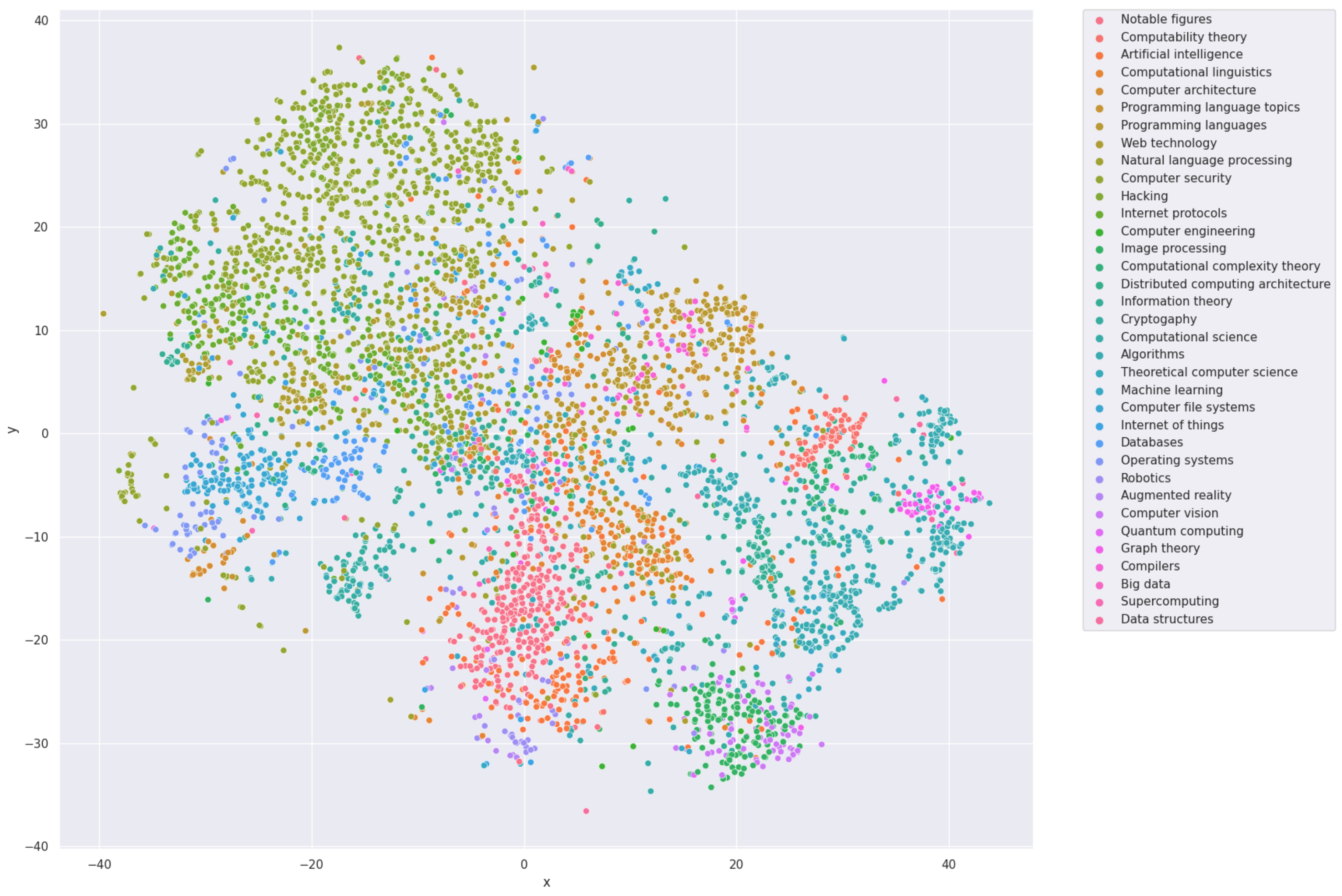


Figure 9: 50 Dimensions, 10 Epochs

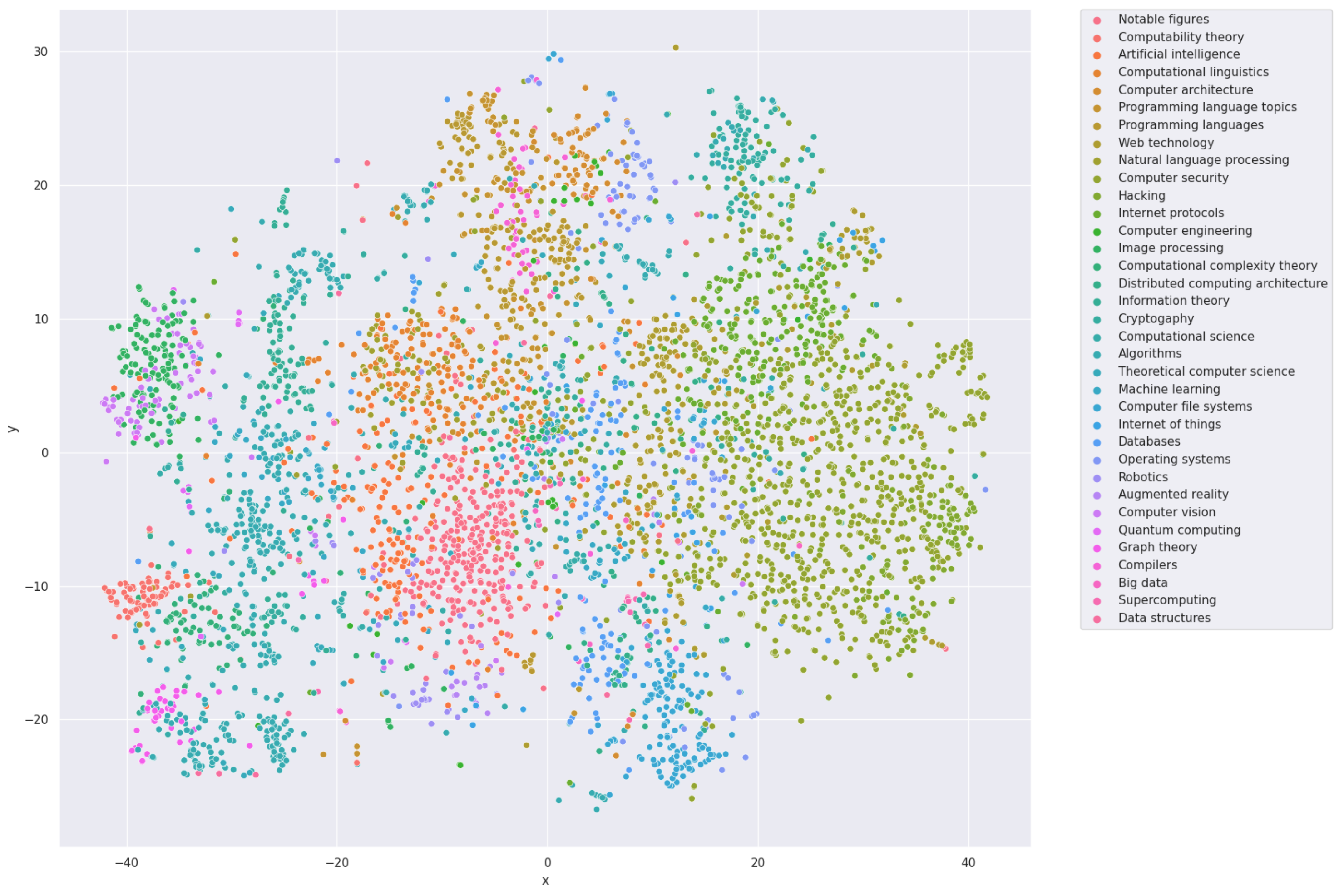


Figure 10: 50 Dimensions, 30 Epochs

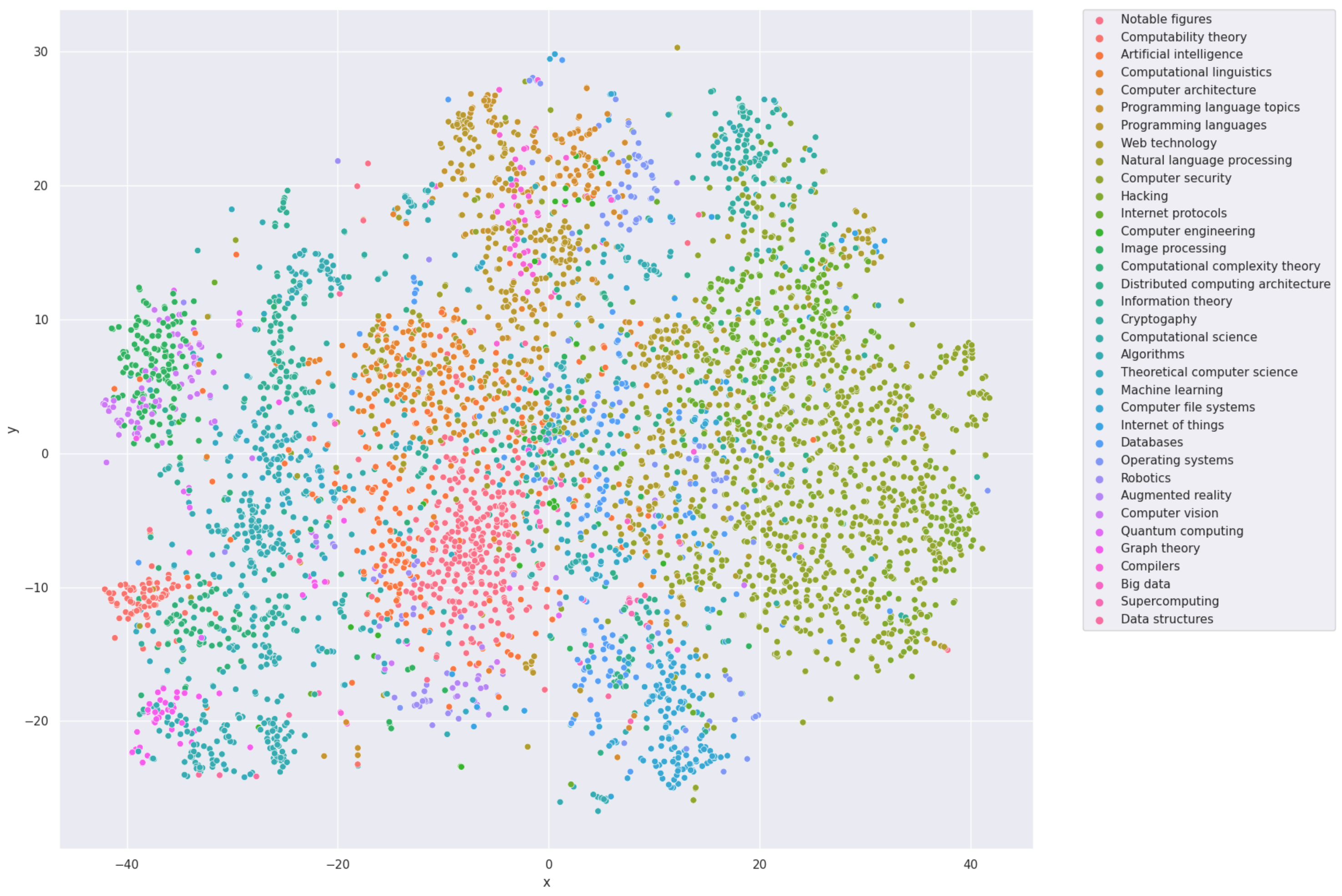


Figure 11: 50 Dimensions, 50 Epochs

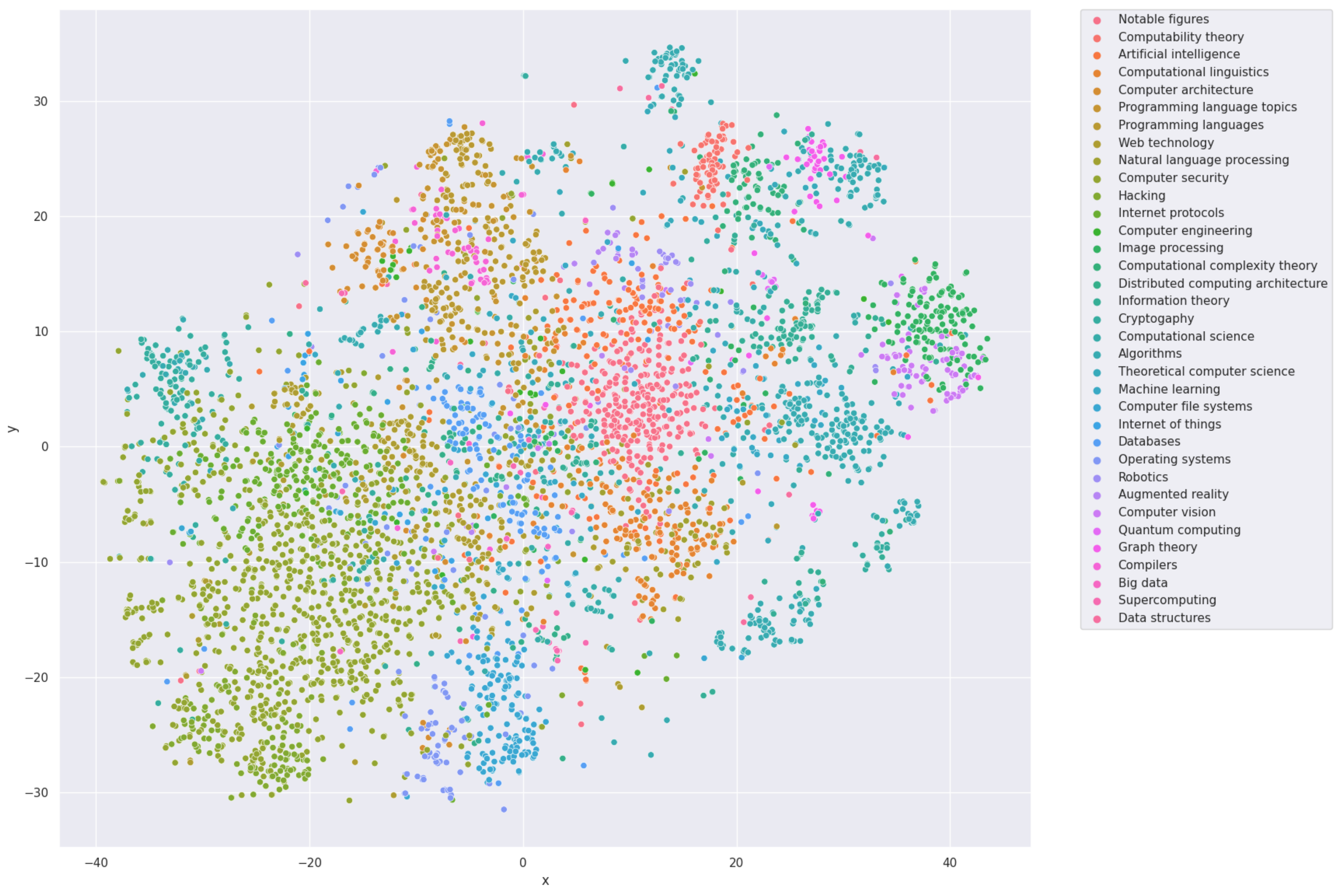


Figure 12: 50 Dimensions, 80 Epochs

Observing by eye, I would say 40 dimensions yield a graph structure with the clearest clusters. Among the 40 dimensions runs, the 30 epochs forms the best combination of parameters with it. But overall, I would say the results are not too different.

7.2 Doc2Vec Model Accuracy Summary

The Doc2Vec workflow is different from the usual machine learning workflow, even though they both contain a 'blackbox' component. In machine learning, the data set is split into a set for training and a set for testing, a conventional split is 80-20. The machine learning model is then trained on the training set then tested on the test set. It is important to ensure that the test set is not seen until the final testing stage.

For this project however, we are not learning a mapping to categorical values, and we have no idea how 'true' the model's computed vector for an article is as there is no ground truth to compare the result against. Therefore, testing is not performed like in machine learning.

7.2.1 Testing on Unseen Articles

With that being said, we can still leave a few articles out of the training set, and once the model is trained, infer the vectors for them and compare the results by eye. Gensim has a *most_similar* function, which returns the top-N most similar keys (i.e. articles) to the given articles and this can be used to describes how similar articles are to themselves. For example, testing five random documents (ran with 30 epochs and 20 dimensions as the model parameters):

Most similar for Apache Samza:

- Fuse Services Framework
- Apache CXF
- Sun Web Developer Pack
- Apache Axis2
- List of web servicce frameworks

Most similar for Gordon–Loeb model:

- Economics of security
- Profit impact of marketing strategy
- Single-loss expectancy
- Information assurance
- World Database of Happiness

Most similar for Bring your own operating system:

- Physical access
- Mount (computing)
- Public computer
- Sleep mode
- Hibernation (computing)

Most similar for HPE Helion:

- Altibase
- Red Hat Gluster Storage
- FlexiScale

- Ceph (software)
- Hekaton (database)

Most similar for Systat (protocol):

- Host Monitoring Protocol
- TCP Port Service Multiplexer
- Apache Rampart module
- FTP bounce attack
- Improper input validation

Most similar for Relying party:

- Pool User
- Site Security Handbook
- Online Certificate Status Protocol
- OAuth
- Rublon

I'm not well-versed enough in these topics to know how similar they are but through a quick Google definition search the results seem fitting. This experiment was also repeated a couple of times, and the results are fairly consistent, with the only changes being the order of the similar articles.

7.2.2 Assessing the Model in a Different Manner

Using this similarity metric described in the previous section, we can assess our trained model. For each article, we can compute the similarity scores for all articles in the corpus and rank them. Assuming the model is well trained, the top ranked article should always be the article itself.

A counter object can be used to keep track of the ranks with respect to the corpus. A sample result is shown below:

```
Counter({0: 2223, 1013: 2, 1253: 2, 2120: 1, 289: 1, 658: 1, 15: 1, 448: 1, 1629: 1,
363: 1, 1097: 1, 784: 1, 2197: 1, 2038: 1, 2: 1, 1369: 1, 336: 1, 1268: 1, 1735: 1,
1354: 1, 38: 1, 406: 1, 1829: 1, 1678: 1, 868: 1, 813: 1, 1421: 1, 313: 1, 1680: 1,
2118: 1, 1357: 1, 1360: 1, 2132: 1, 654: 1, 656: 1, 440: 1, 1865: 1, 481: 1, 1323:
1, 1625: 1, 445: 1, 1077: 1, 1405: 1, 1725: 1, 2025: 1, 680: 1, 604: 1, 279: 1, 938:
1, 51: 1, 1286: 1, 1777: 1, 1473: 1, 151: 1, 2191: 1, 1883: 1, 1598: 1, 969: 1, 1910:
1, 1250: 1, 1460: 1, 2110: 1, 373: 1, 1949: 1, 616: 1, 84: 1, 1833: 1, 2158: 1, 602:
1, 1291: 1, 1251: 1, 22: 1, 1742: 1, 2155: 1, 1556: 1, 911: 1, 1778: 1, 739: 1, 1886:
1, 1201: 1, 1738: 1, 846: 1, 7: 1, 2150: 1, 1149: 1, 80: 1})
```

The first element of the counter is the most important one. It tells us the number of articles whose most similar article was itself. This number divided by the total number of articles (I.E. 5230) is taken as the accuracy of the model. A summary table of the accuracy values with different hyperparameters is displayed below:

Dimension	**Epoch**	**No. Articles whose Most Similar Article is Itself** (/5230)	**Accuracy (%)**
30	20	5010	95.8
30	30	5011	95.8
40	10	5000	95.6
40	20	5007	95.7
40	30	5009	95.8
50	10	5004	95.7
50	30	5009	95.8
50	50	5015	95.9
50	80	5020	96.0

Full code outputs can be found in the Appendix.

7.2.3 Fine-Tuning Parameters for t-SNE

The parameters were fine-tuned using 40 dimension and 30 epoch model, which are the final perimeters that were decided on for the Doc2Vec model.

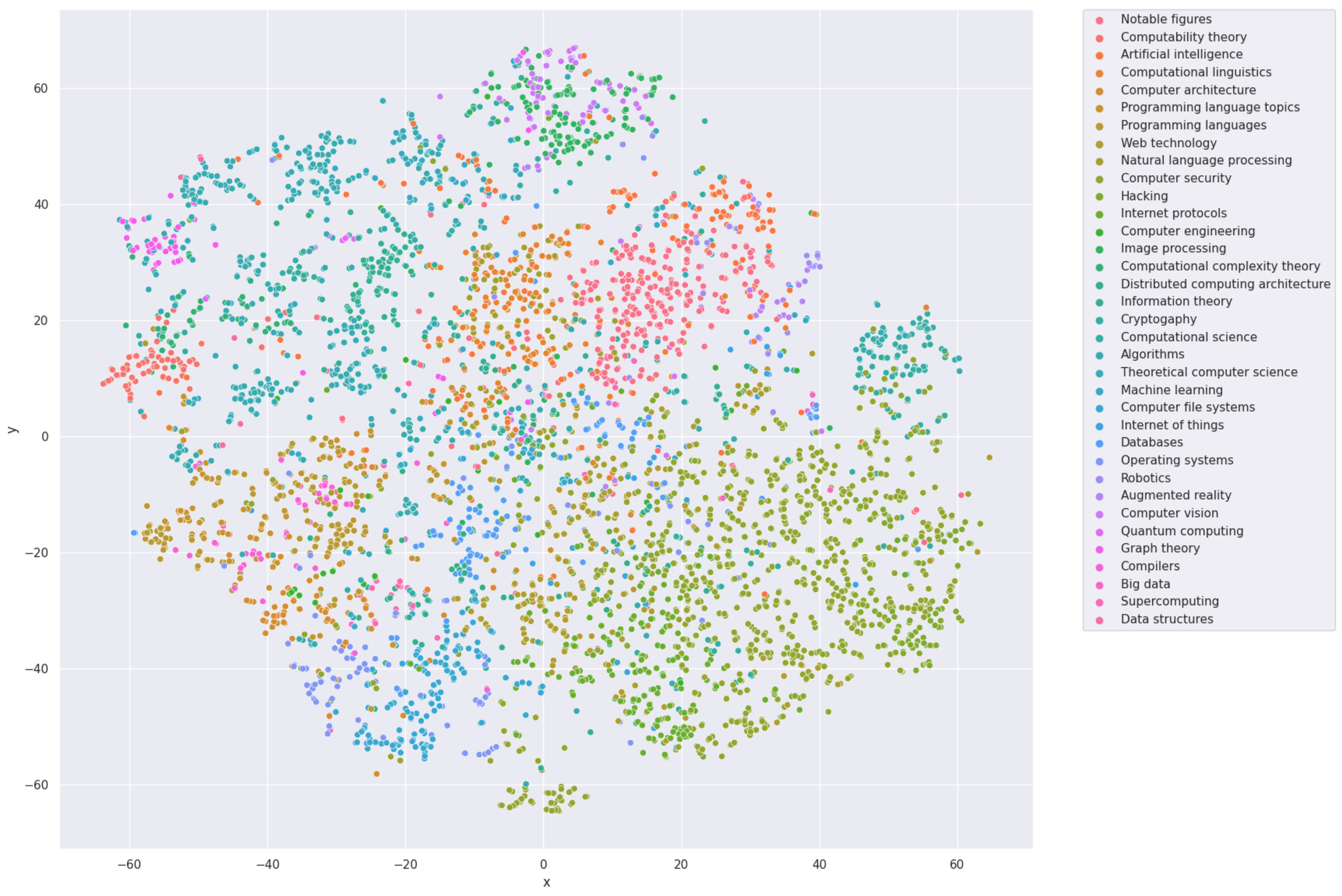


Figure 13: Perplexity = 10

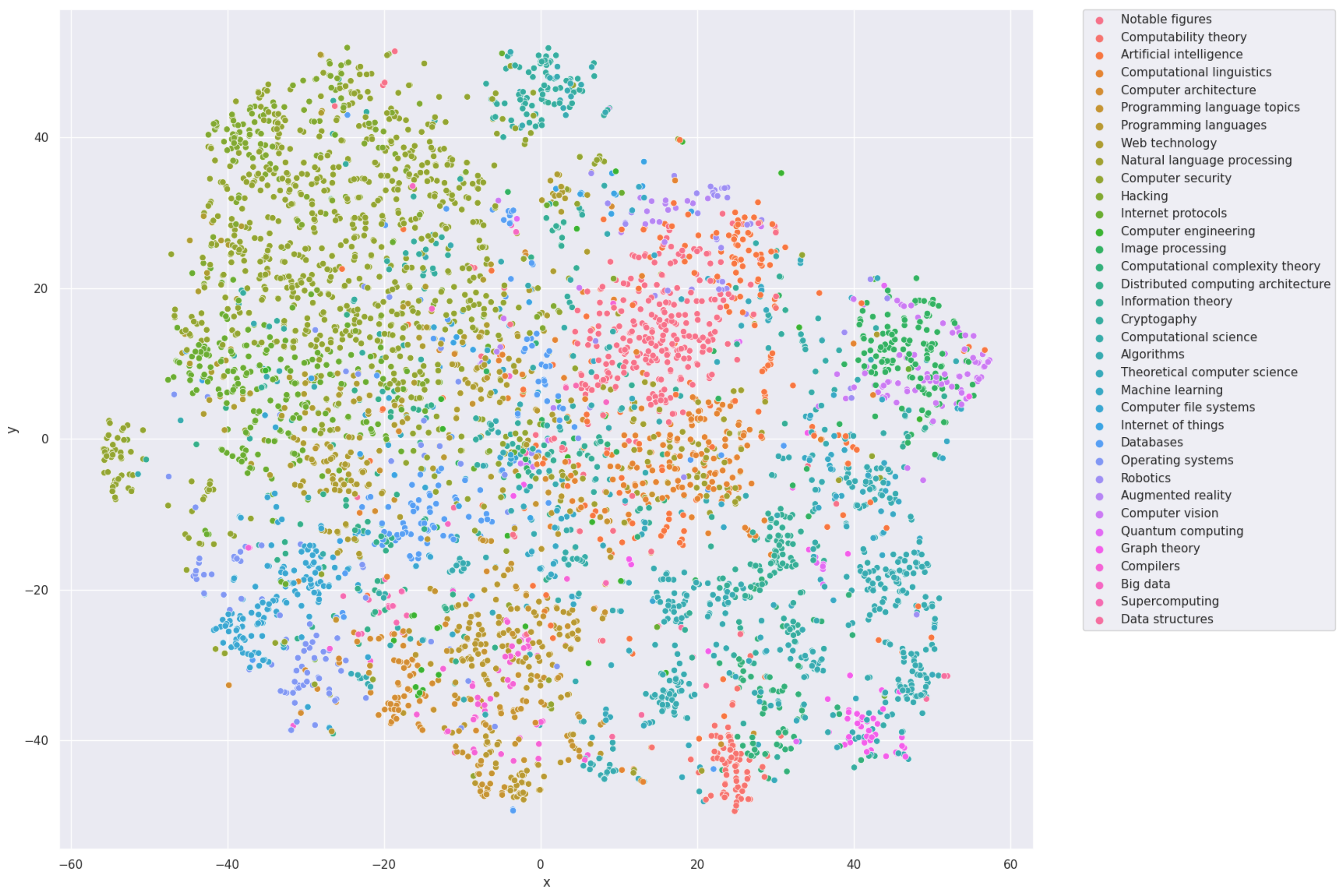


Figure 14: Perplexity = 20

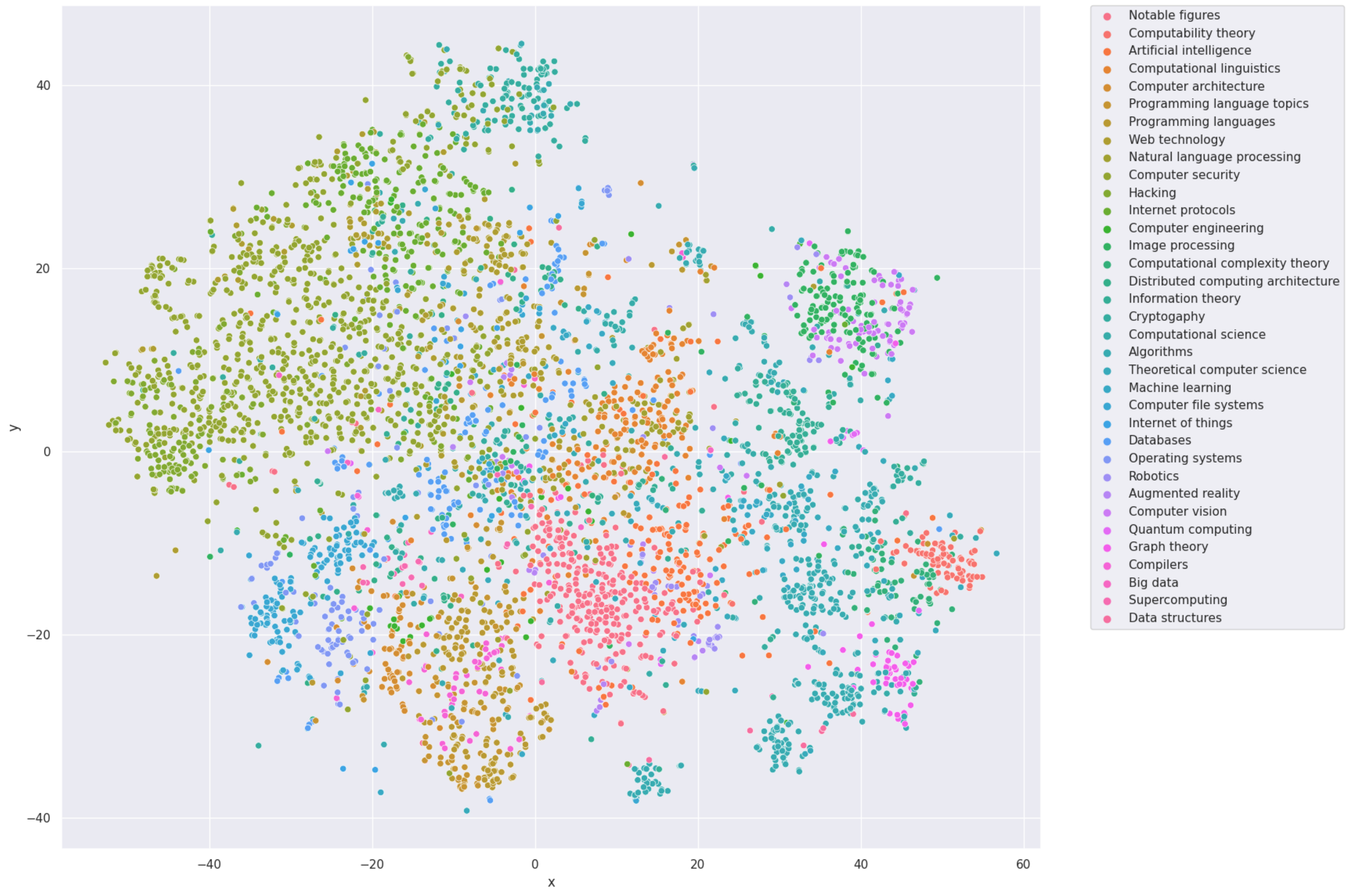


Figure 15: Perplexity = 30

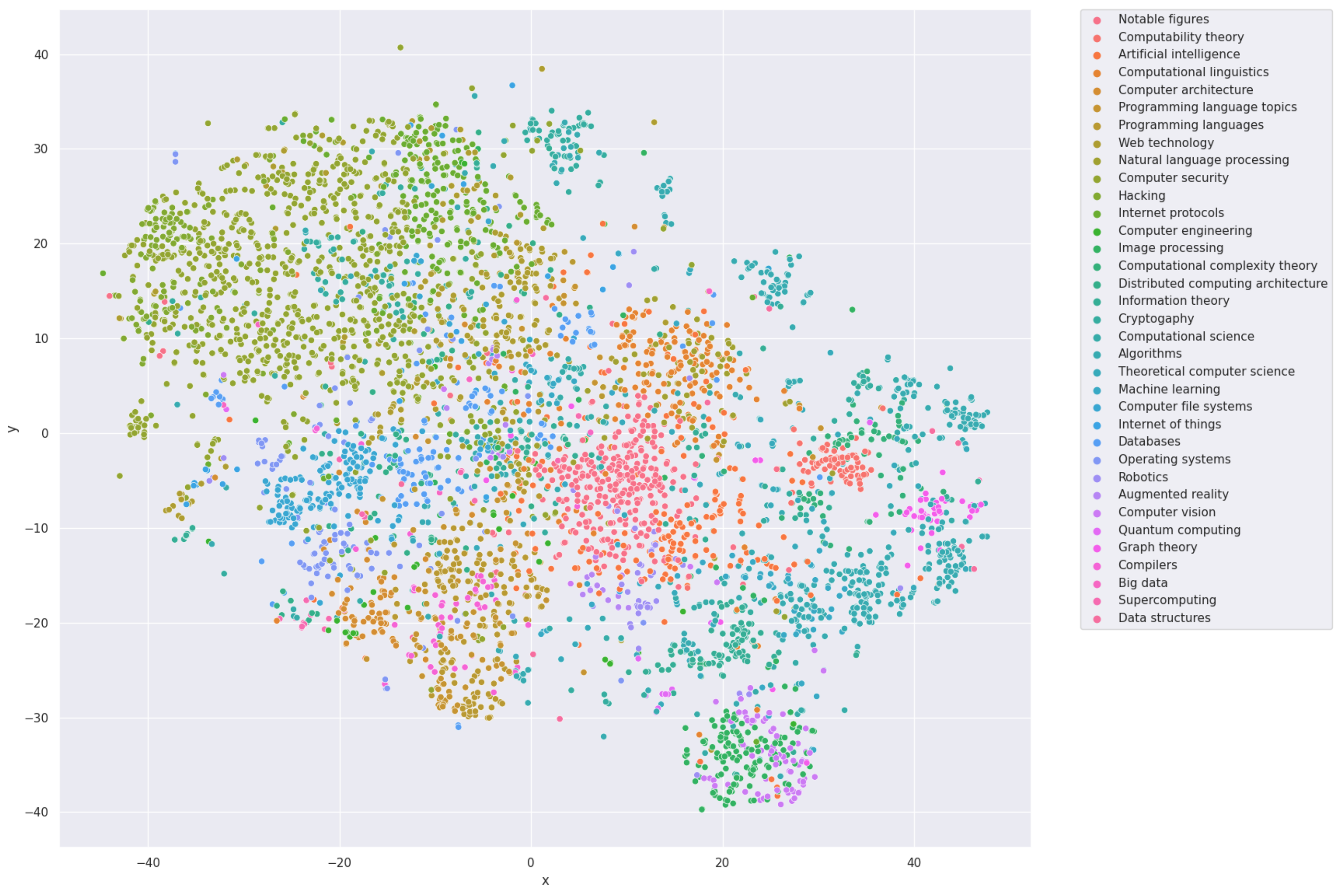


Figure 16: Perplexity = 40

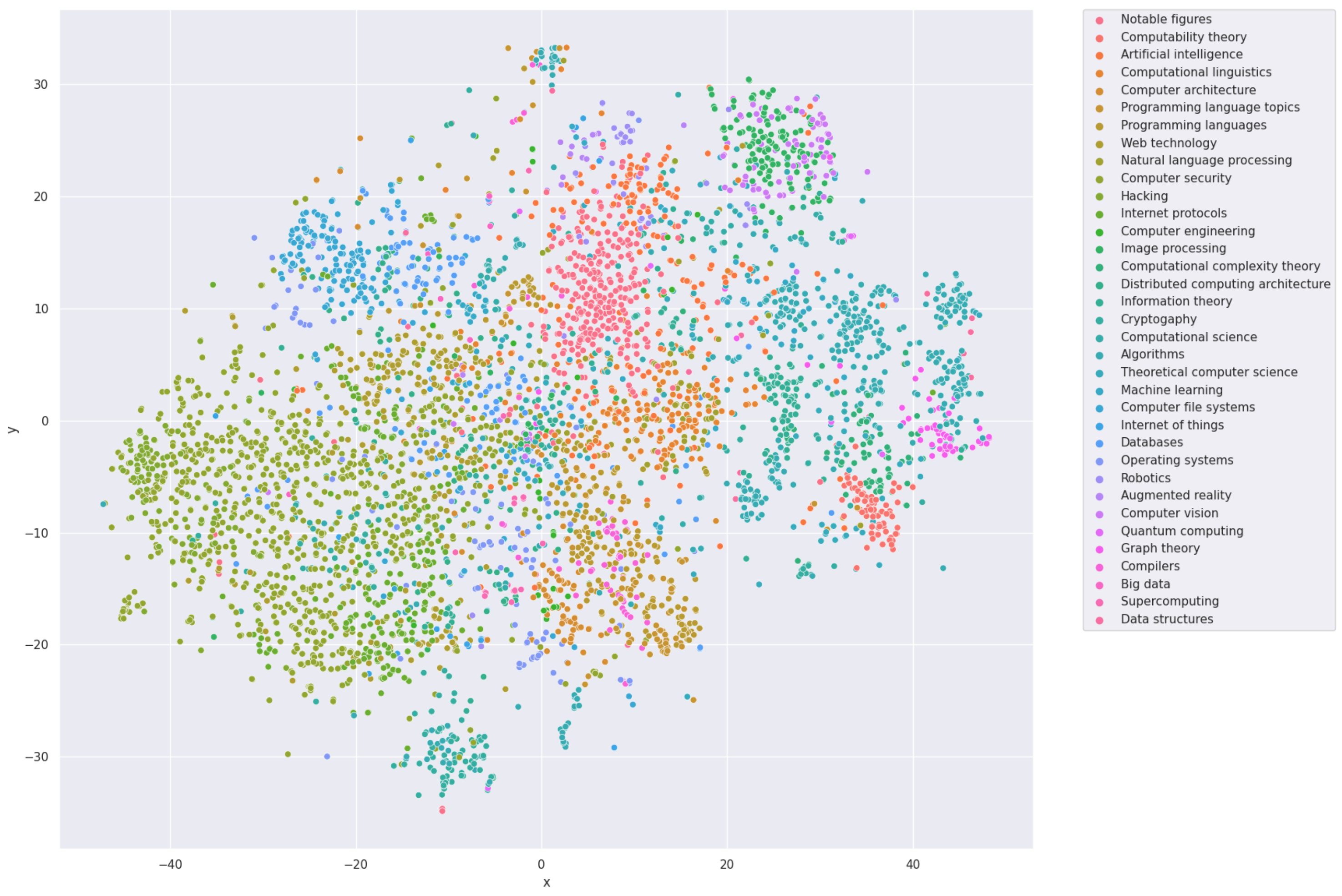


Figure 17: Perplexity = 50

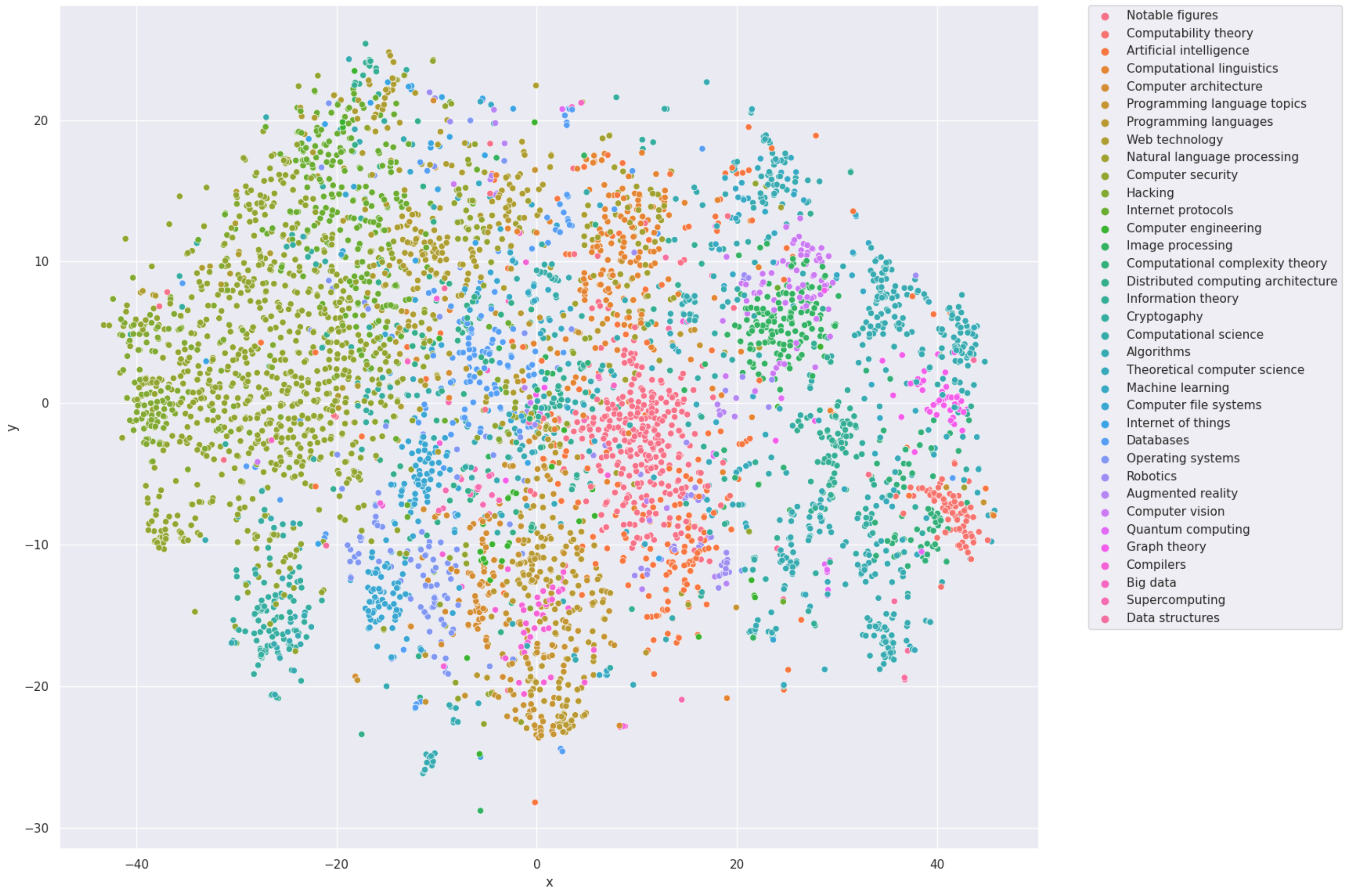


Figure 18: Perplexity = 60

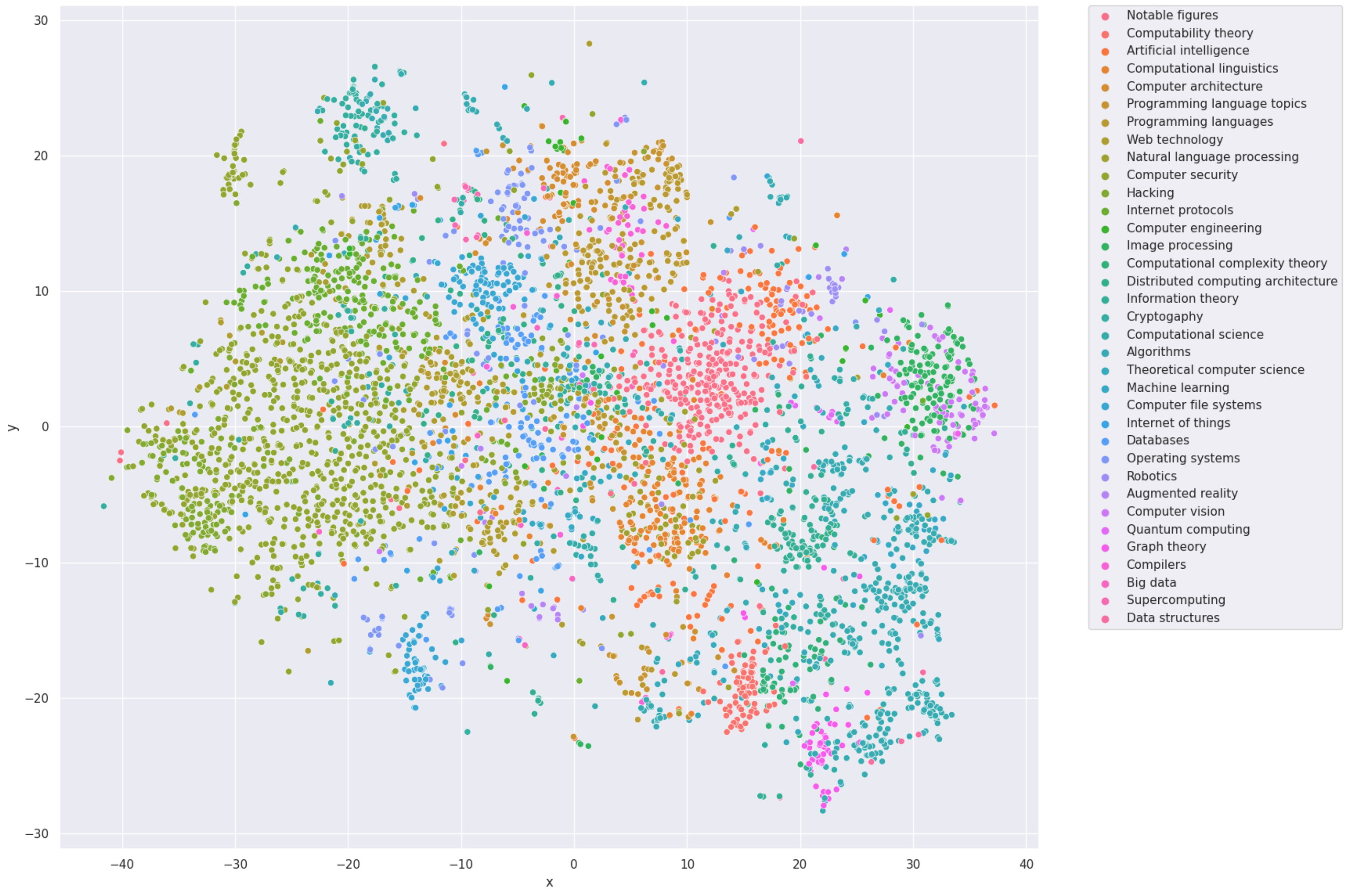


Figure 19: Perplexity = 70

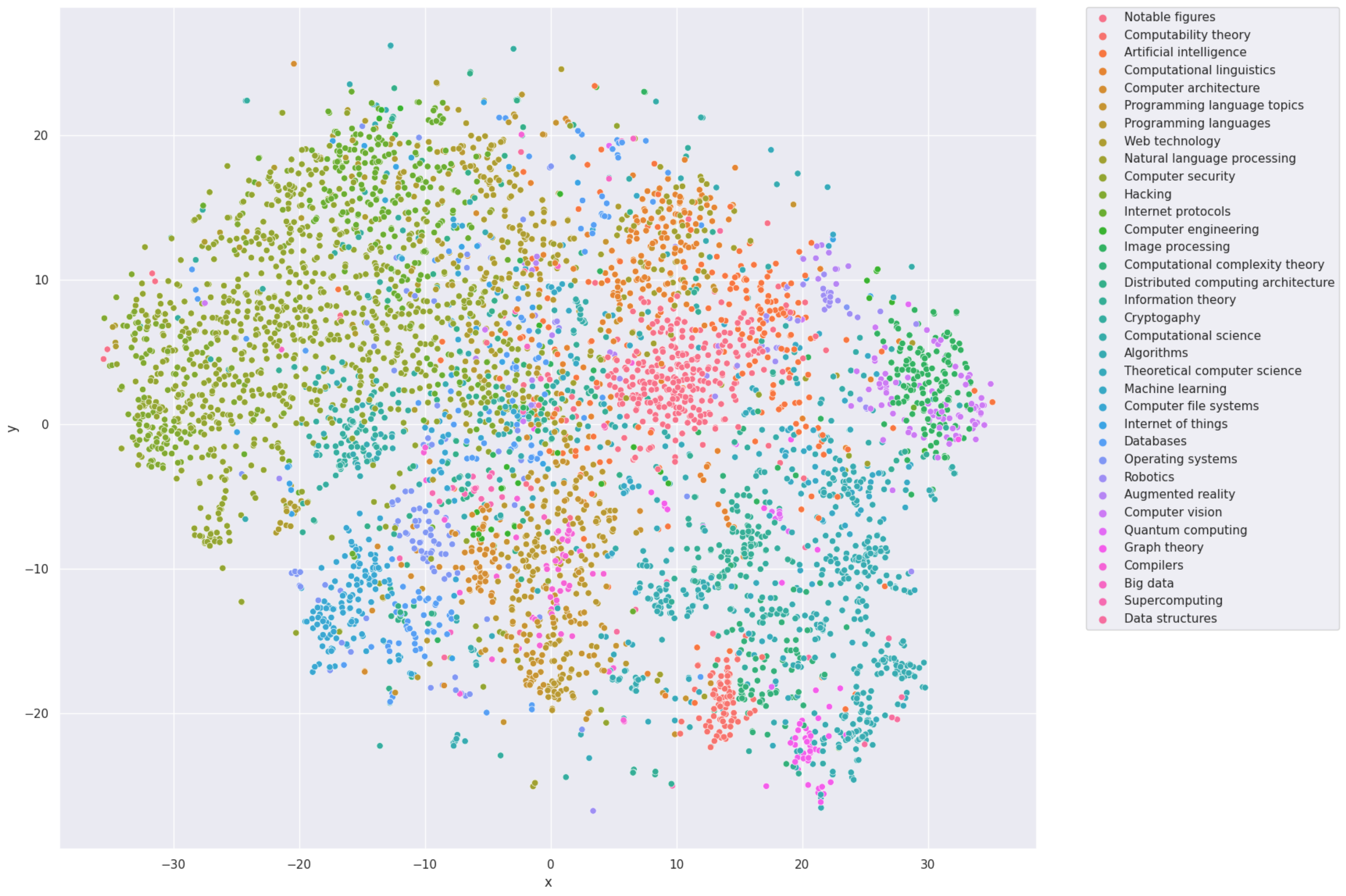


Figure 20: Perplexity = 80

I also tried out the *cosine distance* method. The notion of Euclidean distance breaks down in high dimensions, so t-SNE traditionally does not preserve global distance. OpenTSNE provides cosine distance as an option and the usage should be considered according to the context in which the project resides in. Cosine distance is all about the angle from the origin. Cosine similarity (i.e. correlation) is greater for articles with similar angles from the origin. Euclidean distance on the other hand, is lowest between articles with the same distance *and* angle from the origin, meaning two articles with the same angle may have a far Euclidean distance.

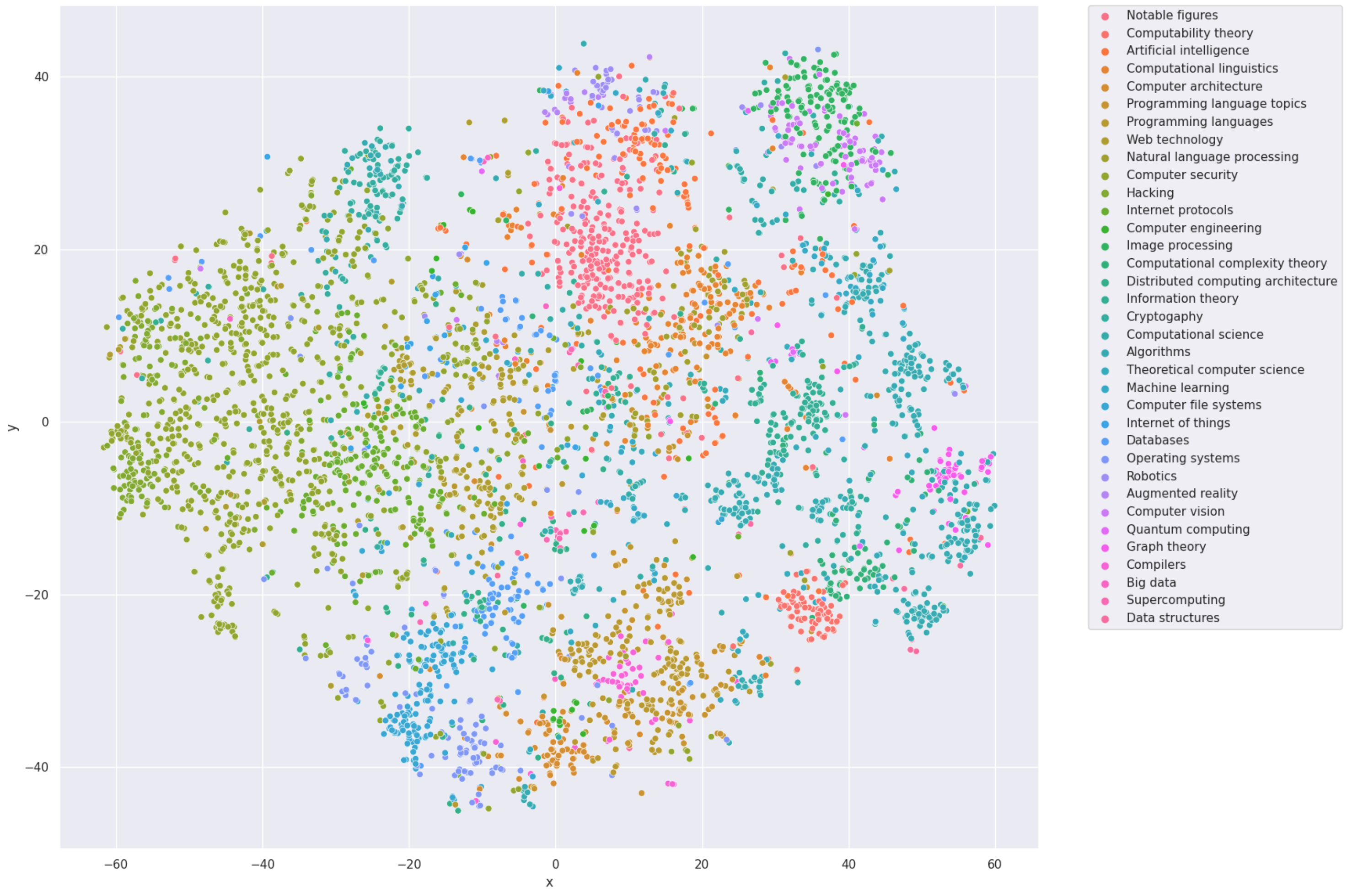


Figure 21: Cosine Distance, Perplexity = 30

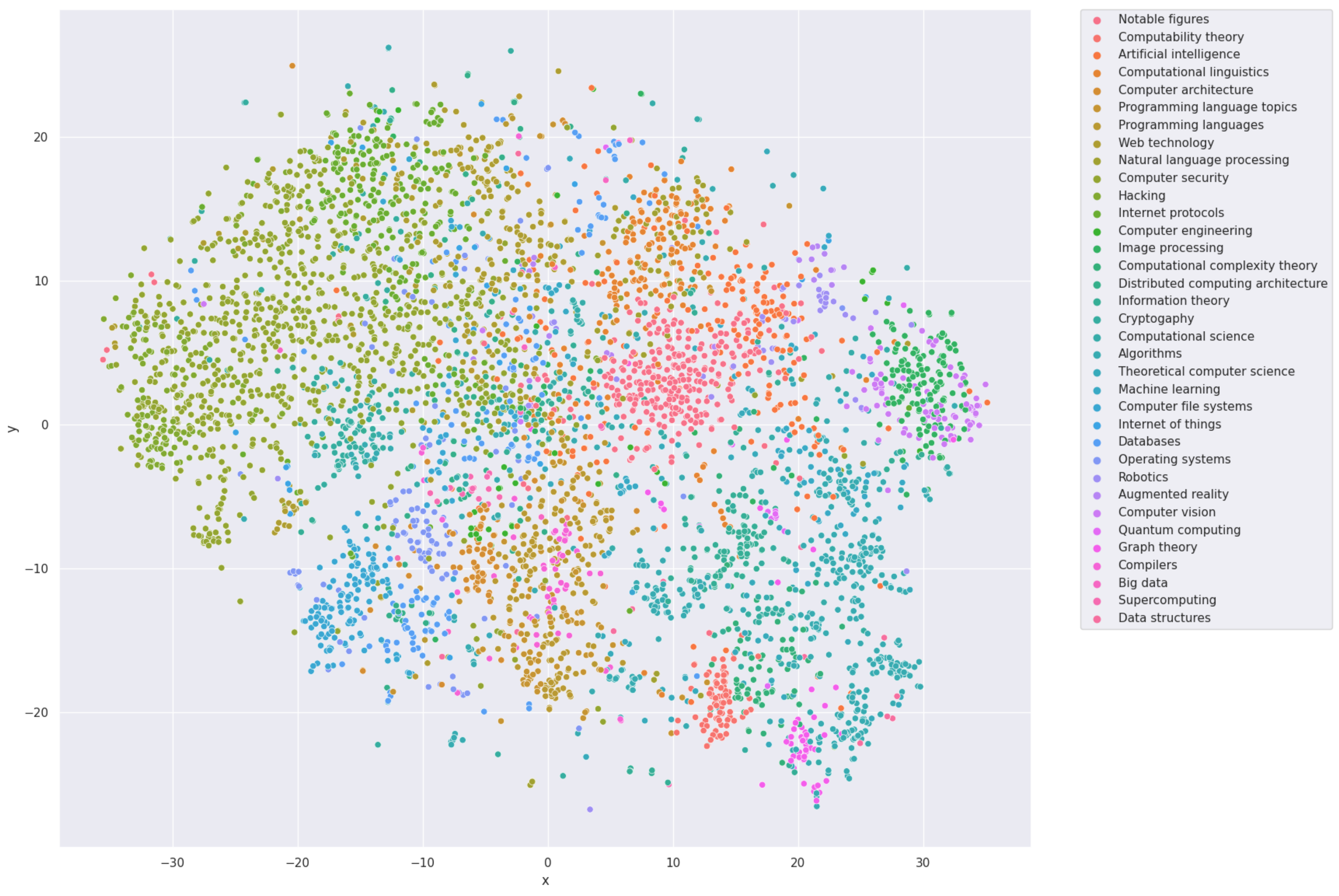


Figure 22: Cosine Distance, Perplexity = 50

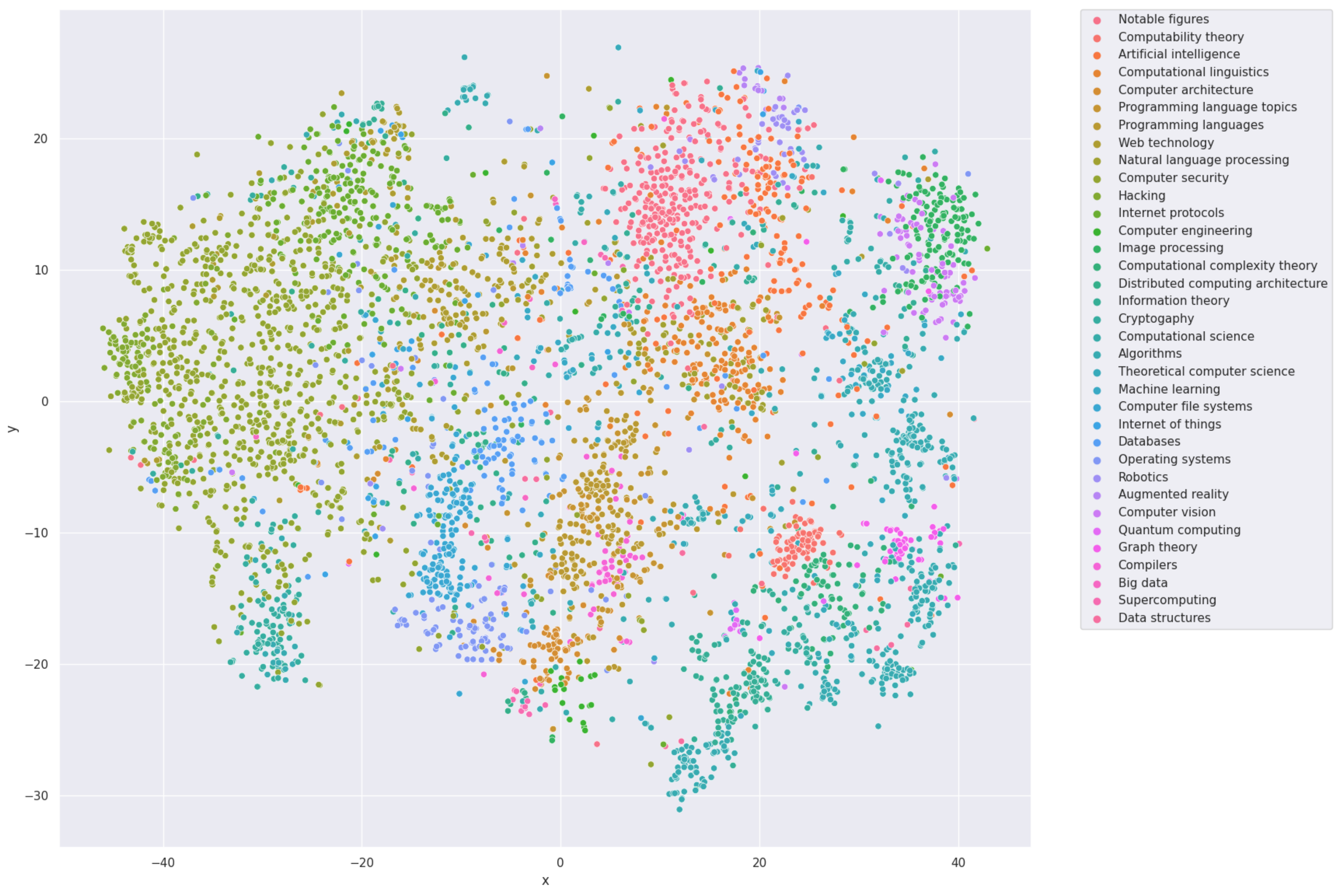


Figure 23: Cosine Distance, Perplexity = 70

Perplexity=10, 20, 30 yield relatively sparse plots compared to higher perplexities. It almost looks as if categories are further divided into subcategories. This makes sense as the perplexity value represents the number of neighbours the algorithm pays attention to, so the lower the value, the more granular the clustering will be.

The plots for perplexity=40 and upwards are not too different. I ended up going for perplexity=50.

7.3 Generating a Graph Structure using only Hyperlinks - NetworkX

Another way of constructing the graph structure is to simply use the intrinsic hyperlink structure of the articles. **NetworkX** is a Python package often used for network analysis, it contains an implementation of the Fruchterman-Reingold layout[10] which generates a force-directed (i.e. aesthetically pleasing) graph. It uses the analogy of springs as edges that pull connected nodes towards each other while pushing away nodes that are far from one another.

I believe this algorithm fits nicely in this case, as hyperlinks can be seen as connections between article nodes. My usage goes something like this: iterate over the list of articles and add them all as nodes. Then for each article, find all of its hyperlinked articles using Regex and add an edge from the article to each of its hyperlinks that are also in the structure (i.e. is a node in the structure). Note that some hyperlinks take the form of multiple titles separated by a bar symbol, like 'computer file—file'. We need to split the hyperlink text into its individual titles when this occurs.

Below is a sample output of information of the learnt layout, ran with the first 20 articles in the database:

```
Graph with 20 nodes and 15 edges
Nodes:  ['algorithm', 'alan turing', 'analysis of algorithms', 'artificial intelligence',
'cipher', 'ada (programming language)', 'buffer overflow', 'advanced encryption standard',
'apl (programming language)', 'basic', 'kolmogorov complexity', 'list of artificial
intelligence projects', 'asynchronous transfer mode', 'ai-complete', 'file archiver',
'ackermann function', 'bluetooth', 'bqp', 'b (programming language)', 'bra{ket notation']
Edges:  [('algorithm', 'alan turing'), ('algorithm', 'analysis of algorithms'), ('algorithm',
'artificial intelligence'), ('algorithm', 'ackermann function'), ('algorithm', 'bqp'),
('algorithm', 'cipher'), ('alan turing', 'artificial intelligence'), ('alan turing',
'cipher'), ('artificial intelligence', 'apl (programming language)'), ('artificial
intelligence', 'list of artificial intelligence projects'), ('artificial intelligence',
'ai-complete'), ('cipher', 'advanced encryption standard'), ('ada (programming language)',
'buffer overflow'), ('advanced encryption standard', 'bluetooth'), ('apl (programming
language)', 'basic')]
```

I will briefly explain relevant parametres of NetworkX's Fruchterman-Reingold algorithm implementation:

- k - optimal distance between nodes. The higher this value is, the further nodes will be to each other
- iterations - maximum number of iterations allowed
- threshold - the difference in node positions is calculated through each iteration, and if this value is below the given threshold, iteration stops as the optimal layout has been found

The resulting graph is highly dependent on the value of k used. I adjusted the k value so that the most connected nodes are placed in the centre.

Including labels in the graph structure allows us to understand the graph a lot better. However, due to the high volume of article nodes and the their high connectivity, labelling all nodes is not a good approach because the labels would all be overlaid. Therefore, I used a dictionary object to find the number of connections (i.e. hyperlinks) of each node, and then the dictionary is ordered from most connected to least connected. Now, we can find the top N most connected nodes and we can choose to only label these.

7.3.1 First 100 Articles

Graph has 100 nodes and 319 edges.

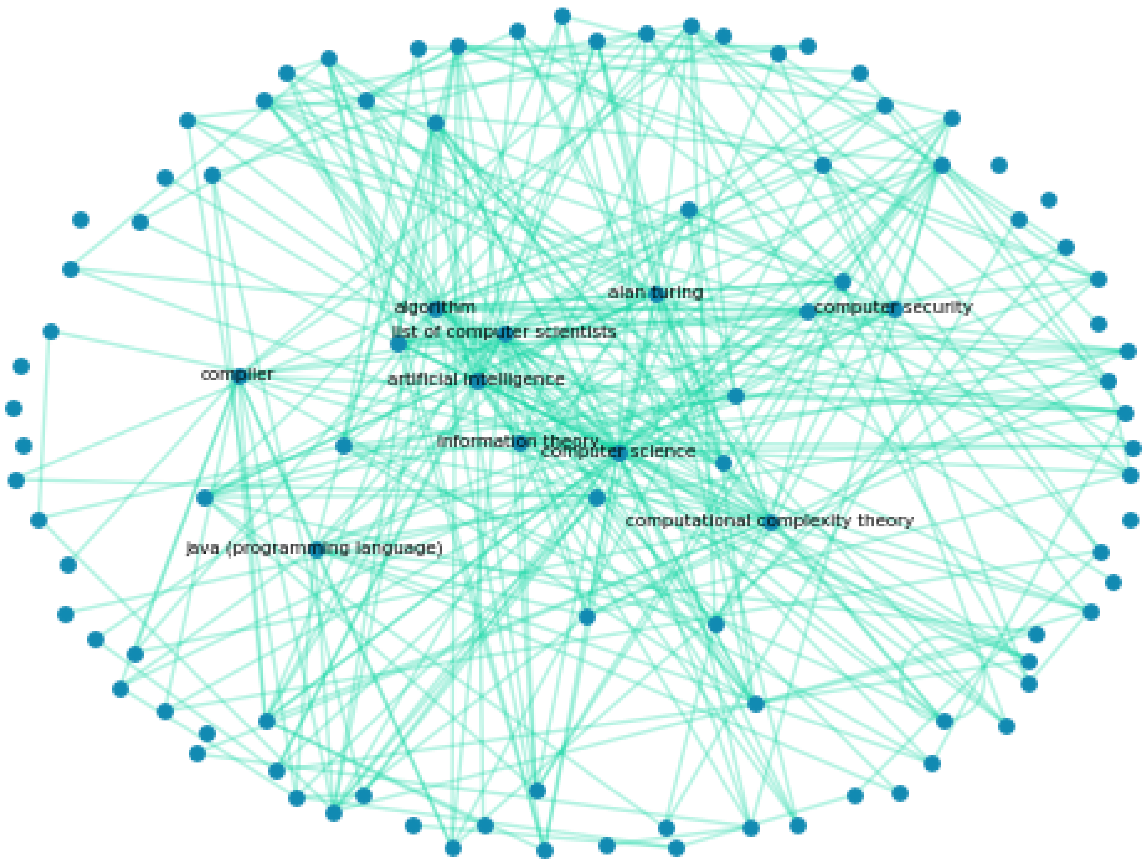


Figure 24: Connectivity of 100 Articles

7.3.2 First 500 Articles

Graph has 500 nodes and 2597 edges.

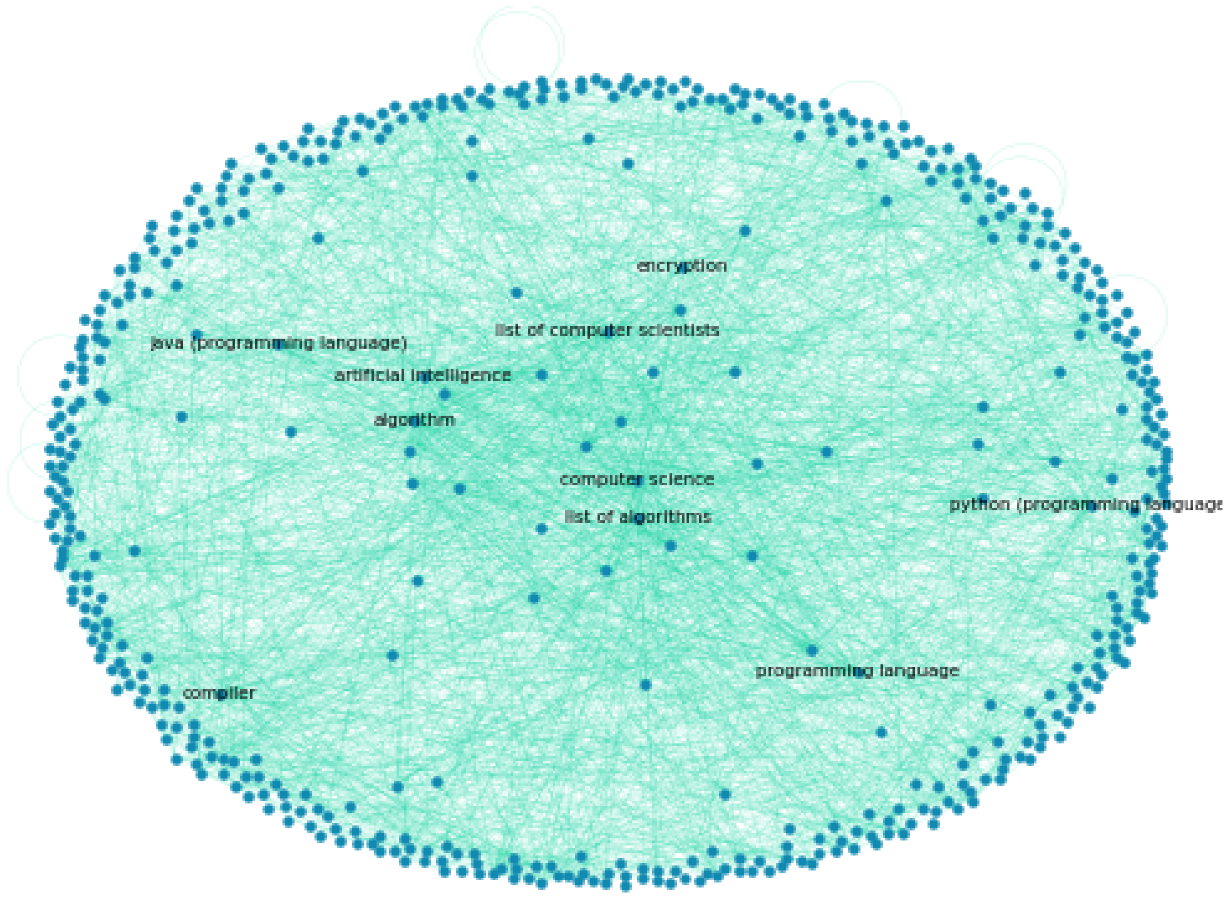


Figure 25: Connectivity of 500 Articles

7.3.3 First 1000 Articles

Graph has 970 nodes and 5182 edges.

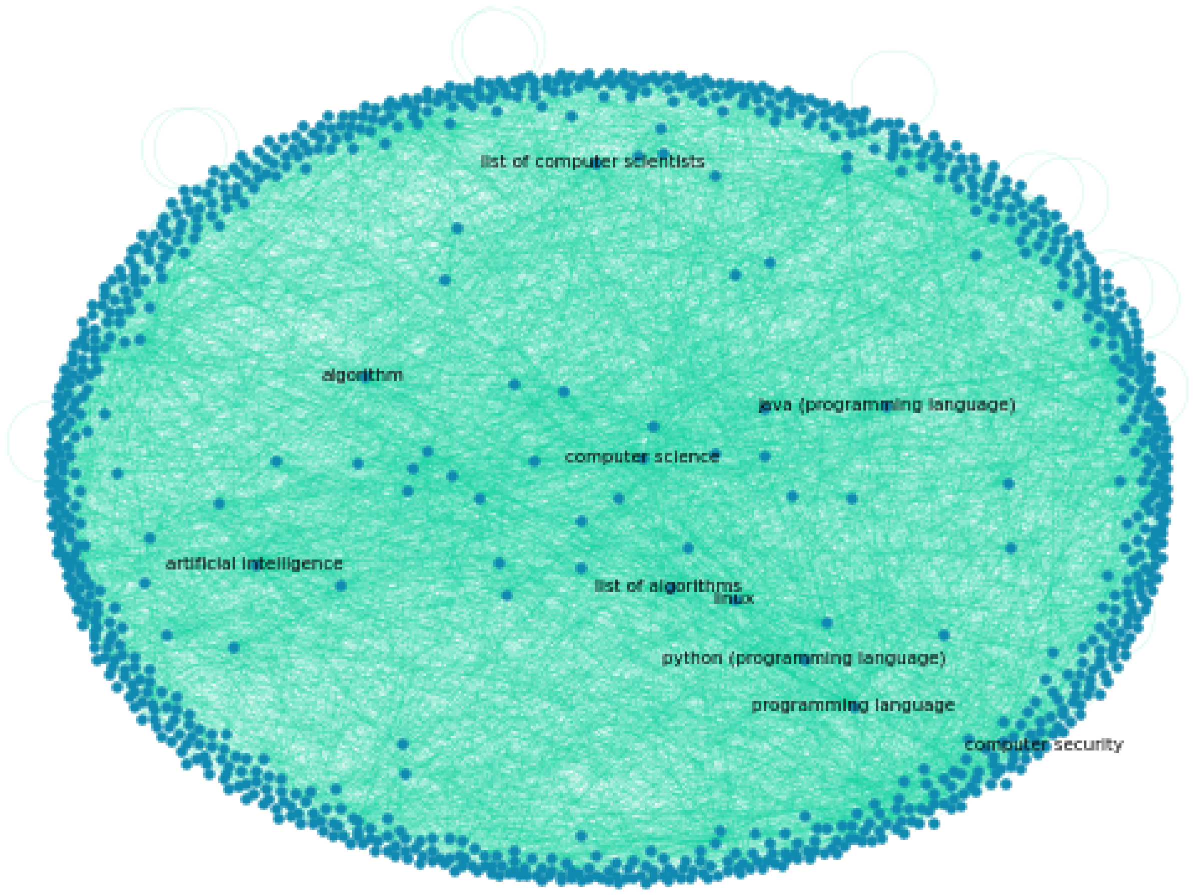


Figure 26: Connectivity of 1000 Articles

7.3.4 First 3000 Articles

Graph has 2922 nodes and 13563 edges.

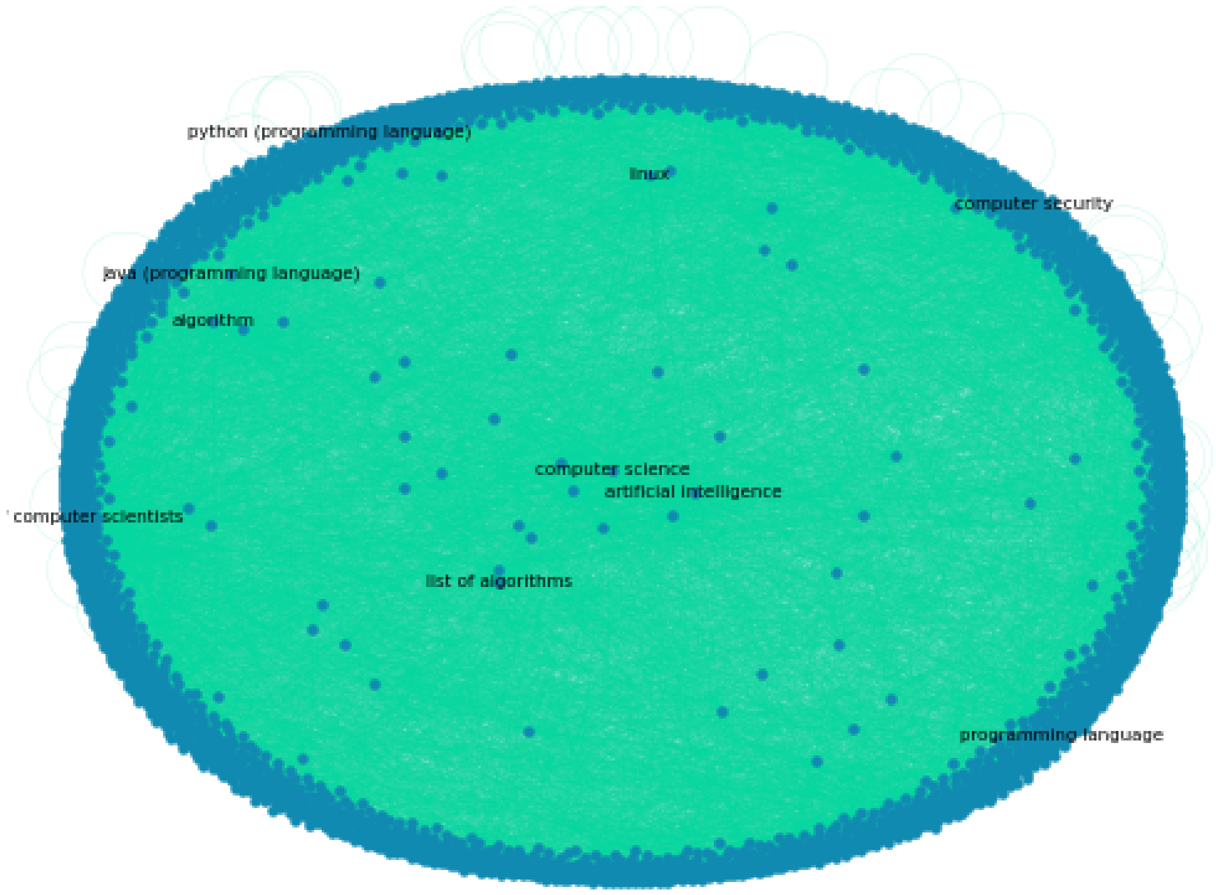


Figure 27: Connectivity of 3000 Articles

7.3.5 All Articles

Graph has 5008 nodes and 23262 edges.

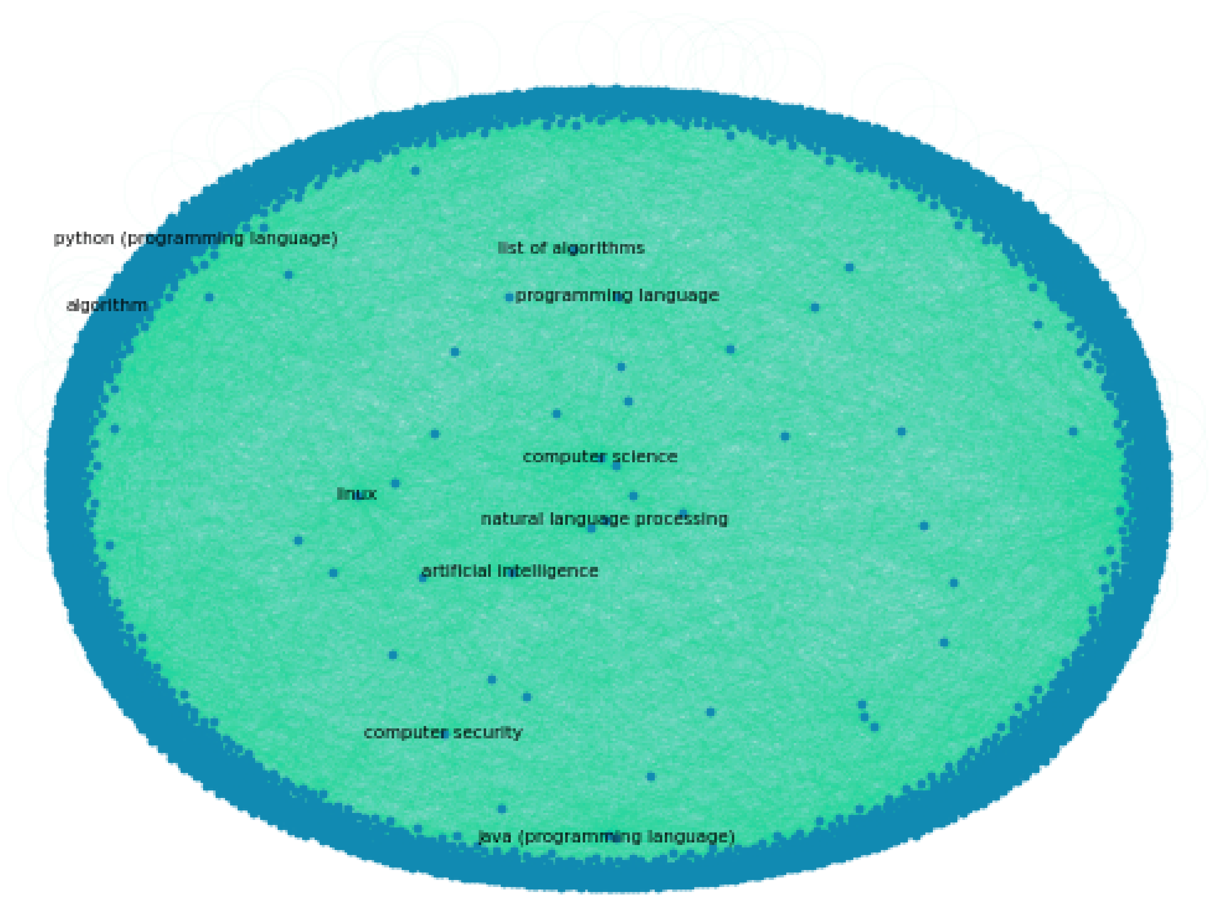


Figure 28: Connectivity of All Articles

Nodes with more than 150 edges are displayed below:

- Computer science - 527
- Artificial intelligence - 423
- Linux - 399
- Algorithm - 360
- Computer security - 293
- Java - 279
- Programming language - 264

- Natural language processing - 242
- Python - 232
- List of Algorithms - 232
- Google - 212
- Cryptography - 201
- Computer vision - 185
- Malware - 182
- List of computer scientists - 173
- Database - 169
- Encryption - 169
- Compiler - 164
- Information security - 156
- Information theory - 151

7.3.6 Discussion

This approach does not take the contents of articles into account, it only looks at the way the articles reference each other. The node with the most edges directed towards it is the most referenced node, and will represent the 'core' of the graph structure. It should come as no surprise that the centre of all of the plots, regardless of the number of articles used in constructing it, yields **computer science** as the centre point.

A less obvious fact is that Java and Python are amongst the most referenced articles in Computer Science. These are arguably the two most common programming languages and they are versatile enough to be used for almost everything, hence why they are referenced so often.

Now to compare this method of generating a graph structure of Computer Science and using the contents of the articles. I believe they are somewhat different in their purpose. This hyperlink approach presents the most referenced articles in the field, but most referenced does not imply the most significant. The content analysis approach takes the contents of the articles into account, which means the relationship between articles are constructed based on the information they entail. Of course, the content analysis approach considers hyperlinks to other articles to an extent, as the title of hyperlinked articles will appear in the text too. However, the underlying principle of the content analysis approach is that it places more significant on the semantic meaning of text. Furthermore, the hyperlink approach does not consider the weight of connections. For example, the term 'natural language processing' could appear in the article 'speech recognition' many times, but NetworkX will only capture the extent to which the term is encapsulated in the hyperlink format.

In the end, I went with the Doc2Vec approach as I believe taking the contents of articles into account makes the final model more fitting.

7.4 Colouring Categories

Now that we have our list of categories, we can start to colour them. Two approaches were tested. For both of these, the first step is to identify the categories, as described in previous sections. A unique colour is then assigned to each category, which will mark the boundaries of the category on the final graph structure.

The first method of colour assignment simply involves assigning random colours to each category - there will be no correlation in colour between similar/neighbouring categories, which makes the clusters quite visible. The fact that there is no correlation between neighbouring elements might seem counter-intuitive. However, I believe it fits with a key characteristic of t-SNE which is that the global structure is not preserved, indicating neighbouring clusters may not be near each other in meaning.

The second method will see that the centroids of each category are retrieved, and a colourmap involving these centroids is plotted according to a colourmap. Then, the colour assigned to each category by the colourmap is duplicated to all other articles in that particular category.
This method as opposed to the first one makes more sense when neighbouring categories are also similar in meaning. The second method assumes that the more similar two categories are in colour, the more similar they are in meaning too. This is due to the nature of colourmaps - it is a continous spectrum so colour implies adjacency in the metric space.

7.4.1 Testing Different Colourmaps

The Seaborn library seaborn makes it easy to visualise graph structure in terms of colours. It provides a large set of colourmaps to choose from, which gives the user a lot of choices but is also overwhelming to an extent.

The general principle behind colours is the idea of the three components, red, green, and blue. In the digital world, colours are usually represented by their RGB values, which each define red, green, and blue the colour is, respectively. This is what we will use to colour the points on our final plot.

However, for analysing the attributes of a colour, it is better to think in terms of hue, saturation, and luminescence. Hue is the component that distinguishes different colours, saturation is the colourfulness, and luminescence corresponds to how bright (i.e. light or dark) it is. Hue is useful for representing categories, humans see the different in hue quite easily (even when the other two components are kept constant). Luminescence on the other hand is often used to represent the differing values in quantity. For example, for confusion matrices, the same colour scheme is often used but a darker colour will represent a higher value.

So, in our case, we focus on the hue component. A basic approach to finding unique hues to represent categories is to draw evenly-spaced colours in a circular colour space. A circular colour space is one where the hue changes while keeping the brightness and saturation the same. The hls colour space is a simple transformation of RGB values, and is often used.

Seaborn includes six common matplotlib palettes. I tried out *colorblind*.

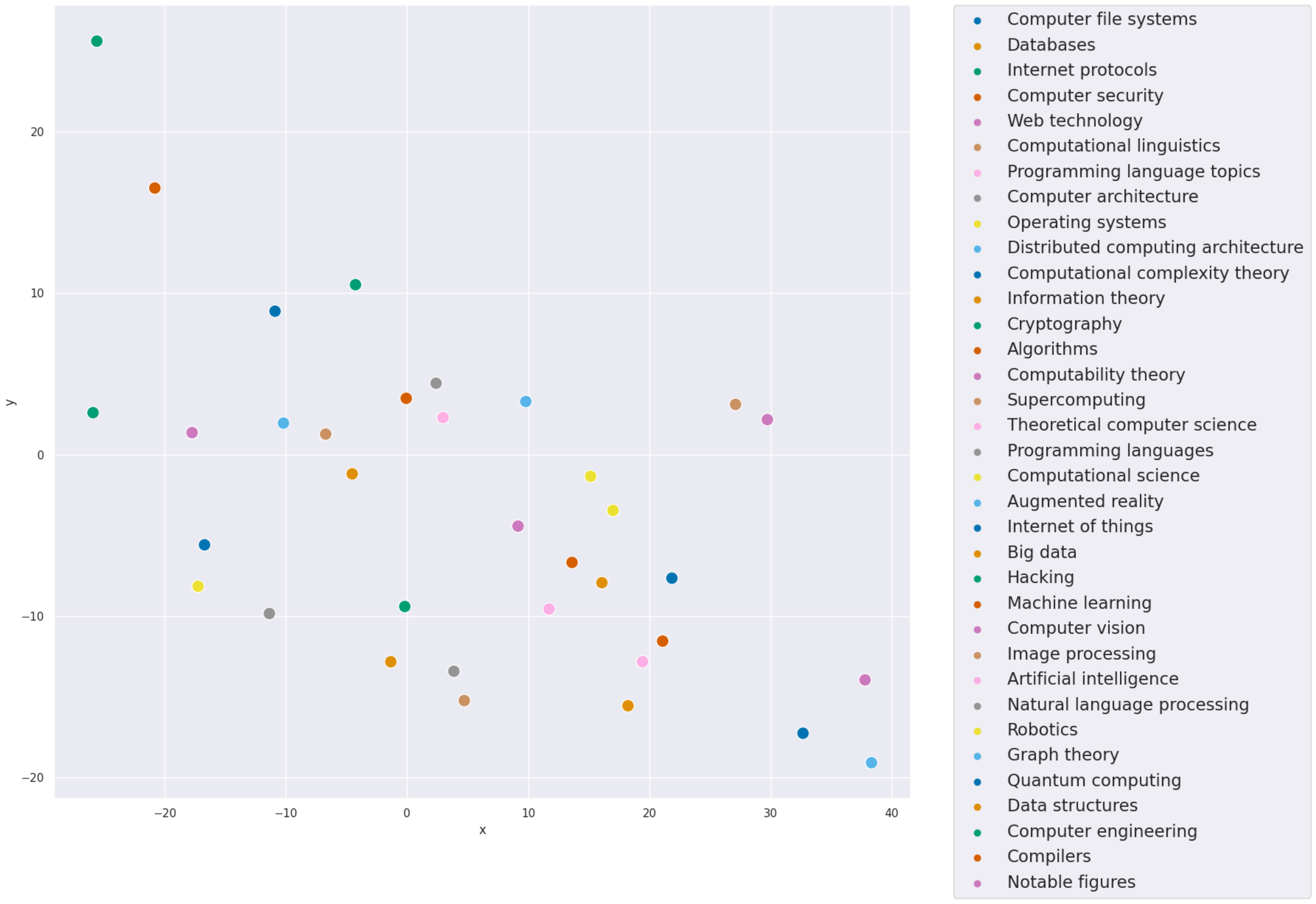


Figure 29: Colorblind Colourmap

Qualitative palettes are well-suited to representing categorical data because they are a pre-set group of distinct colours which make it easy to distinguish between the categories. There is no order which is what we want in this case as we have not assumed any relation between the categories.

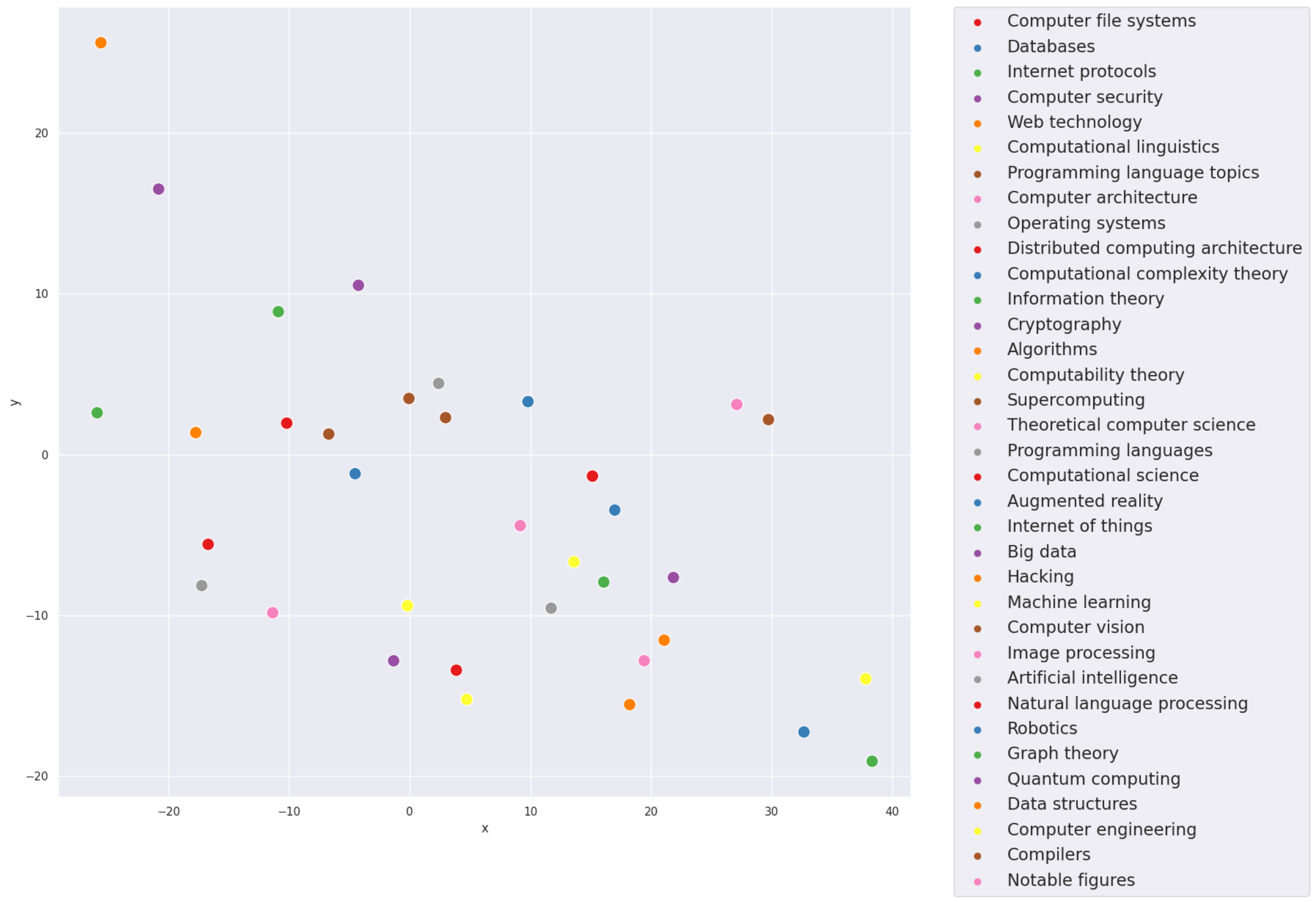


Figure 30: Set1 Colourmap

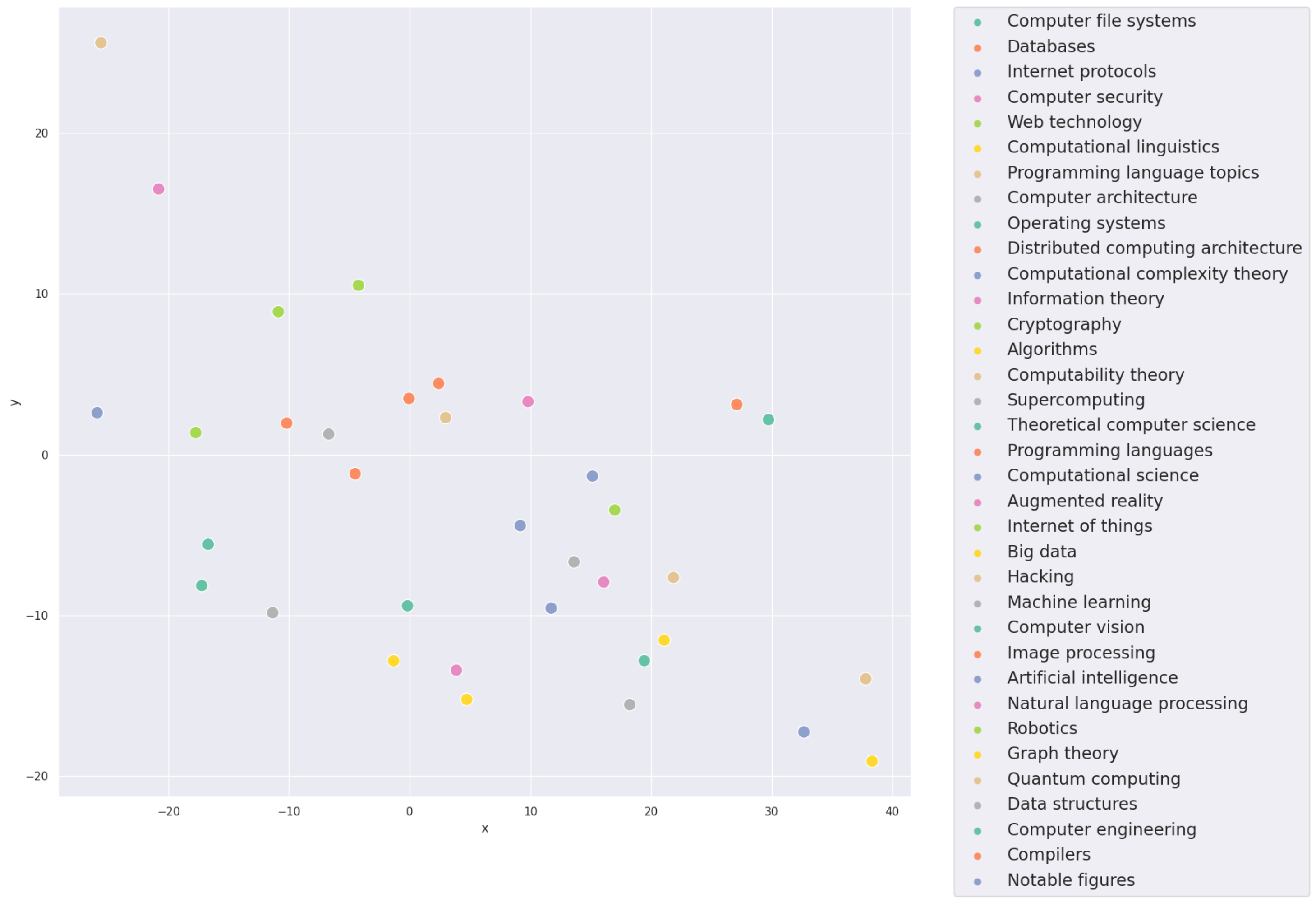


Figure 31: Set2 Colourmap

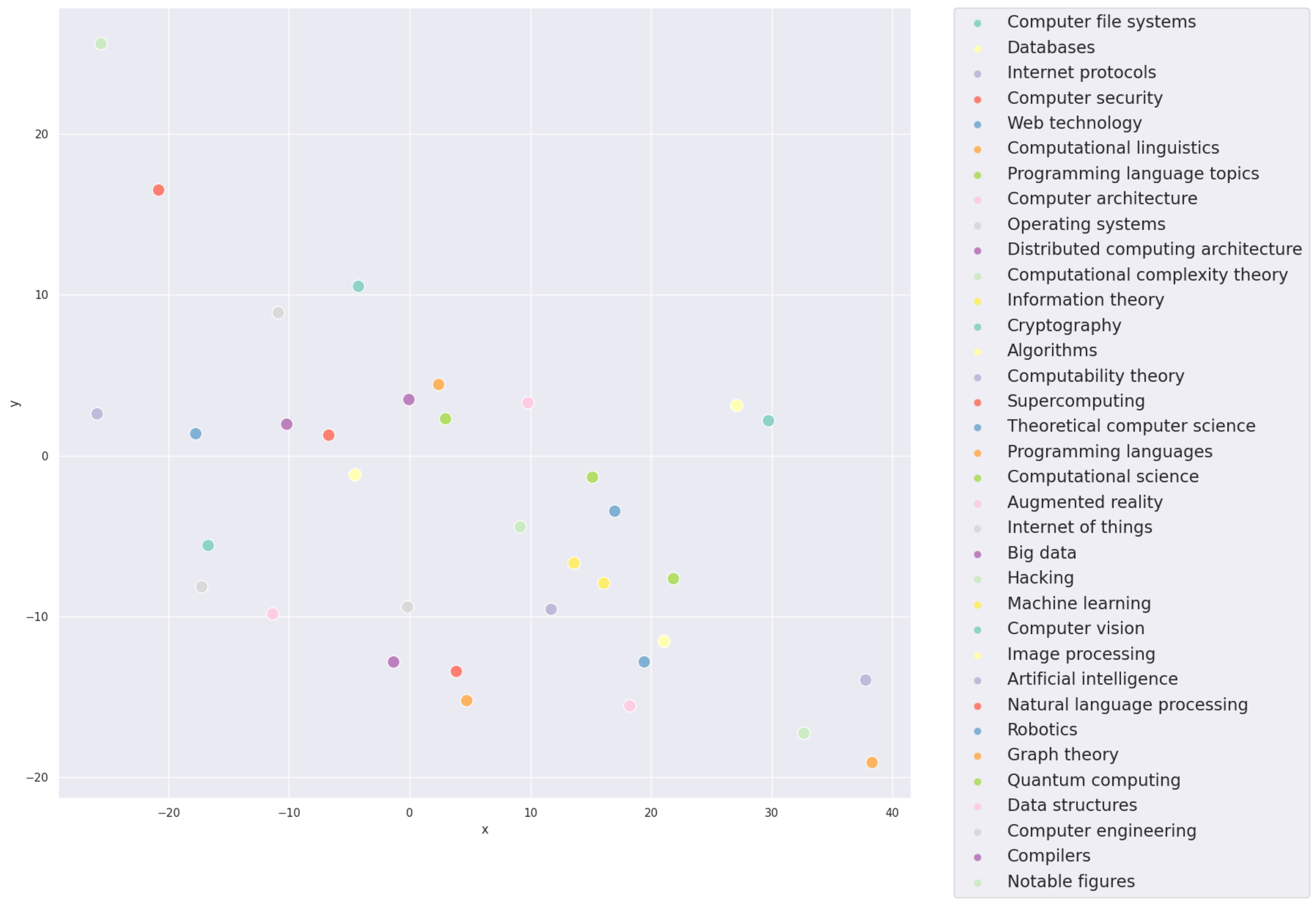


Figure 32: Set3 Colourmap

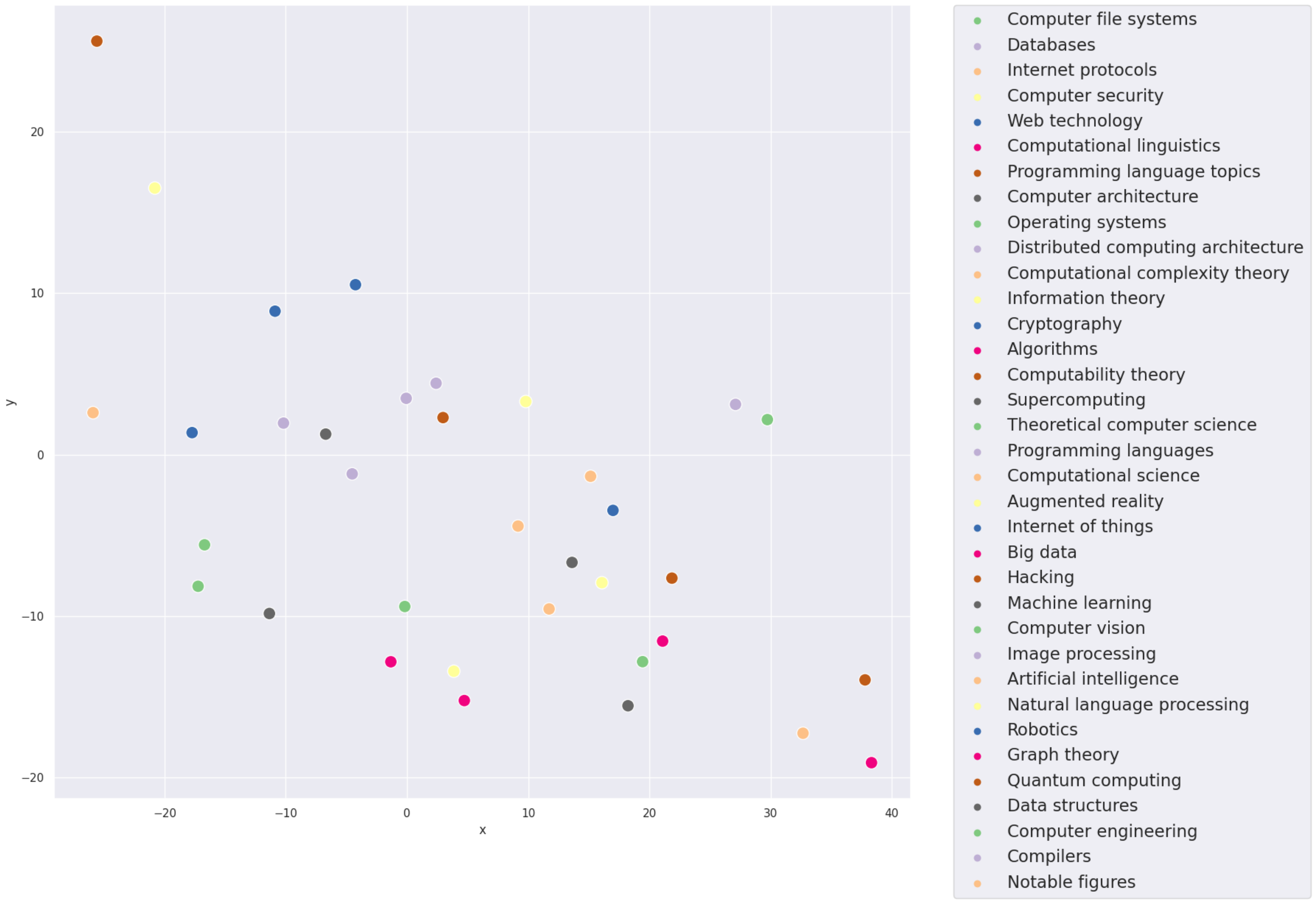


Figure 33: Accent Colourmap

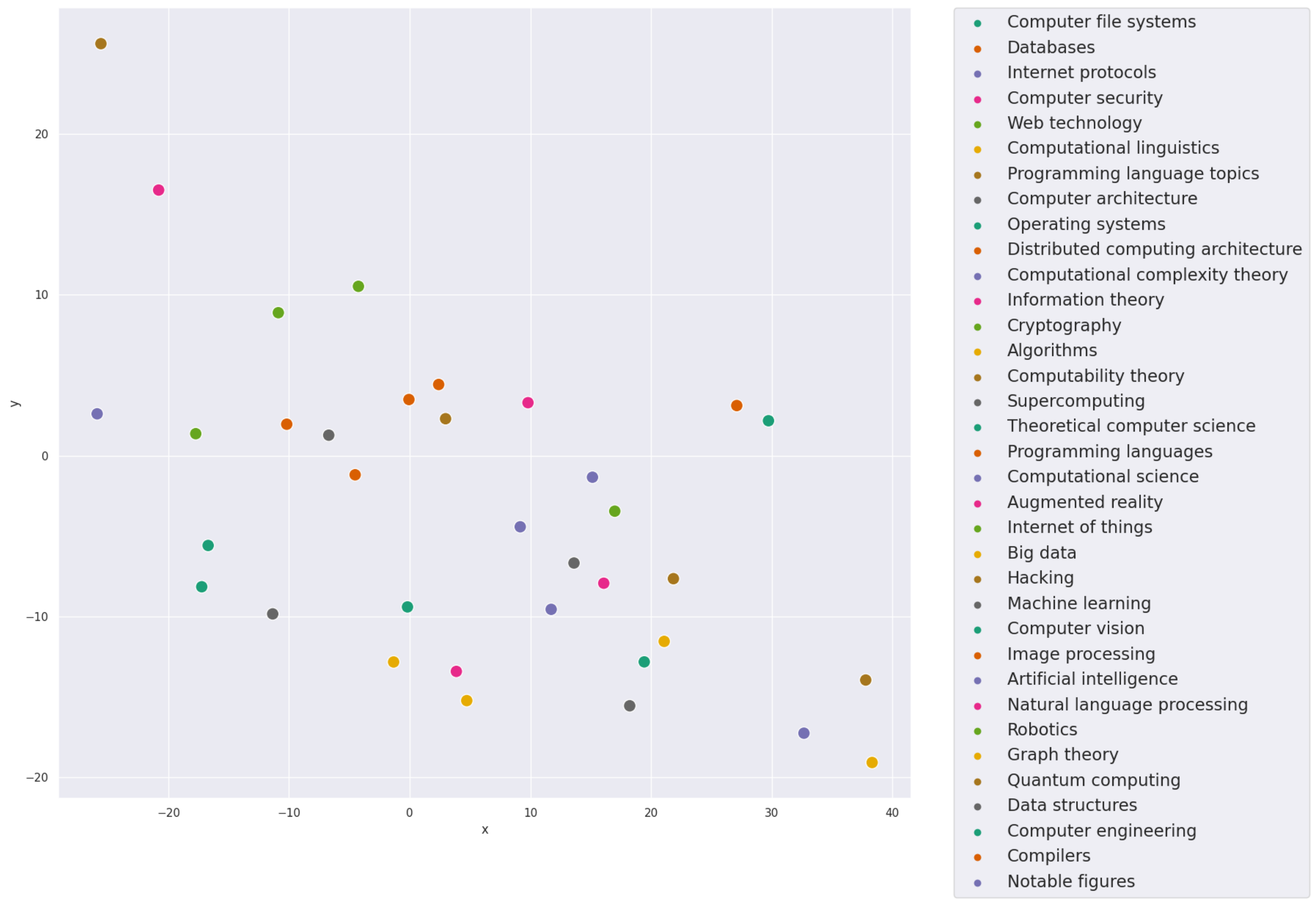


Figure 34: Dark2 Colourmap

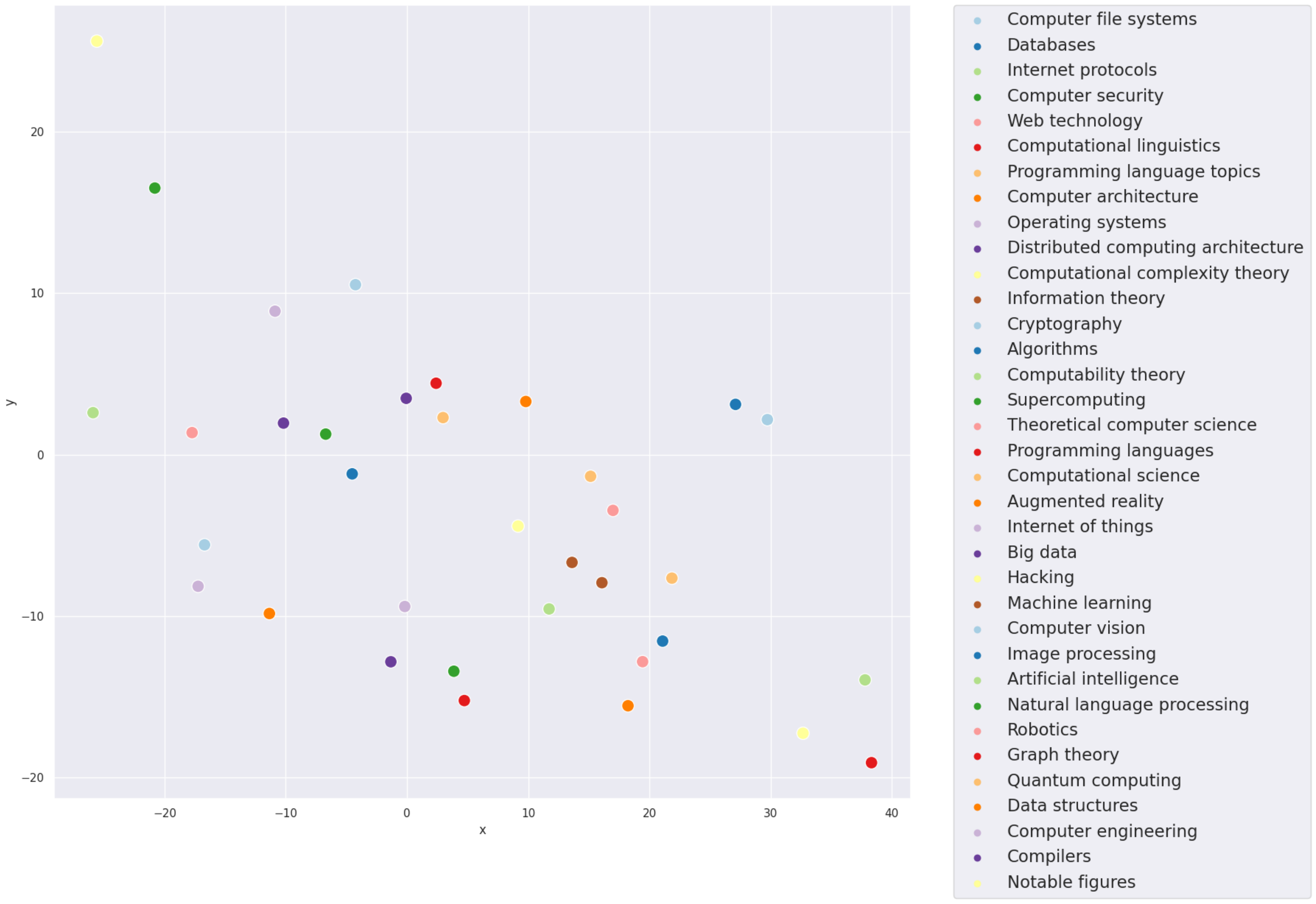


Figure 35: Paired Colourmap

I selected the set 2 colourmap in the end because the colours are really nice and it does not contain colours that are not visible (like yellow in set 3 and accent).

8 Frontend Showcase

Investigation into the traffic patterns of CS articles was also performed. The hypothesis is that readers will follow hyperlinks on an article they are reading to related articles. The Wikipedia Pageview API pageview provides a collection of REST endpoints that serve data about pageview stats in Wikipedia projects and articles. The requests are structured like '/metrics/pageviews/{endpoint}/{parameter 1}/{parameter 2}/.../{parameter N}'.

For each article, the hyperlinks are retrieved and filtered so that only links to another article within the structure are left. For example, the article 'Natural language processing' includes a hyperlink to the article ‘Linguistics’, but because linguistics is not considered an article in the field of Computer Science, it is filtered out. Next, the page view stats of the article are compared to each link it contains through a plot that displays on the side panel.

On the frontend, when a point is hovered on, the title is displayed. When it is clicked on, the side panel will appear along with basic information and traffic patterns with some related articles.

8.1 Features

The final graph structure looks like this on the frontend:

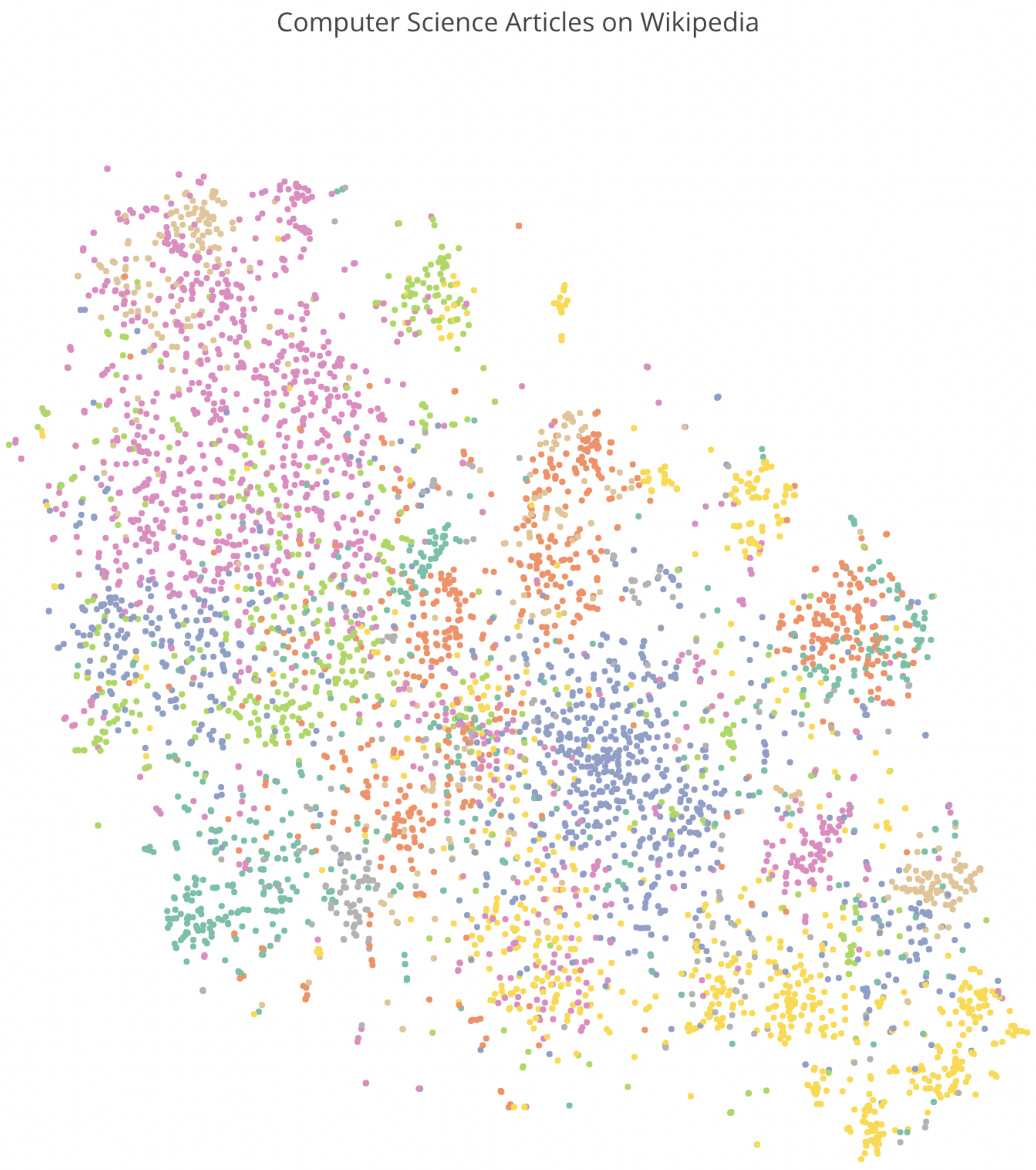


Figure 36: Graph Structure

The user can hover over a node and its title will be displayed:

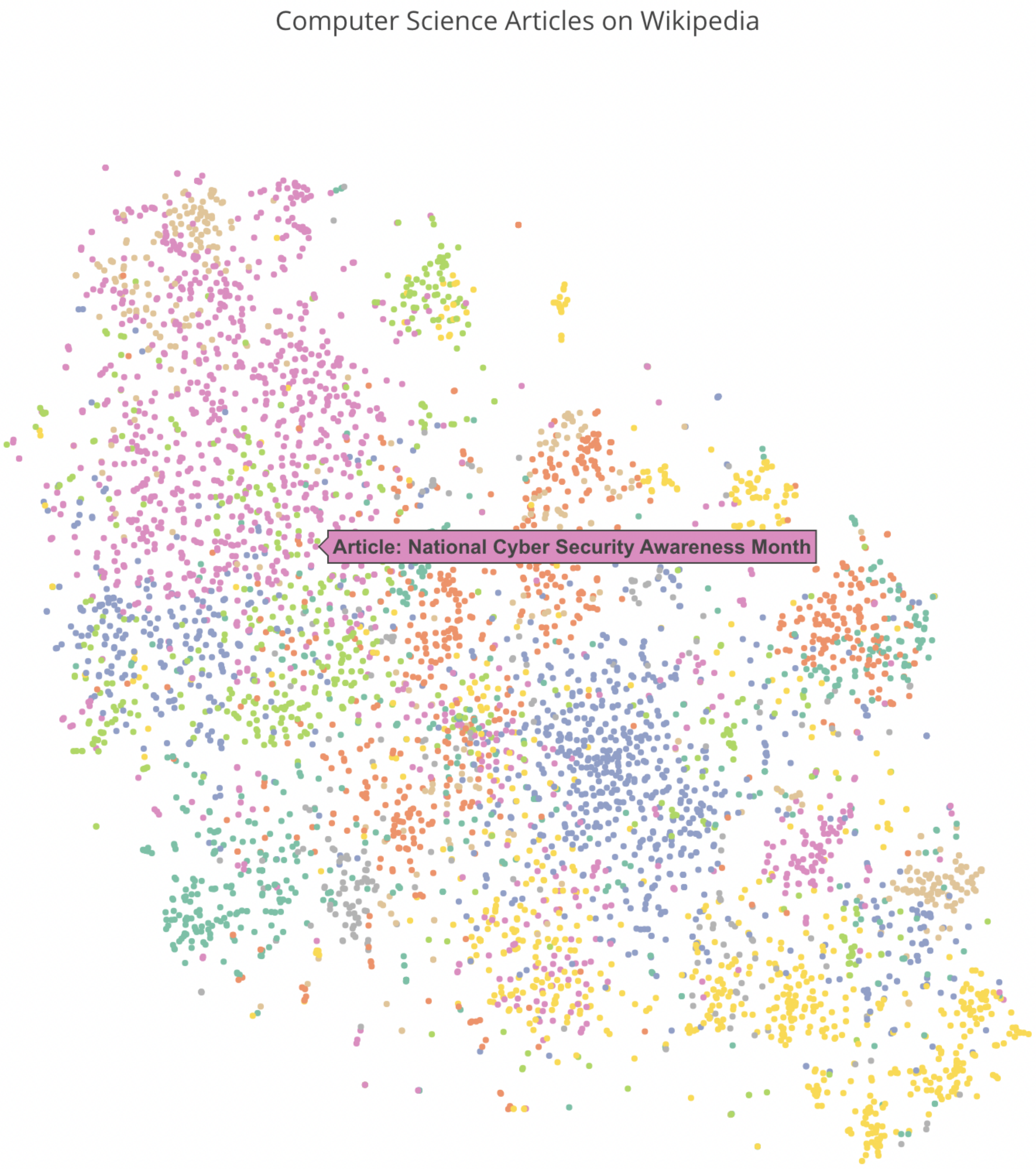


Figure 37: Hover over Node

When the user clicks on the node, the clicked node will turn a different colour (i.e. bright red) and a side panel will appear on the right hand side of the screen. Its title, category, and today's views are displayed. Its views over the last ten days and its hyperlinked article's views over the last ten days are also displayed. Note that the hyperlinked articles must be contained in the graph structure to be displayed:

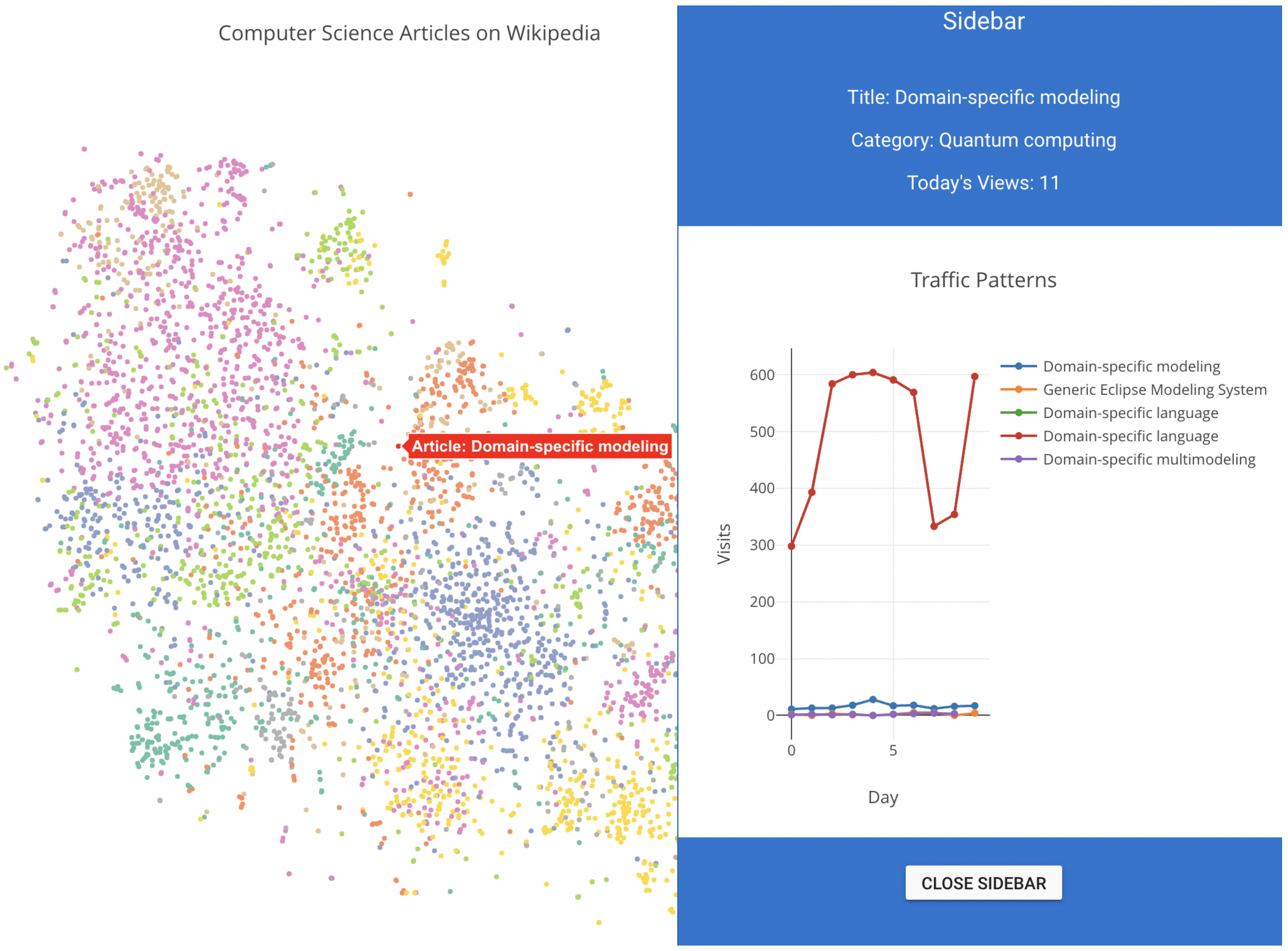

Computer Science Articles on Wikipedia
Article: Domain-specific modeling
Sidebar
Title: Domain-specific modeling
Category: Quantum computing
Today's Views: 11
Traffic Patterns
Visits
Day
600
500
400
300
200
100
0
0
5
Domain-specific modeling
Generic Eclipse Modeling System
Domain-specific language
Domain-specific language
Domain-specific multimodeling
CLOSE SIDEBAR

Figure 38: Side Panel

The default date range is set as ten days ago till today. The user can change this using the calendar date range picker:

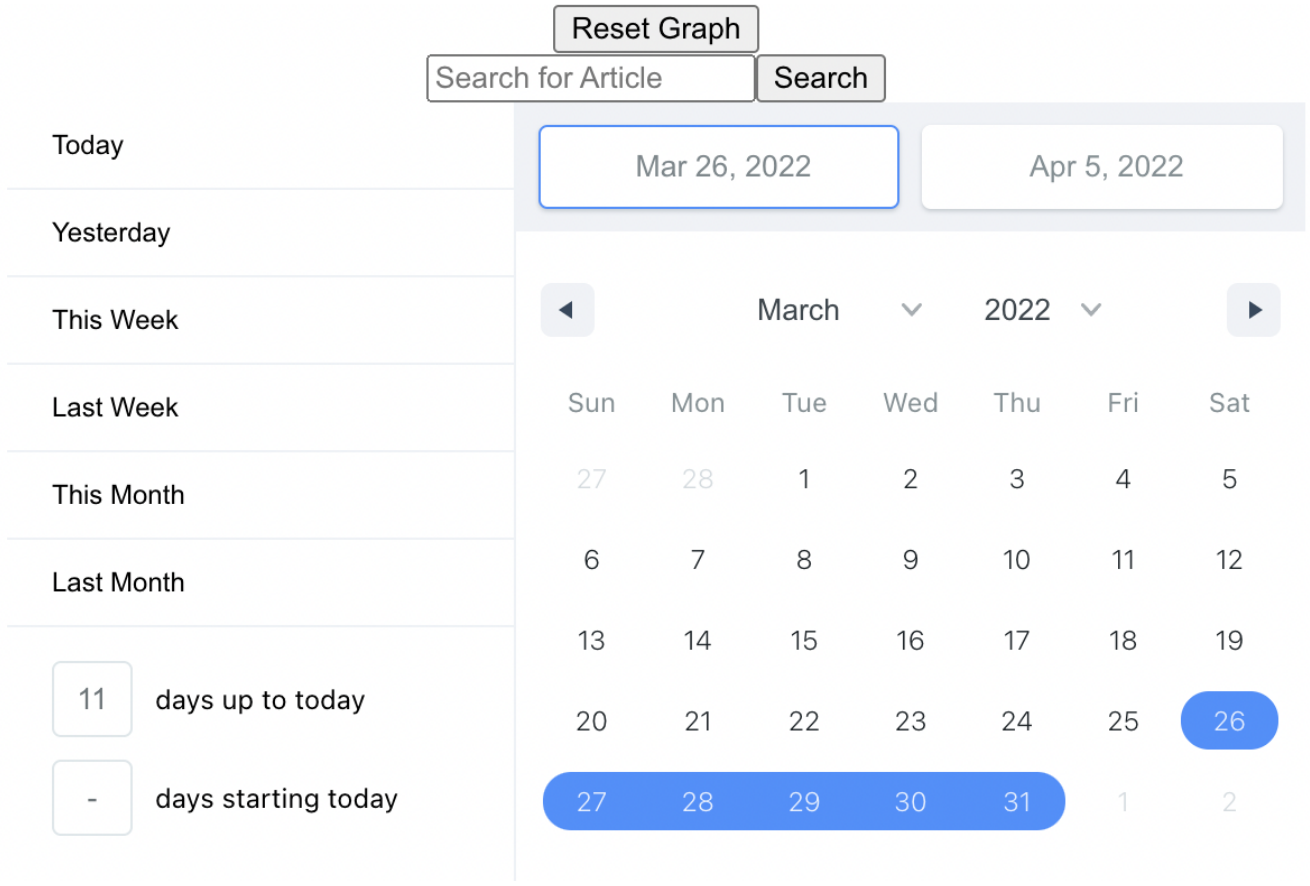


Figure 39: Date Picker

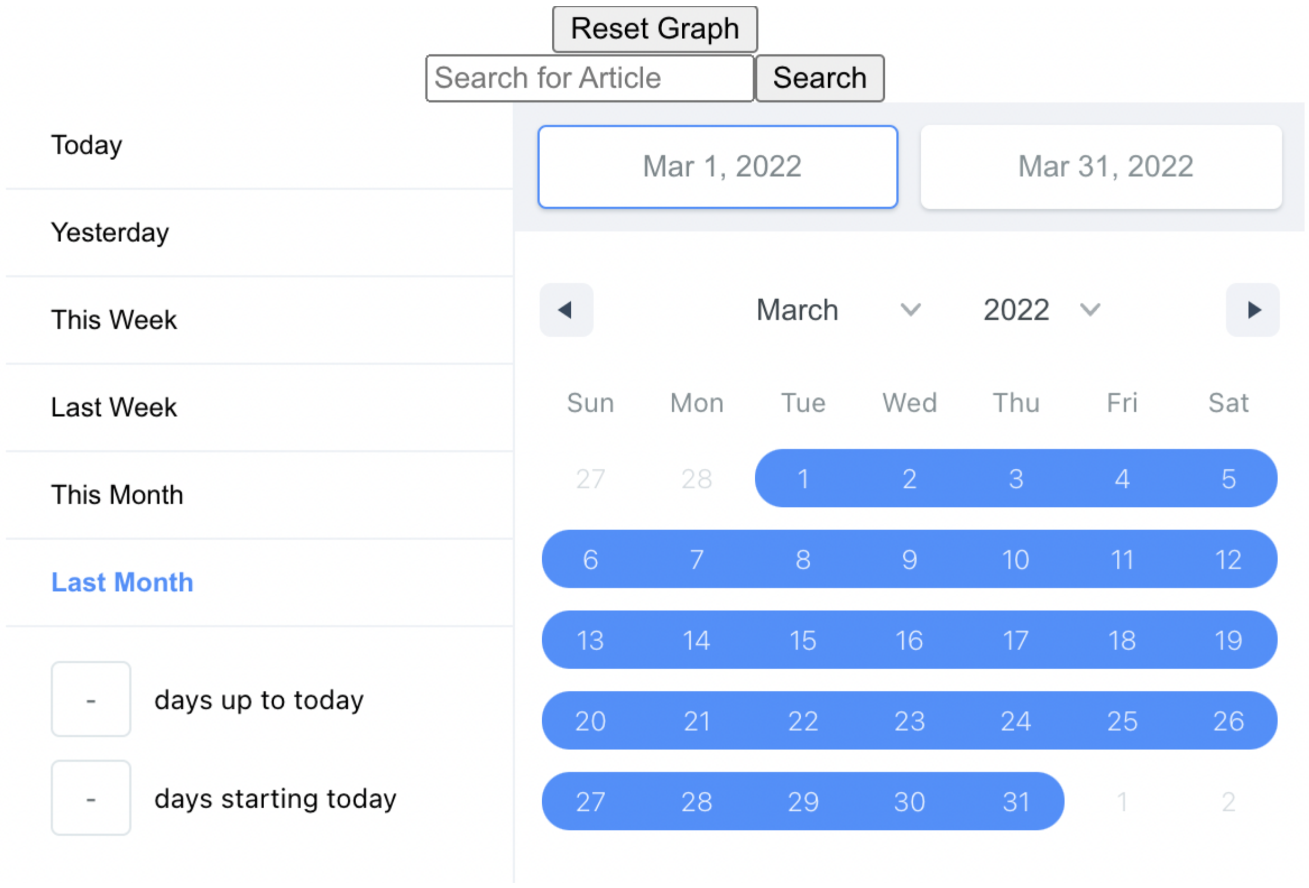
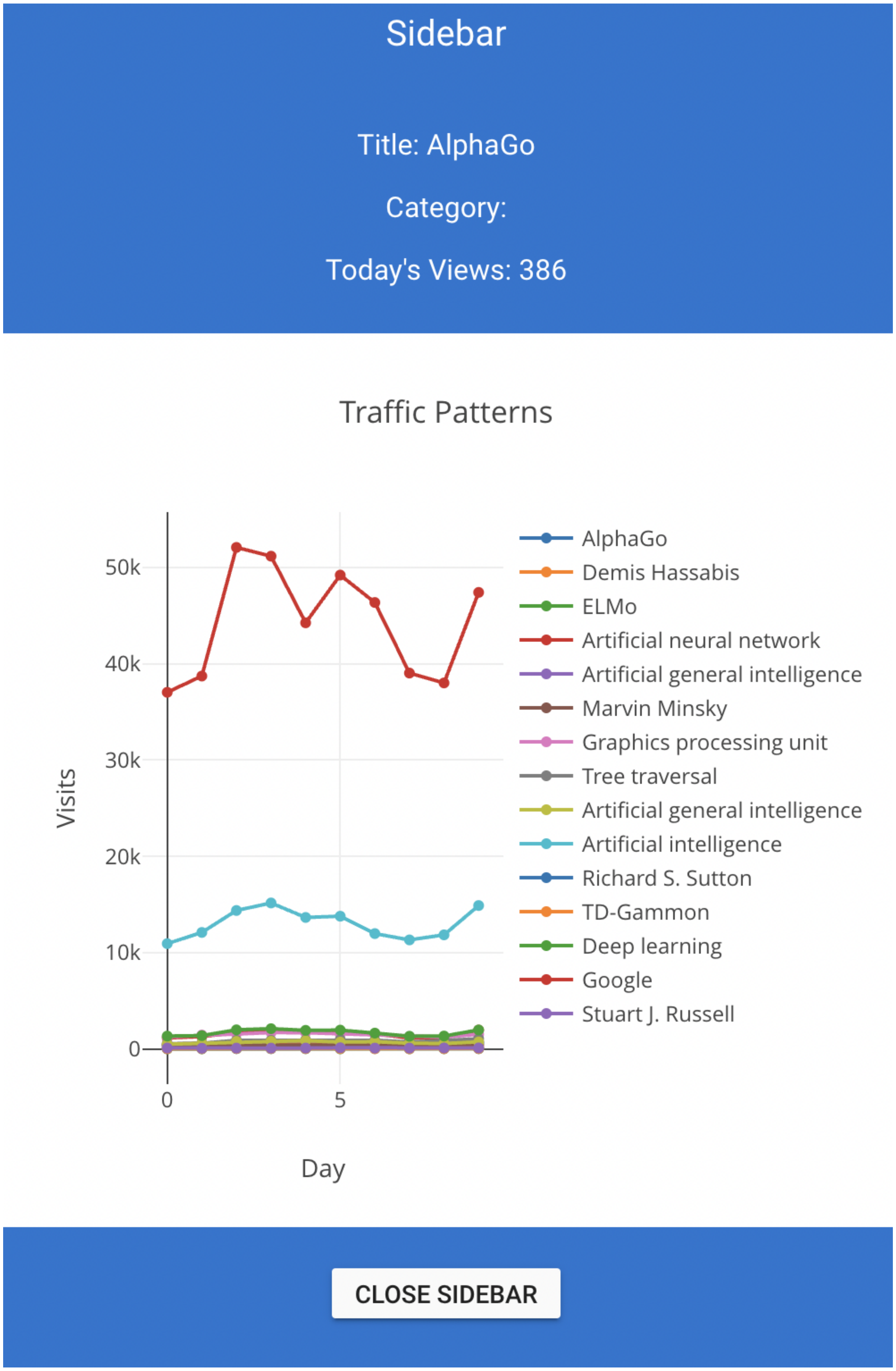


Figure 40: Select Dates

The user can also type the title of an article that is not in the graph structure into the search bar, and a new point will appear in the graph structure representing that article.


Sidebar

Title: AlphaGo

Category:

Today's Views: 386

Figure 41: Add Node to Graph

8.2 Interesting Patterns

In this section, we will look at some case studies of interesting traffic patterns. Our hypothesis is that there is a chance that readers will hop over to a hyperlinked article when reading a Wikipedia article, so the traffic patterns should match up to a certain extent.

8.2.1 AlphaGo

Date range: 2022/03/31 - 2022/04/06

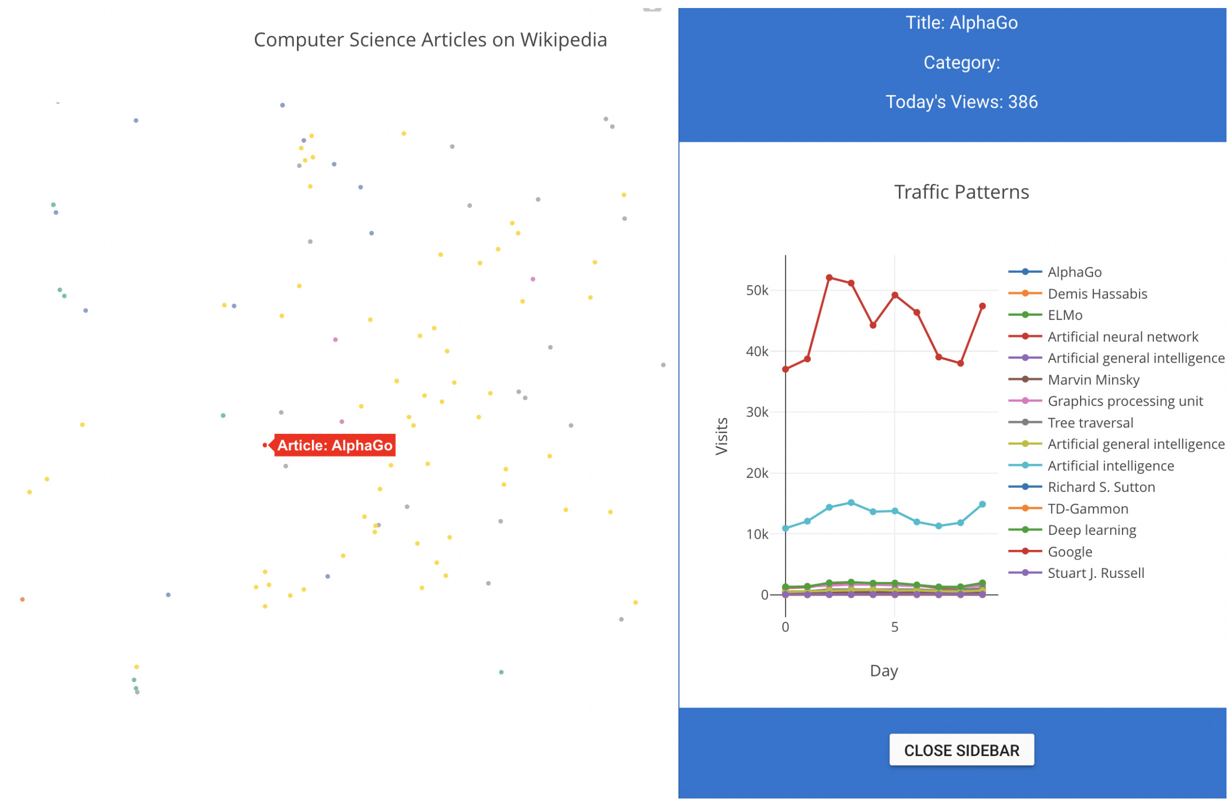
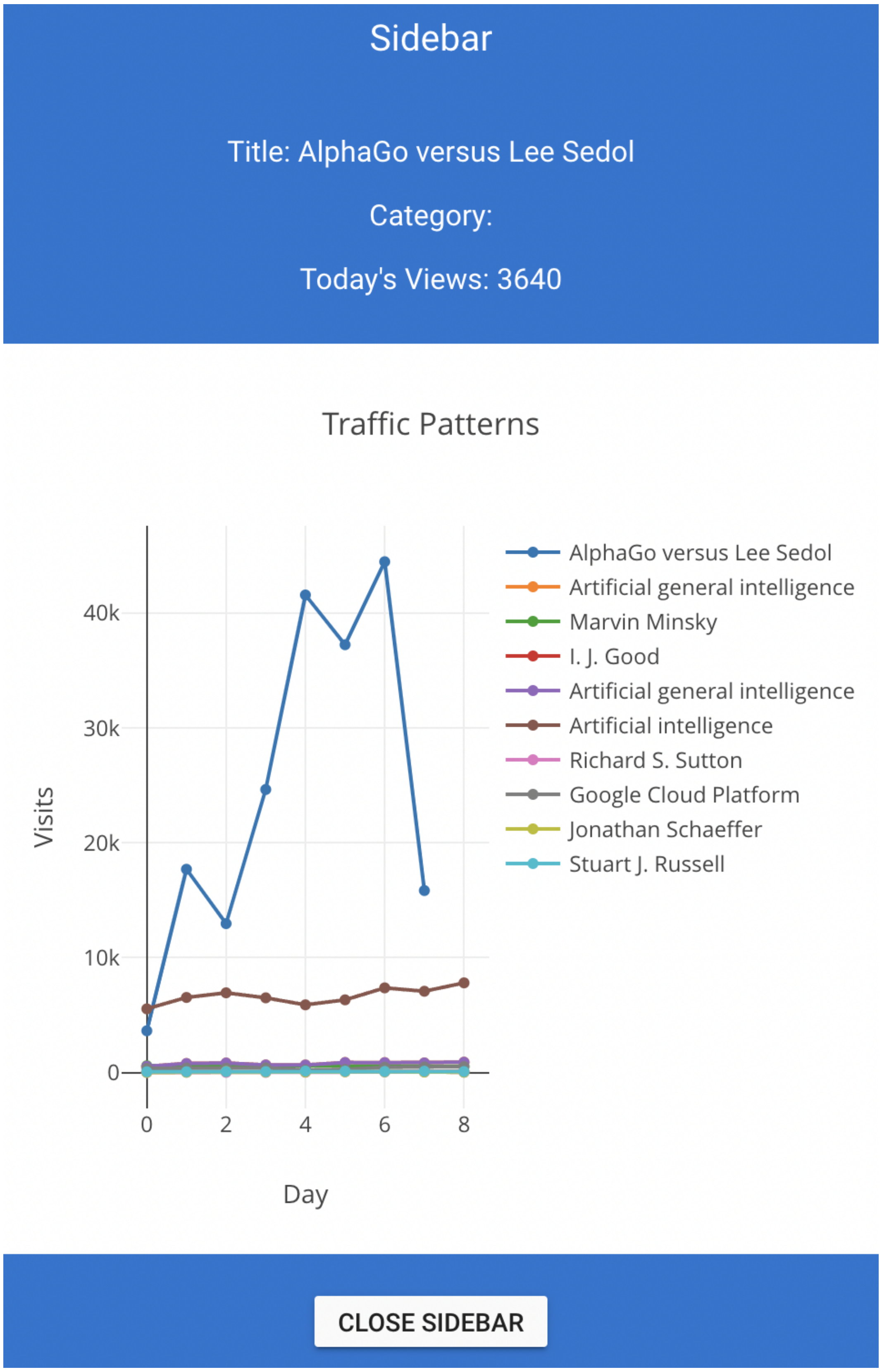


Figure 42: AlphaGo

Date range: 2016/03/08 - 2016/03/16. The match between AlphaGo (a computer Go program) and Lee Sedol (a top Go player) happened between 9th and 15th March 2016.



Sidebar

Title: AlphaGo versus Lee Sedol

Category:

Today's Views: 3640

Figure 43: AlphaGo Side Panel

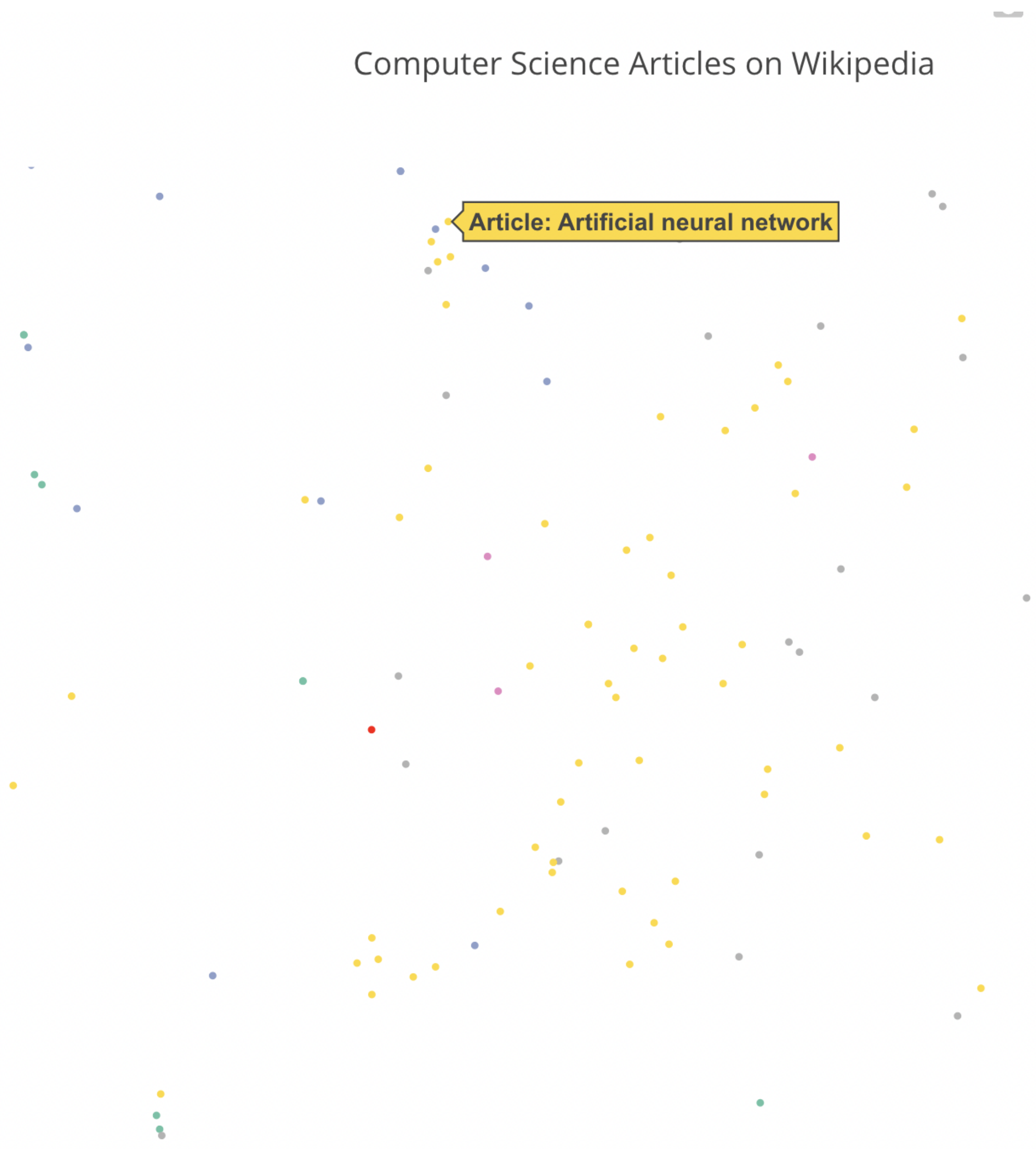


Figure 44: Neighbour of AlphaGo

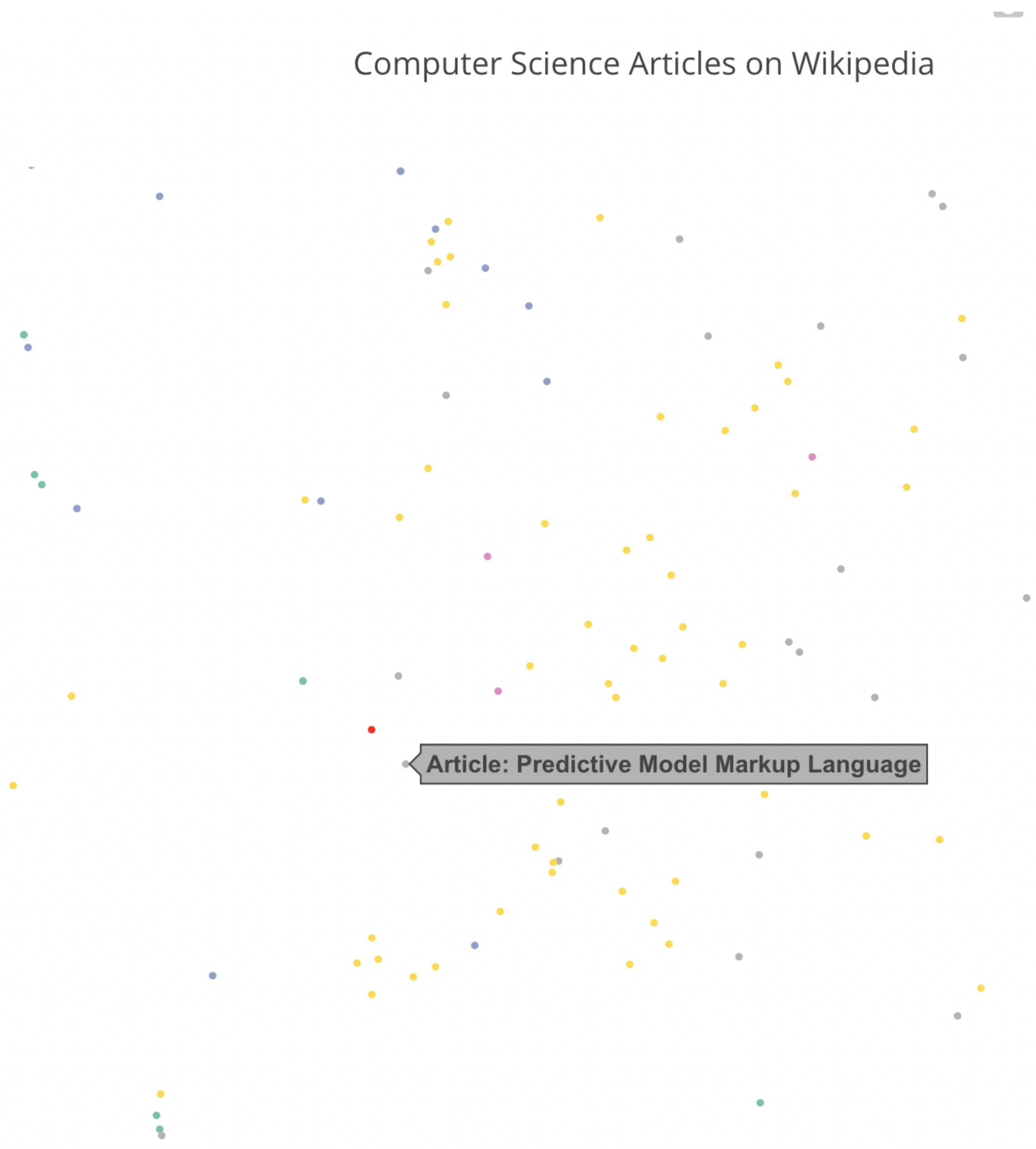


Figure 45: Neighbour of AlphaGo

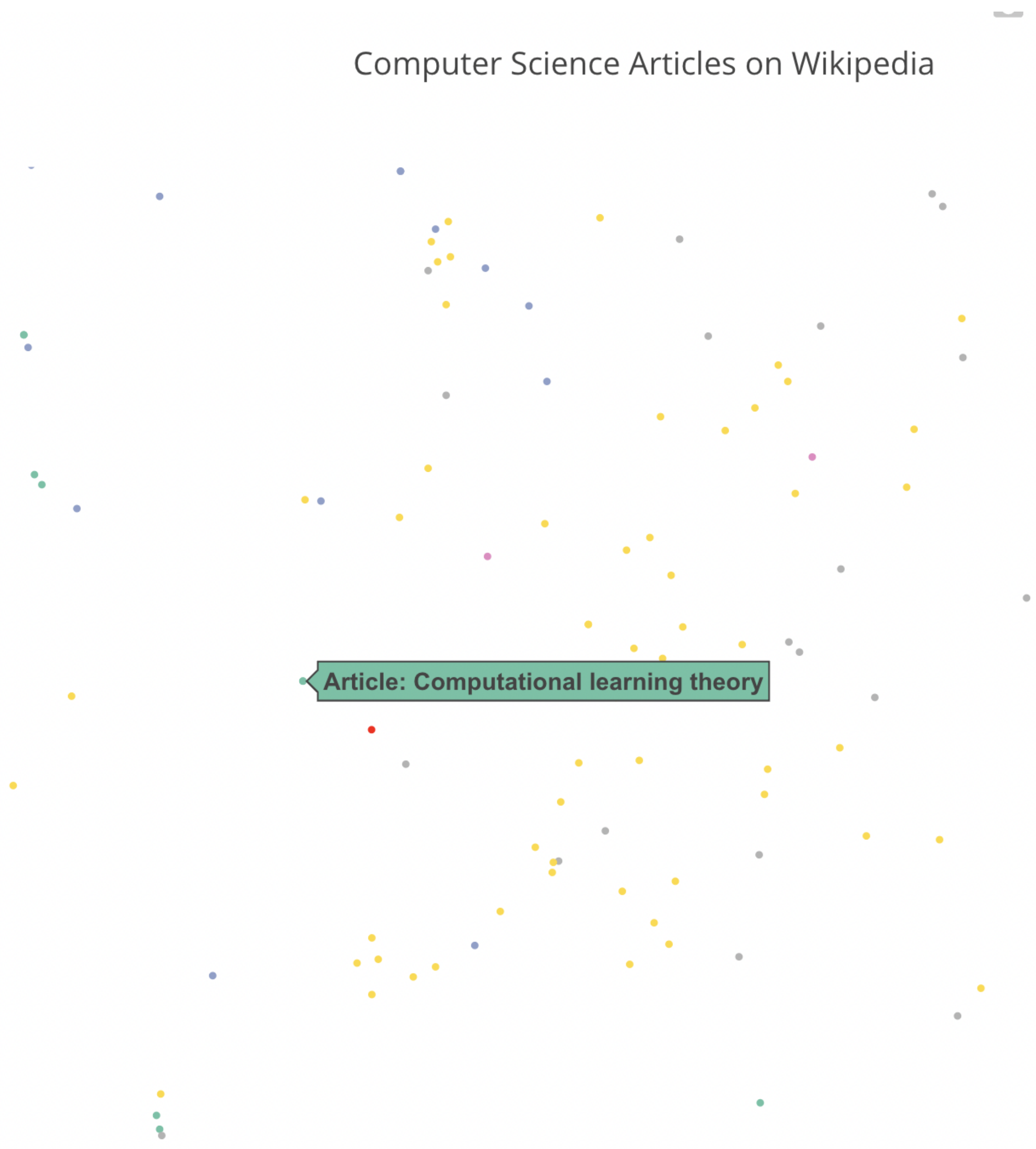


Figure 46: Neighbour of AlphaGo

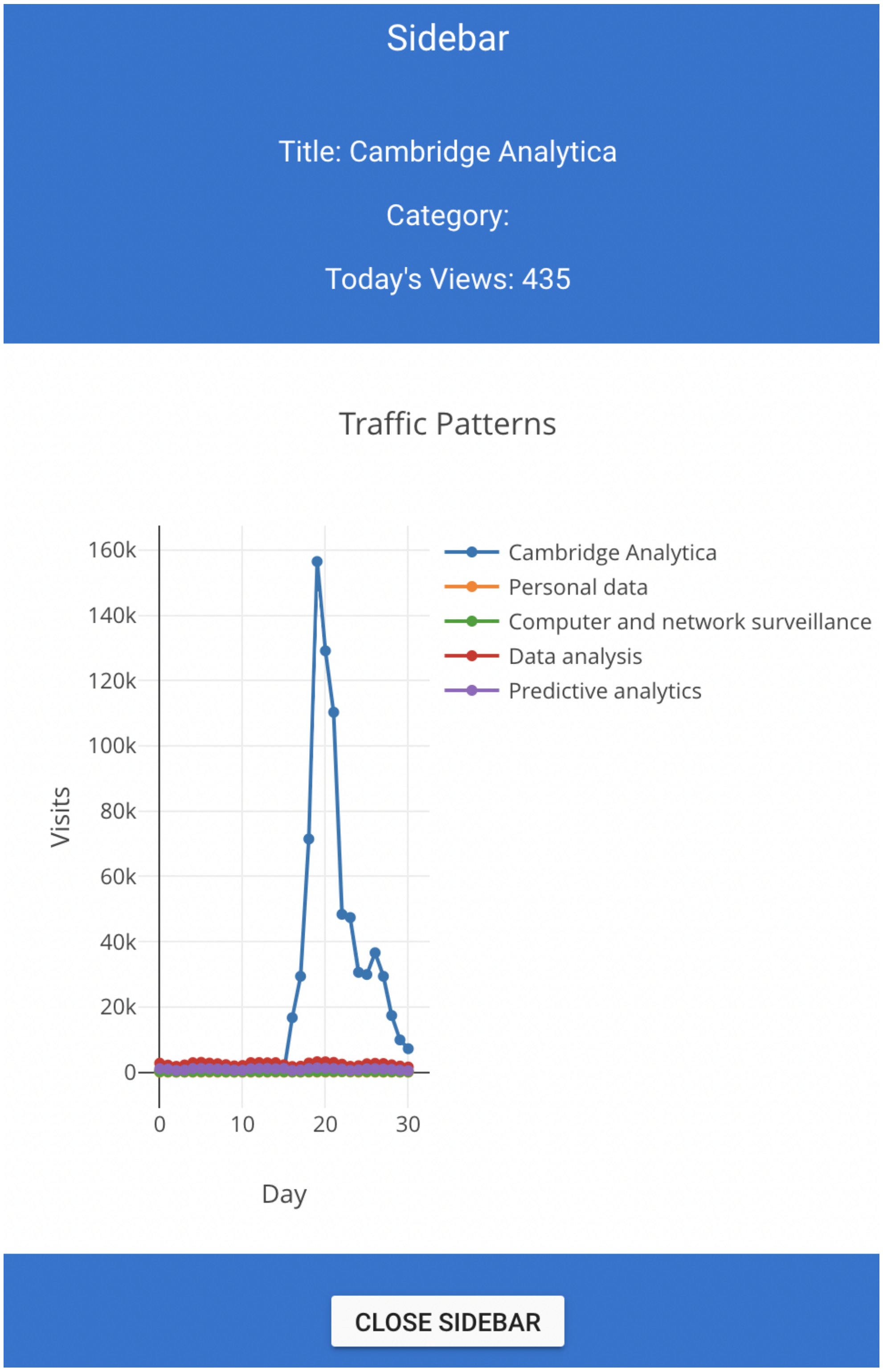
Computer Science Articles on Wikipedia
Article: Computational learning theory

Figure 47: Neighbour of AlphaGo

The first graph shows that the trends for hyperlinked articles of AlphaGo match up to the page view trend of AlphaGo, especially artificial neural network and artificial intelligence. This is as expected because these are the underlying ideas behind AlphaGo.

8.2.2 Cambridge Analytica

Date range: the month of June 2016.



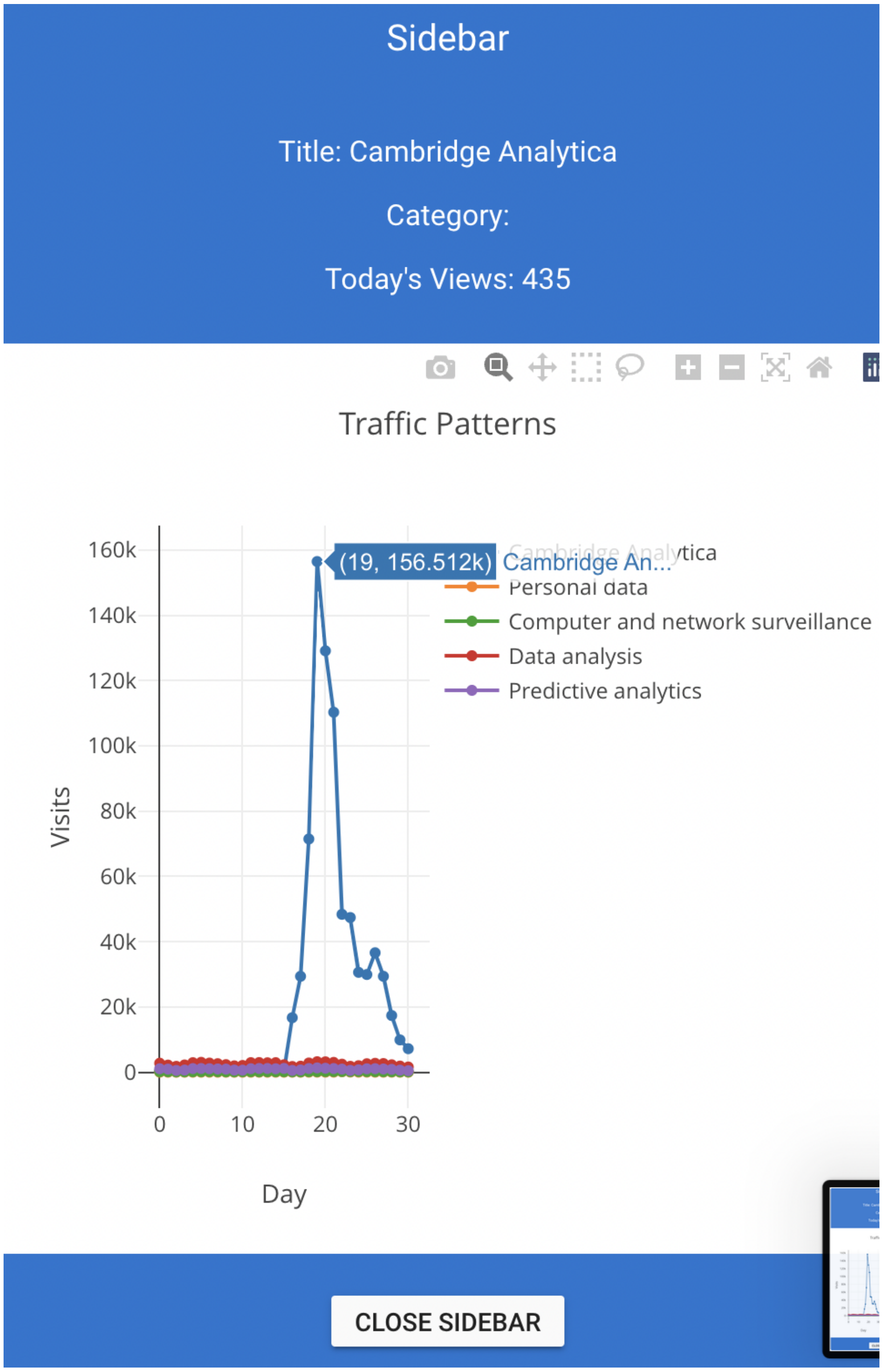
Sidebar

Title: Cambridge Analytica

Category:

Today's Views: 435

Figure 48: Cambridge Analytica



Sidebar

Title: Cambridge Analytica

Category:

Today's Views: 435

Figure 49: Cambridge Analytica Views

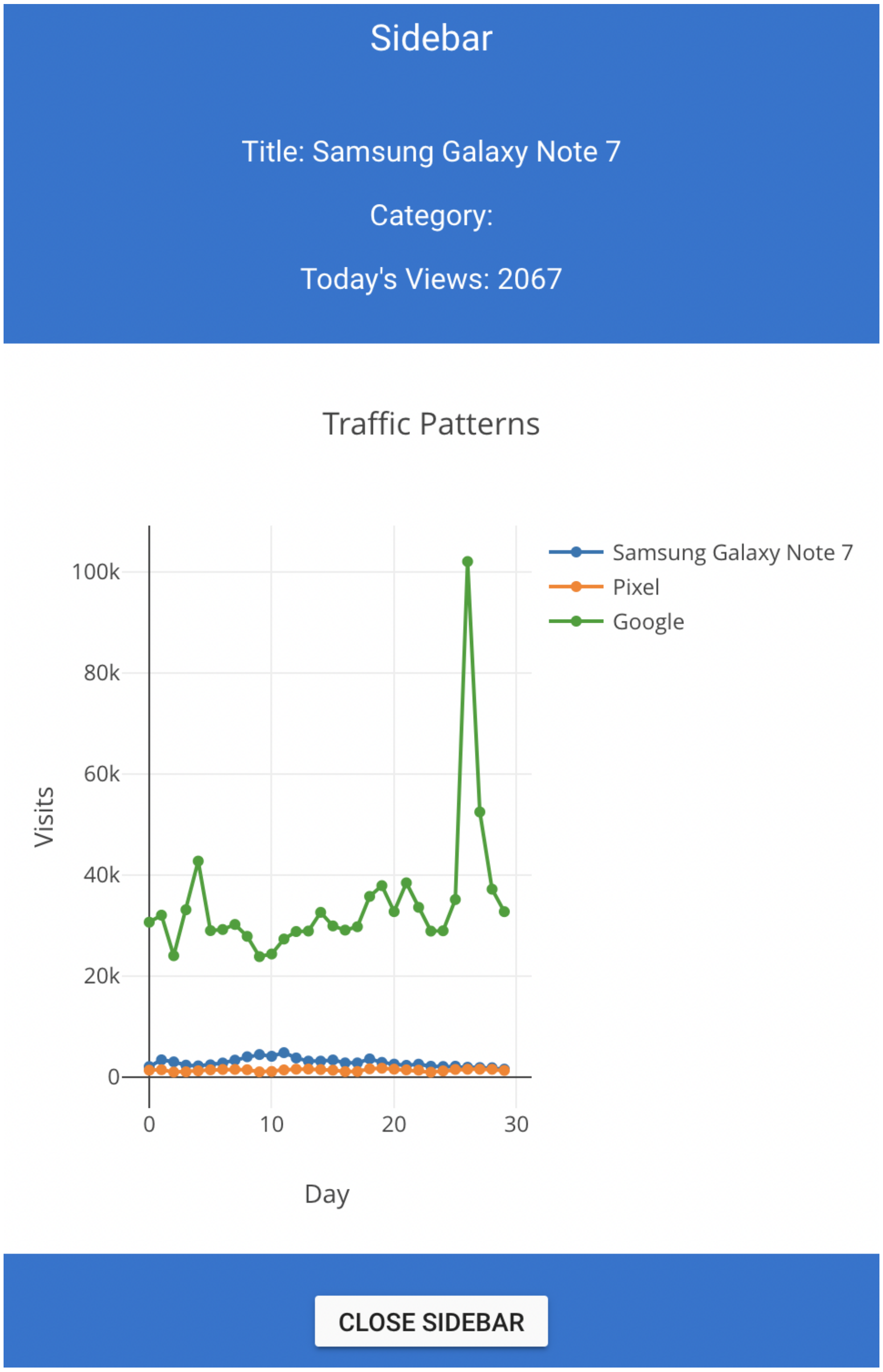
Article: Wi-Fi Protected Access

Figure 50: Cambridge Analytica

As the second figure shows, the Cambridge Analytica scandal reached a height of 156.5k views. However, the hyperlinked articles seem to not have followed the same trend. This might be because readers already gathered all they wanted to know from the Cambridge Analytica article page alone, as their motivation for reading it is not from a technical standpoint but rather a social one.

8.2.3 Samsung

Date range: the month of September 2016. Around the time when Samsung phones started exploding due to the batteries generating excessive heat.

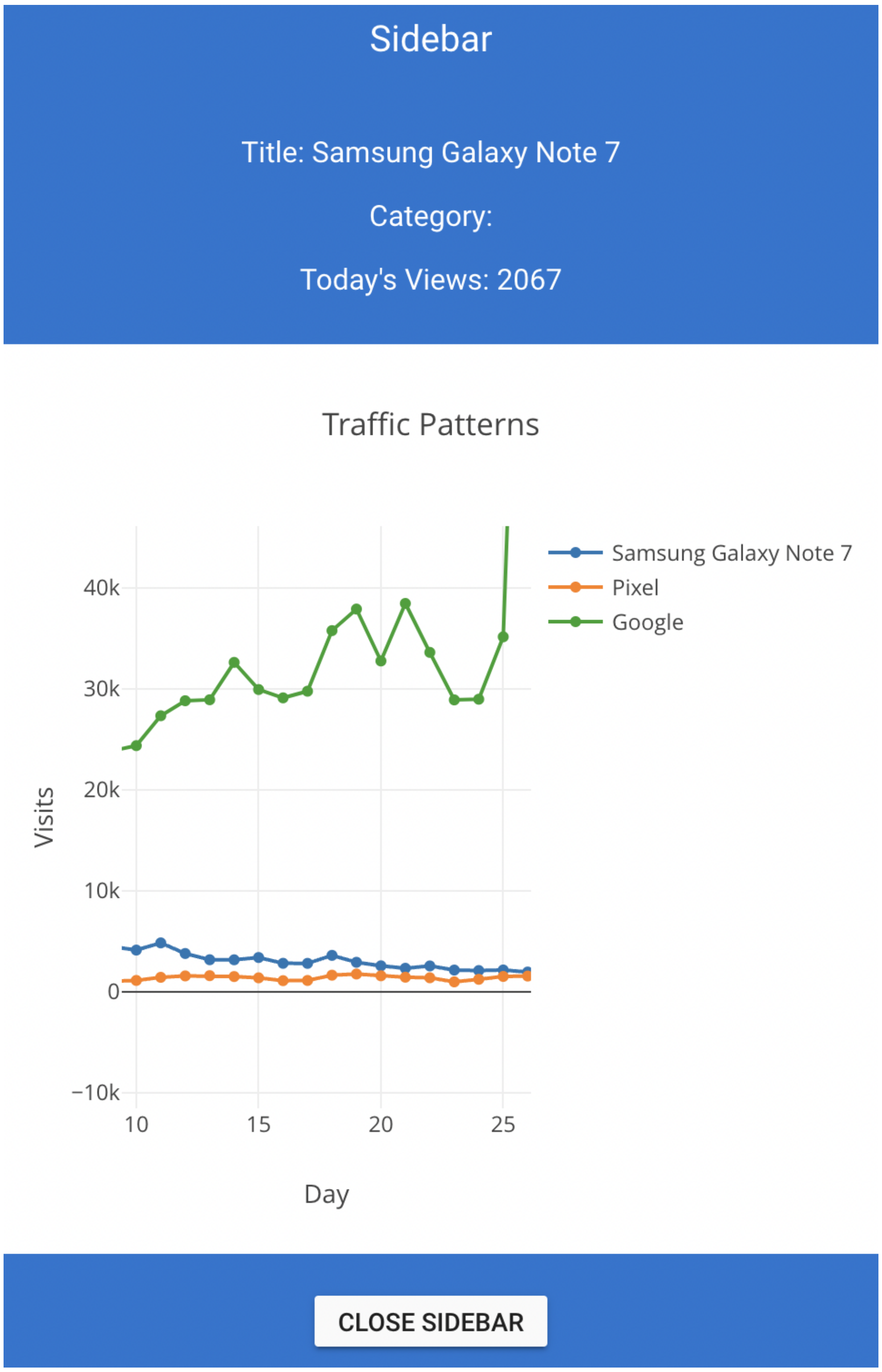

Sidebar

Title: Samsung Galaxy Note 7

Category:

Today's Views: 2067

Figure 51: Samsung



Sidebar

Title: Samsung Galaxy Note 7

Category:

Today's Views: 2067

Figure 52: Samsung

8.2.4 Quick Sort

Date range: 2022/03/31 - 2022/04/06

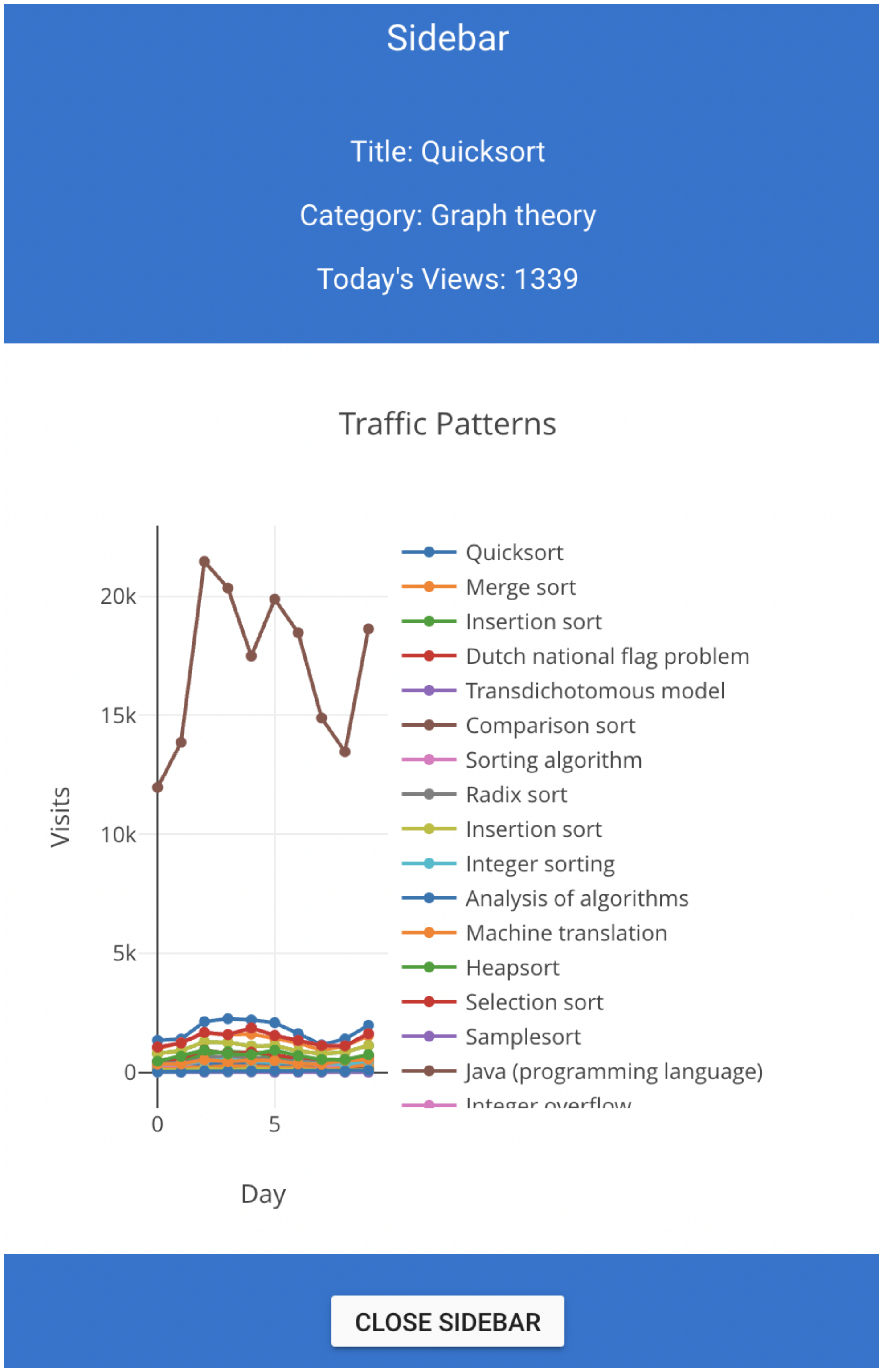
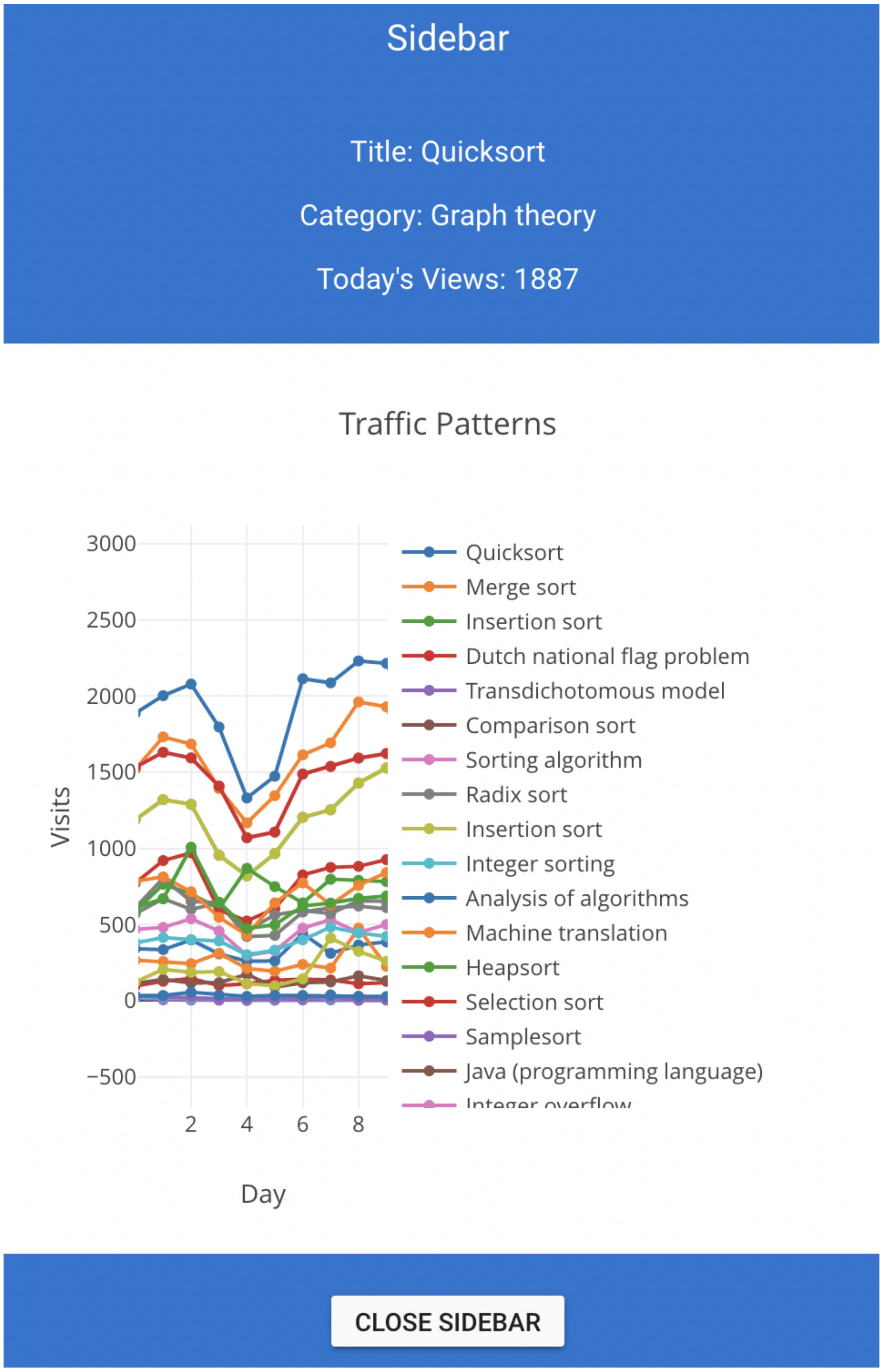


Figure 53: Quick Sort


Sidebar

Title: Quicksort

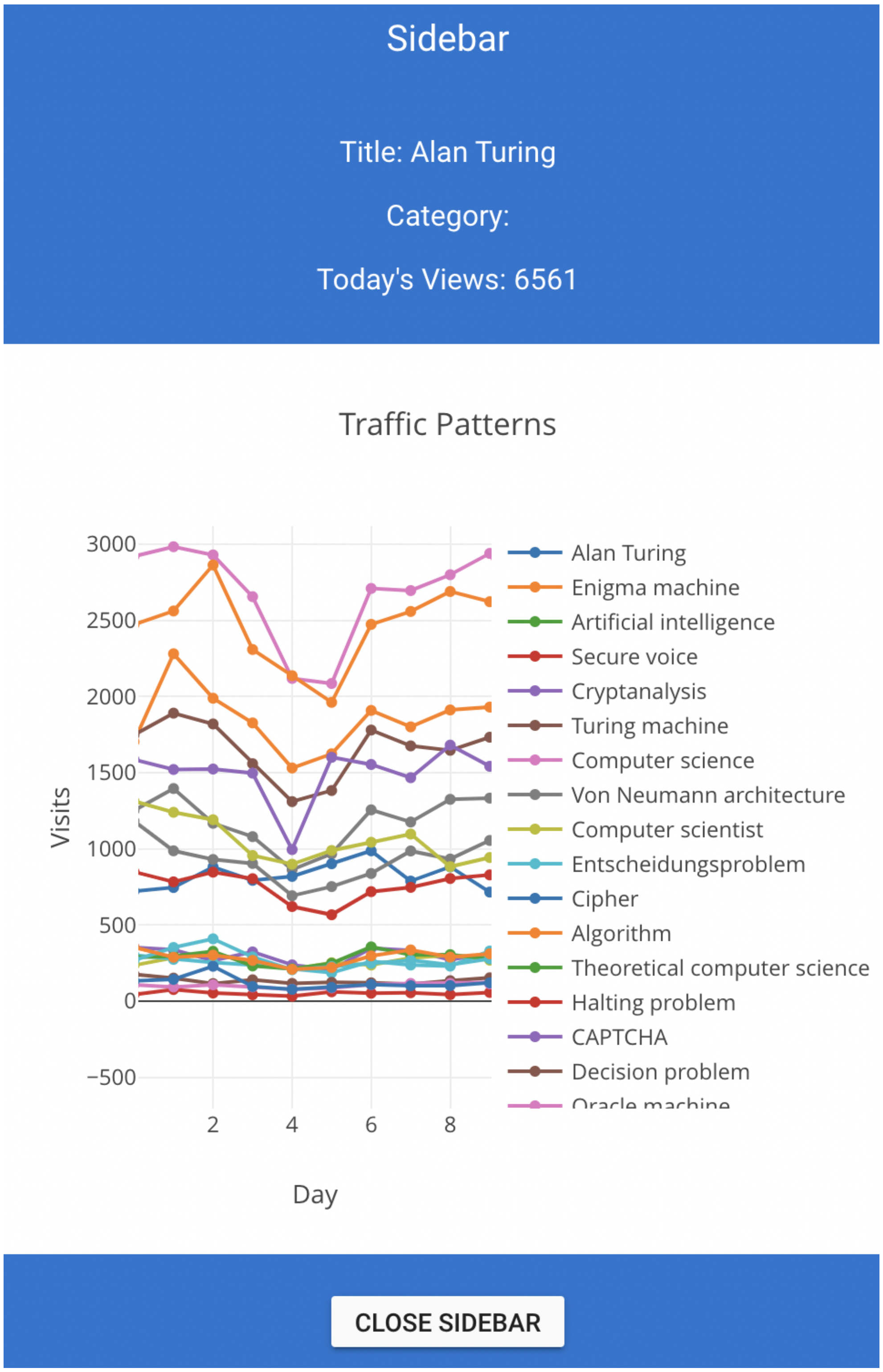
Category: Graph theory

Today's Views: 1887

Figure 54: Quick Sort

These plots show that the traffic patterns for various sorting algorithms match up well. This could be because sorting algorithms are quite technical, and readers of a technical article is more likely to do extensive research compared to readers of a scandal like the Cambridge Analytica incident.

8.2.5 Others

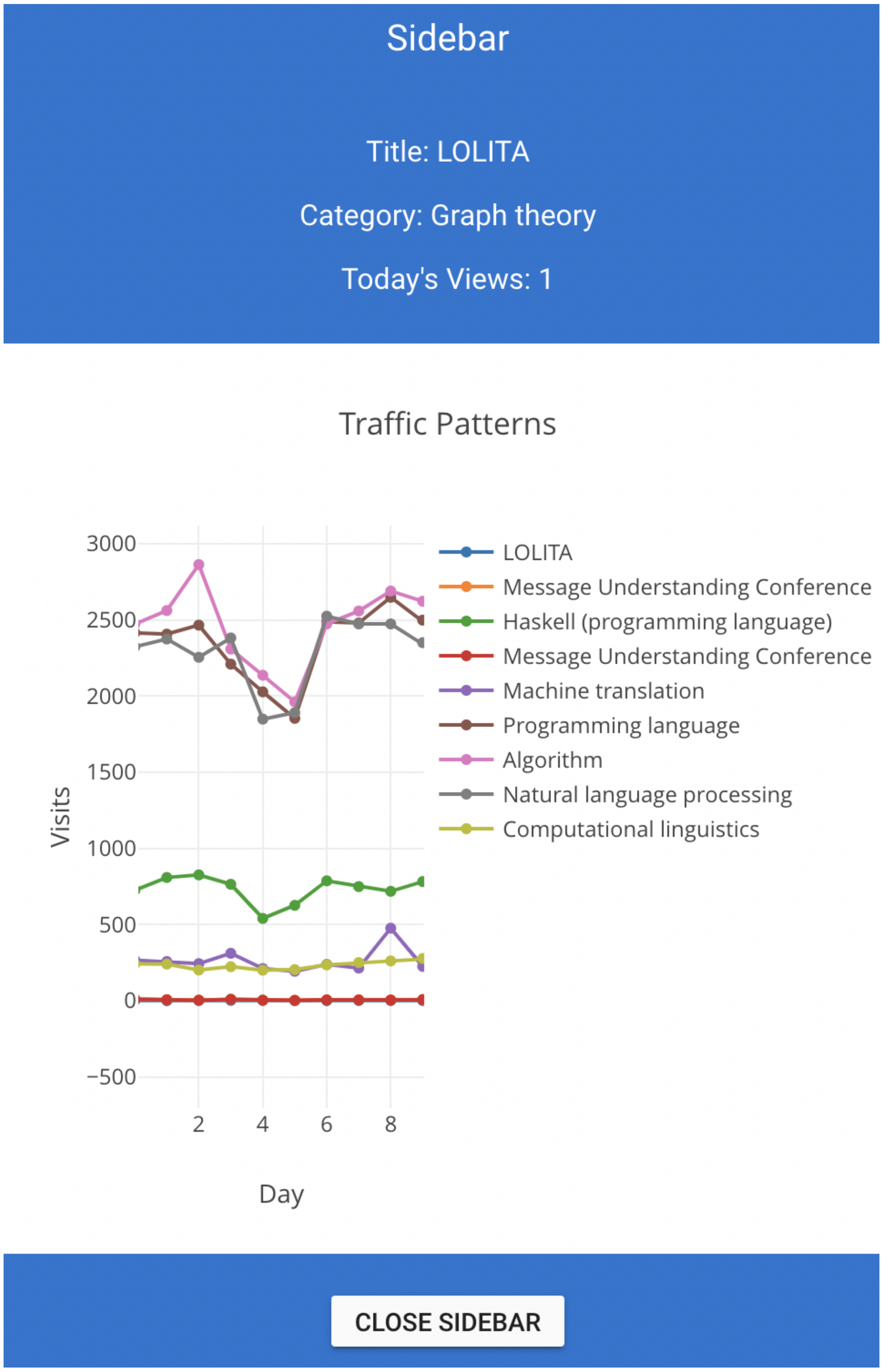

Sidebar

Title: Alan Turing

Category:

Today's Views: 6561

Figure 55: Alan Turing



Sidebar

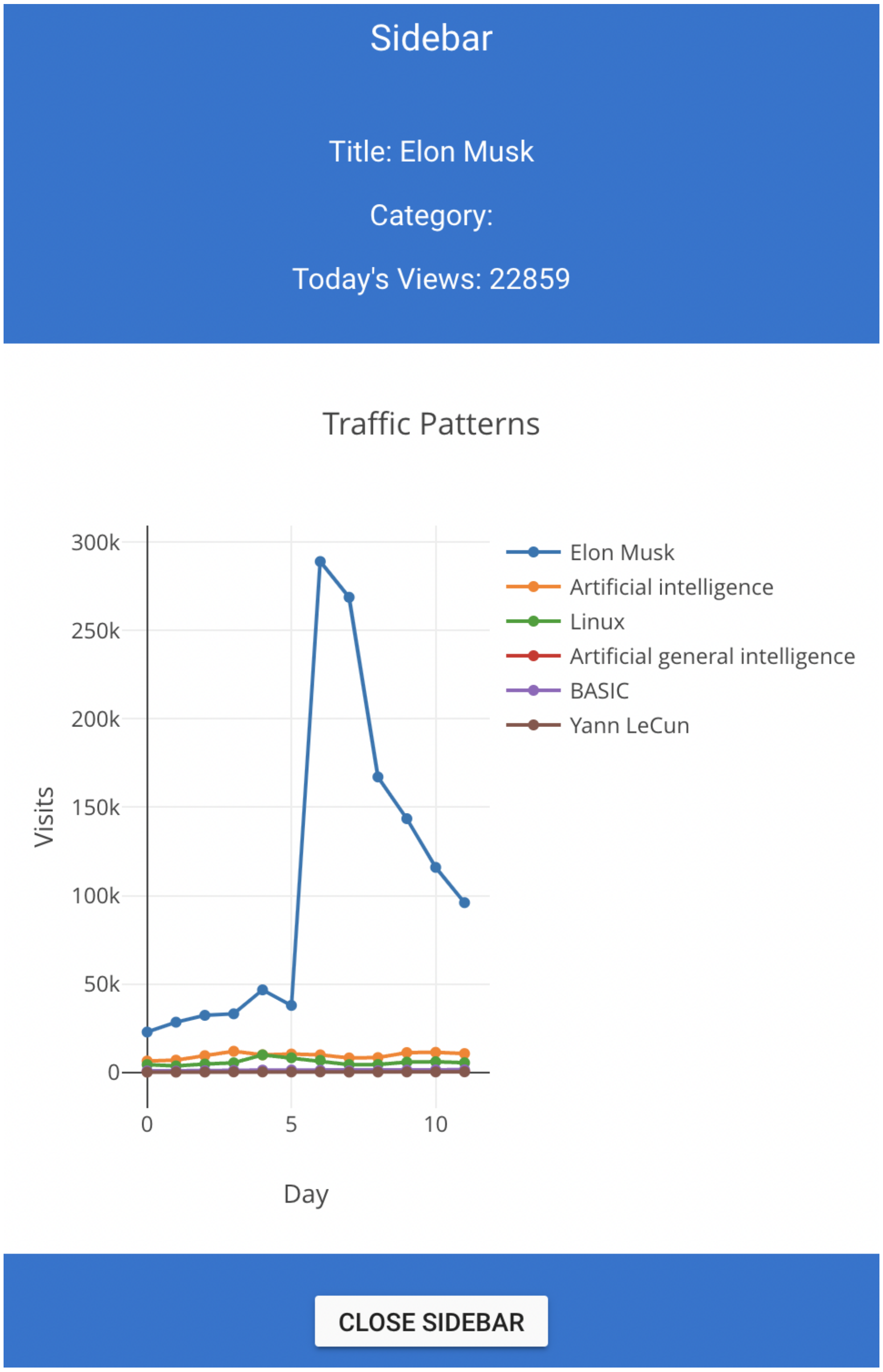
Title: LOLITA

Category: Graph theory

Today's Views: 1

Traffic Patterns
Visits
Day
3000
2500
2000
1500
1000
500
0
−500
2
4
6
8
LOLITA
Message Understanding Conference
Haskell (programming language)
Message Understanding Conference
Machine translation
Programming language
Algorithm
Natural language processing
Computational linguistics
CLOSE SIDEBAR

Figure 56: LOLITA



Sidebar

Title: Elon Musk

Category:

Today's Views: 22859

Figure 57: Elon Musk

9 Critical Evaluation

9.1 Requirements Met

The features of the final project have evolved drastically from what was outlined in the DOER at the start of last semester.

The first primary goal was to come up with a way of generating a graph structure. This was achieved through identifying articles in the field of Computer Science, constructing a corpus of them, and training a Doc2Vec model on this specialised corpus.

The second primary goal states that a topic (i.e. category) should contain nodes (i.e. articles) and each article should be connected to other nodes so the relationships can be tracked and investigated. The third primary goal states that there should be metrics for evaluating the progress of a given category. I strayed from the idea of 'progress' and focused only on the structure at a certain point in time (i.e. November 2021). Initially, I wanted to capture the progression in Computer Science categories as I thought it would be interesting to see when and how certain categories emerged and changed. Many subcategories within Computer Science have gone through several hype cycles (e.g. boom in interest followed by disappointment) and I thought they would be reflected in the Wikipedia content. I ended up focusing on just a set point in time largely due to the impracticality of downloading and processing more than one Wikipedia datadump. In order to investigate the progression of topics, data spanning a long time period is required. However, each English Wikipedia datadump is worth around 18 GB and running the categoriser takes too long.

The last primary goal was to create a webpage that displays the results of the project. This is satisfied. The user can explore different categories and articles in the field of Computer Science as they wish, and can specify the range of dates they want to view traffic information for.

The secondary goals were written under the assumption that time-series data for pageview stats of Wikipedia articles are available. I initially wanted some form of animation highlighting the ups and downs of the categories, which ties in to the also dropped idea of progression. Wikimedia used to provide a dataset that contains hourly statistics of every request to their hosts. Hourly data can be used to estimate how readers navigate from one article to the next. However, this project was discontinued in 2016. Starting July 2015, Wikimedia has a new API that provides page view statistics with a daily granularity. This is what was used in the project in the end.

The tertiary goal is to investigate how the public's perception of artificial intelligence related topics and articles have changed over time through tracking the amount of positive and negative words in the edit history. This goal fits in with my final project quite well as this also falls under natural language processing. I believe that given more time, the project can be extended to satisfy this goal.

9.2 What was Achieved

Initially, I wanted the frontend visualisation to be 3-dimensional. I believe 3-dimensionality encourages user interaction. However, the developer of OpenTSNE[6] said that dimension reduction into 3D space is not supported and will not be supported. This is because openTSNE implements an approximation algorithm called flt-sne which scales linearly with the number of points but exponentially in the number of dimensions. This means 3-dimensional visualisation will take exponentially longer than 2-dimensional visualisation.

The developer raises another good point which I have also thought about: 3-dimensional visualisations do not translate to 2-dimensional print. Even if I used 3-dimensional data on the frontend, I would still have to construct a 2-dimensional version for reporting the results.

The developer also mentions that an algorithm for t-SNE (Barnes-Hut) support embedding into 3-dimensional (i.e.   2 dimensions) space. However, this functionality was not tested extensively so I did not try it.

A feature I would like to highlight is the ability for the user to type the title of an article into the search bar, and the project will insert a node representing that article onto the Computer Science based graph structure. This is achieved through storing the trained model on the backend. When the user types in the title of an article in to the search bar, the request for estimating the vector position for that article within the graph structure is sent to the backend. The English Wikipedia API at api is used to retrieve the contents of that particular article, which is then tokenised and passed through the Doc2Vec model. The computed vector is then dimensionally reduced by OpenTSNE and the coordinates are returned to the frontend.

Title text input has not been optimised to be robust to the search. Particularly, there is an issue where the difference in capitalisation of the same title could result in vastly different views. For instance, inputting 'AlphaGo' would return a valid response, but inputting 'Alphago' would not. The search bar functionality demands stringent title input that conforms to the Wikipedia naming convention for capitalisation. However, ideally, a redirection should occur.

9.3 Filtering Articles Process Evaluation

Initially, I used categories identified by Pmernyei et al to select the relevant articles which are considered to be in the field of Computer Science. This process lead to a collection of around 2300 articles. The generated graph structure is shown below:

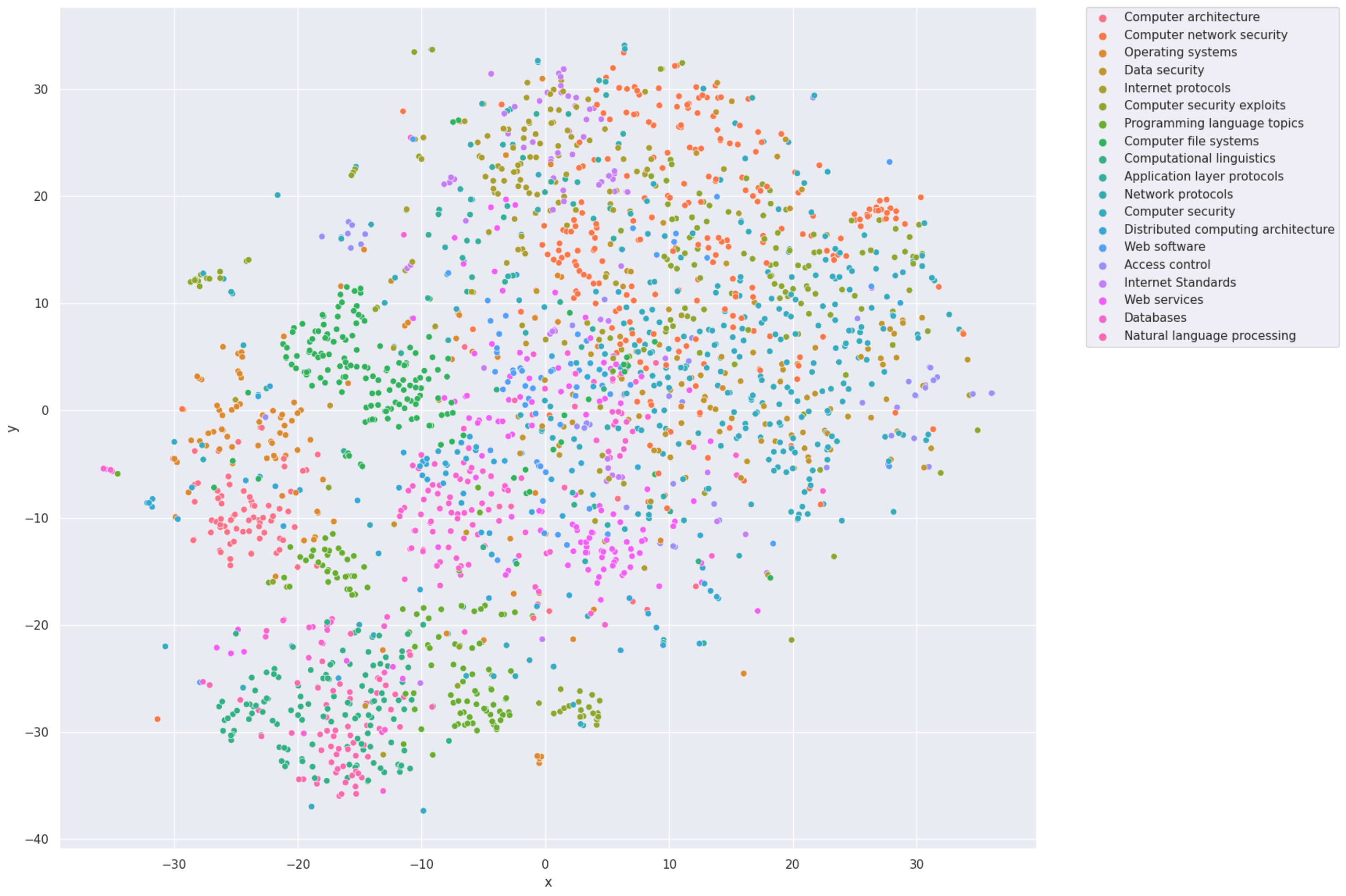


Figure 58: Graph Structure with 2300 Nodes

I believe this graph provides insight into the structure of the nodes. There are clusters that can be spotted quite easily. With 2300 articles, I thought the list of categories identified by Pmernyei et al did not extensively cover all of Computer Science, so I altered the initial stage of the pipeline and I attempt to extract more categories than the provided ones.

Even then, a large number of topics I consider to be in the field of Computer Science were not included. Hot topics such as internet of things, virtual reality, and machine learning also are not identified as notable categories. Some may argue that fields like these have gotten a life of their own, and have separated themselves from Computer Science as a core concept. However, I believe that a lot of them originated from Computer Science, and complement core Computer Science such as algorithms, so a harsh separation would do more harm than good. Furthermore, I realised that notable figures are not included at all. Alan Turing, who is regarded as the father of Computer

Science is mapped to a long list of categories (even sporting categories) but none fall within the confines of the categories Pmerneyei generated. Due to these reasons, I went back and added some categories (based on Dominic's map) to the list used for extracting Computer Science articles. The list more than doubled in size and resulted in an increase of around 3000 articles.

The pipeline/workflow was set up in a way that made it easy to go back and alter the filtering process. Articles that are already put in the database were not added redundantly to the database, only newly identified articles' content were added.

9.4 Corpus Usage

I would have liked to do some experiments on the effect of using a specialised corpus versus a general corpus. A general corpus could be the entirety of English Wikipedia.

9.5 Model and t-SNE Tuning

I realise that when it comes to visualisation, 'best' is a very subjective idea. In this project, I have tried out different hyperparameters for both the Doc2Vec model and the t-SNE dimension reduction algorithm. I ended up going with what I personally feel yields the **clearest** clustering. It should be noted that the hyperparameters for the Doc2Vec model and t-SNE did not affect the visualisation results as much as I expected, and another developer may well select a different combination of hyperparameters.

10 Conclusion

This project's objective was to construct a graph structure involving Wikipedia articles in the field of Computer Science and then visualise the results. The results are presented on an interactive component which allows the user to investigate the position, neighbours, and traffic patterns of an article they have an interest in.

I believe this project can be extended to include more languages. For this project, only the English Wikipedia was used as understanding English makes it easy to examine and verify the outputs at any stage. Now that the pipeline is set up, we only need to pass in data dumps of a different language. It would be interesting to see how our Doc2Vec model reacts to different grammar, syntax, and semantics of the different languages.

11 Bibliography

References

[1] Paolo Boldi and Corrado Monti. "Cleansing Wikipedia Categories Using Centrality". In: *Proceedings of the 25th International Conference Companion on World Wide Web*. WWW '16 Companion. Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, 2016, pp. 969–974. ISBN: 9781450341448. DOI: 10.1145/2872518.2891111. URL: https://doi.org/10.1145/2872518.2891111.

[2] *Cleansing Wikipedia Categories using Centrality*. https://github.com/corradomonti/wikipedia-categories. Accessed: 2022-04-06.

[3] *Ebbinghaus's Forgetting Curve*. https://www.mindtools.com/pages/article/forgetting-curve.htm. Accessed: 2022-04-06.

[4] Maurice Halbwachs. *On collective memory*. University of Chicago Press, 1992.

[5] *Hebbian Theory*. https://www.sciencedirect.com/topics/neuroscience/hebbian-theory. Accessed: 2022-04-06.

[6] *Increased the n components to 3?* https://github.com/pavlin-policar/openTSNE/issues/121. Accessed: 2022-04-06.

[7] Nattiya Kanhabua, Tu Ngoc Nguyen, and Claudia Niederée. "What triggers human remembering of events? A large-scale analysis of catalysts for collective memory in Wikipedia". In: *IEEE/ACM Joint Conference on Digital Libraries*. 2014, pp. 341–350. DOI: 10.1109/JCDL.2014.6970189.

[8] *Map of Computer Science*. https://www.informationisbeautifulawards.com/showcase/2333-map-of-computer-science. Accessed: 2022-04-06.

[9] Volodymyr Miz et al. "Anomaly detection in the dynamics of web and social networks using associative memory". In: *The World Wide Web Conference*. 2019, pp. 1290–1299.

[10] *Reingold Layout*. https://www.sciencedirect.com/topics/computer-science/reingold-layout. Accessed: 2022-04-06.

[11] *Reliability of Wikipedia*. https://en.wikipedia.org/wiki/Reliability_of_Wikipedia. Accessed: 2022-04-06.

[12] *The World Health Organization and Wikimedia Foundation expand access to trusted information about COVID-19 on Wikipedia*. https://www.who.int/news/item/22-10-2020-the-world-health-organization-and-wikimedia-foundation-expand-access-to-trusted-information-about-covid-19-on-wikipedia. Accessed: 2022-04-06.

[13] *Wiki-CS: A Wikipedia-Based Benchmark for Graph Neural Networks*. https://github.com/pmernyei/wiki-cs-dataset. Accessed: 2022-04-06.

[14] *Wikipedia:About*. https://en.wikipedia.org/wiki/Wikipedia:About. Accessed: 2022-04-06.

[15] *Wikipedia:FAQ/Categorization*. https://en.wikipedia.org/wiki/Wikipedia:FAQ/Categorization. Accessed: 2022-04-06.

[16] *Wikipedia:Size comparisons*. https://en.wikipedia.org/wiki/Wikipedia:Size_comparisons. Accessed: 2022-04-06.

[17] Tae Yano and Moonyoung Kang. "Taking advantage of wikipedia in natural language processing". In: (2016).

12 Appendix

12.1 User Guide

The *data processing* directory contains code for data processing as well as the database and Doc2Vec model. The *sh-app* directory contains frontend and backend code.

Run *npm i* to install relevant packages for the frontend.

Run *pip install -r requirements.txt* in *data processing* to install relevant packages for data processing.

To run the backend, in a new terminal, move to the *sh-app* directory and run *flask run*.

To run the frontend, in a new terminal, move to the *sh-app/api* directory and run *yarn start*.

12.2 Areas of Computer Science

```
{
    "Computer file systems": [
        "Computer file systems",
        "Network file systems",
        "Distributed file systems"
    ],
    "Databases": [
        "Databases"
    ],
    "Internet protocols": [
        "Internet protocols"
    ],
    "Computer security": [
        "Computer security",
        "Computer network security",
        "Access control",
        "Data security",
        "Computational trust",
        "Computer security exploits"
    ],
    "Web technology": [
        "Web software",
        "Web technology",
        "Web services",
        "Networking standards"
    ],
    "Computational linguistics": [
        "Computational linguistics"
    ],
    "Programming language topics": [
        "Programming language theory",
        "Programming language topics",
        "Programming language classification",
        "Programming language concepts"
    ],
    "Computer architecture": [
        "Computer architecture"
    ],
    "Operating systems": [
        "Operating systems",
        "Operating system technology",
        "Operating systems by architecture"
    ],
    "Distributed computing architecture": [
        "Distributed computing architecture"
    ],
    "Computational complexity theory": [
        "Computational complexity theory"
    ],
    "Information theory": [
        "Information theory"
    ],
    "Cryptography": [
        "Cryptography"
    ],
    "Algorithms": [
        "Analysis of algorithms",
        "Optimization algorithms and methods",
        "Lossless compression algorithms",
        "Computer arithmetic algorithms",
        "Cryptographic algorithms",
        "Parsing algorithms",
        "Concurrency control algorithms",
        "Distributed algorithms",
        "Graph algorithms",
        "Sorting algorithms",
        "Classification algorithms",
        "Networking algorithms"
    ],
    "Computability theory": [
        "Computability theory"
    ],
    "Supercomputing": [
        "Supercomputing"
    ],
    "Theoretical computer science": [
        "Theoretical computer science"
    ],
    "Programming languages": [
        "Programming languages",
        "Educational programming languages",
        "Concurrent programming languages"
    ],
    "Computational science": [
        "Computational science"
    ],
    "Augmented reality":[
        "Augmented reality"
    ],
    "Human-computer interaction": [
        "Human-computer interaction"
    ],
    "Internet of things": [
        "Internet of things"
    ],
    "Big data": [
        "Big data"
    ],
    "Hacking": [
        "Hacking (computer security)",
        "Hacking in the 1980s",
        "Hacking in the 1990s",
        "Hacking in the 2000s"
    ],
    "Machine learning": [
        "Applied machine learning",
        "Data mining and machine learning software",
        "Machine learning algorithms"
    ],
    "Computer vision": [
        "Computer vision",
        "Feature detection (computer vision)"
    ],
    "Image processing": [
        "Image processing"
    ],
    "Artificial intelligence": [
        "Artificial intelligence"
    ],
    "Natural language processing": [
        "Natural language processing"
    ],
    "Robotics": [
        "Robotics"
    ],
    "Graph theory": [
        "Graph theory"
    ],
    "Quantum computing": [
        "Quantum computing"
    ],
    "Data structures": [
        "Data structures"
    ],
    "Computer engineering": [
        "Computer engineering"
    ],
    "Compilers": [
        "Compilers"
    ],
    "Notable figures": [
        "Theoretical computer scientist",
        "Machine learning researchers",
        "Computer vision researchers",
        "Computer scientists",
        "Artificial intelligence researchers",
        "Natural language processing researchers"
    ]
}
```

12.3 Doc2Vec Parameter Tuning Counter Object Outputs

Dimension=30, Epoch=20

```
Counter({0: 5010, 415: 2, 178: 2, 1087: 2, 1139: 2, 3206: 2, 3075: 1, 2959: 1, 3963:
1, 788: 1, 2414: 1, 2717: 1, 335: 1, 2080: 1, 3673: 1, 218: 1, 1796: 1, 1485: 1, 3764:
1, 558: 1, 4005: 1, 3276: 1, 4859: 1, 489: 1, 3526: 1, 2539: 1, 1372: 1, 3226: 1, 3478:
1, 3431: 1, 568: 1, 3712: 1, 46: 1, 4701: 1, 4633: 1, 375: 1, 3589: 1, 1232: 1, 4981:
```

```
1, 2191: 1, 2406: 1, 4924: 1, 4887: 1, 2754: 1, 4900: 1, 4519: 1, 4236: 1, 132: 1,
2832: 1, 37: 1, 1680: 1, 1298: 1, 1894: 1, 3640: 1, 2375: 1, 3915: 1, 206: 1, 966:
1, 4471: 1, 384: 1, 8: 1, 4095: 1, 1254: 1, 4491: 1, 197: 1, 3624: 1, 2997: 1, 988:
1, 1010: 1, 73: 1, 3406: 1, 4014: 1, 2902: 1, 4516: 1, 1939: 1, 521: 1, 256: 1, 1655:
1, 1851: 1, 3270: 1, 2518: 1, 4620: 1, 726: 1, 3350: 1, 2201: 1, 41: 1, 410: 1, 3223:
1, 1224: 1, 1749: 1, 3036: 1, 4302: 1, 1573: 1, 1312: 1, 4702: 1, 764: 1, 2657: 1,
4579: 1, 1855: 1, 4803: 1, 3149: 1, 1786: 1, 287: 1, 4985: 1, 447: 1, 2762: 1, 426:
1, 671: 1, 2625: 1, 982: 1, 4651: 1, 2427: 1, 1590: 1, 1364: 1, 3278: 1, 481: 1, 4259:
1, 1486: 1, 3179: 1, 251: 1, 1988: 1, 833: 1, 4865: 1, 2329: 1, 2744: 1, 3995: 1, 4539:
1, 2066: 1, 4343: 1, 4271: 1, 2775: 1, 1986: 1, 4: 1, 3671: 1, 2912: 1, 587: 1, 1530:
1, 4599: 1, 2738: 1, 70: 1, 4967: 1, 2264: 1, 2942: 1, 1794: 1, 3095: 1, 2256: 1, 29:
1, 3055: 1, 14: 1, 1140: 1, 4822: 1, 1218: 1, 4728: 1, 659: 1, 2554: 1, 4849: 1, 2771:
1, 2542: 1, 3635: 1, 4197: 1, 4737: 1, 3610: 1, 2188: 1, 2751: 1, 897: 1, 2660: 1,
123: 1, 3549: 1, 2930: 1, 3111: 1, 1413: 1, 3621: 1, 354: 1, 1752: 1, 979: 1, 1445:
1, 1166: 1, 2861: 1, 1795: 1, 2839: 1, 4576: 1, 28: 1, 2800: 1, 635: 1, 474: 1, 1770:
1, 3447: 1, 246: 1, 1107: 1, 650: 1, 3876: 1, 515: 1, 2721: 1, 3824: 1, 1606: 1, 3118:
1, 1129: 1, 919: 1, 477: 1, 2182: 1, 4837: 1, 1591: 1, 3083: 1, 4517: 1, 2803: 1, 1042:
1, 4027: 1, 3993: 1, 3704: 1, 4857: 1, 1872: 1, 3932: 1, 3613: 1, 11: 1, 3345: 1, 1219:
1})
```
Dimension=30, Epoch=30

```
Counter({0: 5011, 3333: 2, 29: 2, 3108: 2, 3081: 1, 3707: 1, 3956: 1, 351: 1, 1089:
1, 2075: 1, 2470: 1, 4098: 1, 3438: 1, 2721: 1, 2308: 1, 2622: 1, 2840: 1, 739: 1,
2939: 1, 3219: 1, 653: 1, 2022: 1, 530: 1, 706: 1, 3000: 1, 3870: 1, 4532: 1, 1149:
1, 2743: 1, 375: 1, 3526: 1, 17: 1, 389: 1, 3966: 1, 190: 1, 4297: 1, 901: 1, 1131:
1, 2579: 1, 1775: 1, 2190: 1, 4444: 1, 3716: 1, 2883: 1, 1358: 1, 3294: 1, 606: 1,
2735: 1, 769: 1, 323: 1, 1779: 1, 2014: 1, 2453: 1, 4839: 1, 2205: 1, 363: 1, 2147:
1, 3205: 1, 483: 1, 659: 1, 215: 1, 729: 1, 2173: 1, 93: 1, 1437: 1, 2383: 1, 463:
1, 1917: 1, 7: 1, 4386: 1, 3507: 1, 11: 1, 1335: 1, 4807: 1, 785: 1, 529: 1, 2253:
1, 2211: 1, 2197: 1, 609: 1, 3849: 1, 3183: 1, 307: 1, 989: 1, 2131: 1, 4138: 1, 8:
1, 508: 1, 3448: 1, 967: 1, 2442: 1, 944: 1, 1043: 1, 3955: 1, 2772: 1, 512: 1, 196:
1, 1827: 1, 4002: 1, 1685: 1, 1720: 1, 3929: 1, 2626: 1, 193: 1, 4698: 1, 109: 1, 3424:
1, 2723: 1, 428: 1, 3519: 1, 271: 1, 4310: 1, 4110: 1, 788: 1, 466: 1, 4071: 1, 3351:
1, 1830: 1, 4092: 1, 1483: 1, 44: 1, 2664: 1, 3986: 1, 1806: 1, 694: 1, 4517: 1, 3523:
1, 2228: 1, 2209: 1, 1439: 1, 2762: 1, 576: 1, 105: 1, 1342: 1, 1: 1, 4921: 1, 1238:
1, 3725: 1, 3157: 1, 2416: 1, 2672: 1, 77: 1, 4422: 1, 1187: 1, 3536: 1, 593: 1, 32:
1, 3895: 1, 2020: 1, 388: 1, 4224: 1, 4983: 1, 580: 1, 82: 1, 876: 1, 847: 1, 3256:
1, 49: 1, 258: 1, 4746: 1, 4530: 1, 3965: 1, 3626: 1, 2366: 1, 1587: 1, 3552: 1, 201:
1, 3383: 1, 2468: 1, 2440: 1, 476: 1, 1808: 1, 4962: 1, 1041: 1, 1623: 1, 3060: 1,
195: 1, 3529: 1, 5002: 1, 3979: 1, 4496: 1, 18: 1, 256: 1, 605: 1, 2117: 1, 1470: 1,
2083: 1, 2964: 1, 4028: 1, 2235: 1, 1235: 1, 3939: 1, 1078: 1, 3284: 1, 36: 1, 2625:
1, 2353: 1, 2629: 1, 608: 1, 545: 1, 2878: 1, 4611: 1, 2700: 1, 2748: 1, 1763: 1, 1247:
1, 454: 1, 993: 1, 377: 1, 3711: 1, 965: 1, 249: 1, 3264: 1, 1379: 1, 997: 1, 2519:
1, 40: 1})
```
Dimension=40, Epoch=10

```
Counter({0: 5000, 1: 6, 3876: 2, 3896: 2, 4001: 2, 7: 2, 2: 2, 169: 2, 1589: 2, 2997:
1, 4480: 1, 4494: 1, 3879: 1, 3858: 1, 2958: 1, 4394: 1, 2768: 1, 3778: 1, 2978: 1,
4952: 1, 396: 1, 1765: 1, 4720: 1, 2112: 1, 1173: 1, 930: 1, 2161: 1, 3422: 1, 1877:
```

```
1, 641: 1, 2010: 1, 2328: 1, 4734: 1, 189: 1, 4200: 1, 4352: 1, 1571: 1, 1906: 1, 2090:
1, 2114: 1, 177: 1, 1673: 1, 1156: 1, 2707: 1, 2205: 1, 1596: 1, 1910: 1, 411: 1, 4262:
1, 3024: 1, 77: 1, 1286: 1, 2840: 1, 3161: 1, 3428: 1, 2093: 1, 3416: 1, 4934: 1, 279:
1, 1431: 1, 438: 1, 3350: 1, 4263: 1, 1186: 1, 575: 1, 2019: 1, 4845: 1, 3064: 1, 4439:
1, 3636: 1, 3763: 1, 4870: 1, 305: 1, 4995: 1, 140: 1, 959: 1, 1387: 1, 3257: 1, 625:
1, 1535: 1, 3084: 1, 229: 1, 3523: 1, 4756: 1, 4268: 1, 162: 1, 2564: 1, 989: 1, 161:
1, 592: 1, 4426: 1, 3347: 1, 2606: 1, 3880: 1, 3910: 1, 2281: 1, 2686: 1, 971: 1, 3494:
1, 1827: 1, 4705: 1, 1815: 1, 4779: 1, 1510: 1, 2645: 1, 1681: 1, 2799: 1, 1726: 1,
4148: 1, 81: 1, 4287: 1, 2861: 1, 614: 1, 403: 1, 3769: 1, 2984: 1, 1588: 1, 4659:
1, 4803: 1, 1745: 1, 152: 1, 1892: 1, 865: 1, 4941: 1, 37: 1, 3471: 1, 795: 1, 4310:
1, 116: 1, 4127: 1, 3445: 1, 4822: 1, 398: 1, 3929: 1, 135: 1, 3806: 1, 4773: 1, 3:
1, 699: 1, 2483: 1, 1416: 1, 4393: 1, 198: 1, 1903: 1, 58: 1, 1095: 1, 2296: 1, 2280:
1, 4850: 1, 3180: 1, 4817: 1, 3506: 1, 4826: 1, 701: 1, 3056: 1, 2180: 1, 1763: 1,
4318: 1, 448: 1, 580: 1, 2785: 1, 3059: 1, 2993: 1, 1527: 1, 1057: 1, 3860: 1, 3924:
1, 413: 1, 4868: 1, 4445: 1, 4710: 1, 3073: 1, 4991: 1, 2846: 1, 117: 1, 3135: 1, 5001:
1, 1039: 1, 1066: 1, 3326: 1, 4560: 1, 2987: 1, 183: 1, 2217: 1, 1554: 1, 4857: 1,
2548: 1, 3842: 1, 547: 1, 4600: 1, 3208: 1, 1989: 1, 4388: 1, 96: 1, 1025: 1, 1334:
1, 775: 1, 3054: 1, 714: 1, 2357: 1, 2181: 1, 2128: 1, 43: 1, 2451: 1, 1759: 1, 3679:
1, 3356: 1, 2101: 1, 3043: 1, 2666: 1, 3151: 1, 947: 1, 188: 1, 3742: 1, 4680: 1, 3465:
1, 502: 1, 2364: 1, 4371: 1})
```

Dimension=40, Epoch=20

```
Counter({0: 5007, 2784: 2, 870: 2, 2394: 2, 2256: 2, 2283: 2, 707: 2, 820: 2, 2875:
2, 21: 2, 3: 2, 4764: 2, 4508: 2, 289: 2, 4690: 1, 1819: 1, 2569: 1, 937: 1, 1263:
1, 2475: 1, 1681: 1, 1785: 1, 3837: 1, 1501: 1, 2793: 1, 1080: 1, 4290: 1, 3982: 1,
683: 1, 4628: 1, 58: 1, 823: 1, 1899: 1, 1424: 1, 2712: 1, 2539: 1, 3713: 1, 341: 1,
189: 1, 567: 1, 4864: 1, 2159: 1, 288: 1, 1519: 1, 988: 1, 4784: 1, 1200: 1, 4555:
1, 1828: 1, 4986: 1, 817: 1, 2143: 1, 4878: 1, 601: 1, 699: 1, 166: 1, 1565: 1, 1381:
1, 2533: 1, 2850: 1, 2963: 1, 106: 1, 1312: 1, 3924: 1, 750: 1, 149: 1, 1055: 1, 2807:
1, 540: 1, 2817: 1, 3340: 1, 3821: 1, 2495: 1, 3684: 1, 503: 1, 290: 1, 4971: 1, 4739:
1, 3467: 1, 2503: 1, 4240: 1, 4905: 1, 885: 1, 1059: 1, 4824: 1, 4831: 1, 327: 1, 39:
1, 1135: 1, 3694: 1, 1350: 1, 4782: 1, 4101: 1, 4774: 1, 4751: 1, 113: 1, 1989: 1,
863: 1, 3974: 1, 4853: 1, 3310: 1, 2941: 1, 4277: 1, 321: 1, 2415: 1, 271: 1, 1716:
1, 37: 1, 4926: 1, 2791: 1, 2300: 1, 86: 1, 1747: 1, 1536: 1, 3919: 1, 891: 1, 2583:
1, 4386: 1, 3034: 1, 498: 1, 3244: 1, 4617: 1, 2405: 1, 2565: 1, 2739: 1, 2612: 1,
3158: 1, 2024: 1, 2756: 1, 2763: 1, 1169: 1, 1444: 1, 2: 1, 1517: 1, 3959: 1, 1: 1,
3409: 1, 1877: 1, 1584: 1, 19: 1, 369: 1, 32: 1, 2827: 1, 3275: 1, 769: 1, 3081: 1,
1853: 1, 2653: 1, 374: 1, 2262: 1, 4902: 1, 3788: 1, 242: 1, 4888: 1, 3250: 1, 3357:
1, 4647: 1, 231: 1, 877: 1, 1293: 1, 1337: 1, 921: 1, 2464: 1, 2179: 1, 1415: 1, 1354:
1, 3866: 1, 1900: 1, 1133: 1, 2873: 1, 2965: 1, 1503: 1, 2235: 1, 2775: 1, 3325: 1,
493: 1, 457: 1, 2197: 1, 5008: 1, 35: 1, 4893: 1, 728: 1, 3682: 1, 3809: 1, 4063: 1,
2020: 1, 2352: 1, 3871: 1, 1705: 1, 98: 1, 1615: 1, 3017: 1, 2061: 1, 62: 1, 1041:
1, 4657: 1, 4953: 1, 2102: 1, 2635: 1, 1844: 1, 2544: 1, 1134: 1, 4025: 1, 4030: 1,
960: 1, 2664: 1, 4765: 1, 3648: 1, 4911: 1, 12: 1, 2735: 1})
```

Dimension=40, Epoch=30

```
Counter({0: 5009, 1099: 2, 145: 2, 1652: 2, 1675: 2, 314: 2, 2: 2, 262: 2, 143: 2,
2905: 1, 4634: 1, 410: 1, 3183: 1, 1999: 1, 3435: 1, 54: 1, 2346: 1, 2724: 1, 341:
1, 2495: 1, 2613: 1, 4290: 1, 3574: 1, 350: 1, 1638: 1, 3006: 1, 1855: 1, 645: 1, 5:
```

```
1, 598: 1, 1105: 1, 4045: 1, 3052: 1, 583: 1, 1787: 1, 734: 1, 4380: 1, 34: 1, 1980:
1, 2739: 1, 74: 1, 4309: 1, 677: 1, 1161: 1, 1683: 1, 3334: 1, 3834: 1, 1630: 1, 2909:
1, 4102: 1, 1808: 1, 4340: 1, 2541: 1, 3675: 1, 24: 1, 1272: 1, 2156: 1, 1512: 1, 4091:
1, 1702: 1, 2515: 1, 4620: 1, 2978: 1, 1559: 1, 304: 1, 128: 1, 21: 1, 1177: 1, 4088:
1, 1183: 1, 2406: 1, 4702: 1, 619: 1, 4595: 1, 7: 1, 1254: 1, 4810: 1, 11: 1, 3031:
1, 4640: 1, 1976: 1, 4362: 1, 963: 1, 456: 1, 591: 1, 800: 1, 4051: 1, 77: 1, 3886:
1, 2902: 1, 1733: 1, 1: 1, 181: 1, 3603: 1, 3373: 1, 1298: 1, 4152: 1, 4798: 1, 4974:
1, 1383: 1, 212: 1, 1225: 1, 4402: 1, 3974: 1, 1708: 1, 4902: 1, 4688: 1, 97: 1, 4988:
1, 42: 1, 4880: 1, 4454: 1, 1209: 1, 4571: 1, 2580: 1, 960: 1, 845: 1, 4584: 1, 1035:
1, 4873: 1, 2246: 1, 81: 1, 3705: 1, 146: 1, 621: 1, 1498: 1, 498: 1, 1276: 1, 1262:
1, 4128: 1, 1018: 1, 227: 1, 3223: 1, 231: 1, 2883: 1, 744: 1, 1204: 1, 521: 1, 12:
1, 4006: 1, 3691: 1, 165: 1, 3330: 1, 2139: 1, 2806: 1, 1464: 1, 3786: 1, 117: 1, 2899:
1, 3583: 1, 1745: 1, 2785: 1, 184: 1, 1992: 1, 4871: 1, 508: 1, 5007: 1, 294: 1, 3620:
1, 1592: 1, 4601: 1, 3453: 1, 4146: 1, 313: 1, 774: 1, 266: 1, 1277: 1, 1852: 1, 3001:
1, 795: 1, 1886: 1, 1678: 1, 2355: 1, 104: 1, 3983: 1, 130: 1, 94: 1, 4907: 1, 9: 1,
4085: 1, 198: 1, 3962: 1, 3734: 1, 3433: 1, 3094: 1, 3462: 1, 4860: 1, 44: 1, 2274:
1, 2081: 1, 3870: 1, 2863: 1, 8: 1, 2960: 1, 2384: 1, 1793: 1, 3586: 1, 441: 1, 267:
1, 3928: 1, 3190: 1, 1179: 1, 2369: 1, 4774: 1, 431: 1, 1110: 1, 2664: 1, 4929: 1,
3767: 1, 4608: 1, 1650: 1, 1932: 1, 3257: 1, 1457: 1})
```
Dimension=50, Epoch=10

```
Counter({0: 5004, 4: 4, 1140: 2, 885: 2, 4121: 1, 3529: 1, 1476: 1, 515: 1, 2: 1, 4370:
1, 1888: 1, 1396: 1, 167: 1, 165: 1, 4309: 1, 2331: 1, 2340: 1, 566: 1, 3029: 1, 752:
1, 4863: 1, 1286: 1, 1950: 1, 4772: 1, 776: 1, 2681: 1, 815: 1, 1301: 1, 833: 1, 417:
1, 4776: 1, 1: 1, 4711: 1, 4471: 1, 1829: 1, 1886: 1, 561: 1, 164: 1, 124: 1, 1363:
1, 4378: 1, 769: 1, 1988: 1, 4621: 1, 2680: 1, 3749: 1, 672: 1, 521: 1, 1672: 1, 2822:
1, 249: 1, 3520: 1, 3071: 1, 3775: 1, 4014: 1, 4869: 1, 2342: 1, 4791: 1, 4356: 1,
3849: 1, 735: 1, 568: 1, 4429: 1, 2255: 1, 49: 1, 2953: 1, 3957: 1, 2414: 1, 3692:
1, 2324: 1, 59: 1, 4992: 1, 3147: 1, 2847: 1, 3926: 1, 2281: 1, 4203: 1, 829: 1, 3807:
1, 306: 1, 676: 1, 2364: 1, 2341: 1, 2095: 1, 4282: 1, 5001: 1, 4409: 1, 210: 1, 402:
1, 608: 1, 4577: 1, 1839: 1, 1360: 1, 3982: 1, 2559: 1, 4251: 1, 4834: 1, 3985: 1,
3010: 1, 4914: 1, 1689: 1, 4778: 1, 2216: 1, 5005: 1, 1849: 1, 2465: 1, 1275: 1, 1477:
1, 3611: 1, 201: 1, 2933: 1, 4925: 1, 3102: 1, 2298: 1, 4086: 1, 3329: 1, 1263: 1,
5004: 1, 788: 1, 2201: 1, 2263: 1, 1993: 1, 180: 1, 490: 1, 1298: 1, 3525: 1, 4725:
1, 3156: 1, 1940: 1, 3791: 1, 4042: 1, 4301: 1, 4998: 1, 29: 1, 4630: 1, 185: 1, 2729:
1, 2812: 1, 219: 1, 4890: 1, 4224: 1, 4396: 1, 136: 1, 4424: 1, 4484: 1, 3962: 1, 890:
1, 3135: 1, 4524: 1, 1813: 1, 4283: 1, 126: 1, 1678: 1, 4855: 1, 1901: 1, 89: 1, 2300:
1, 3315: 1, 2039: 1, 3264: 1, 926: 1, 1488: 1, 3417: 1, 3512: 1, 4168: 1, 4628: 1,
3443: 1, 3574: 1, 1711: 1, 1972: 1, 4564: 1, 299: 1, 4217: 1, 4074: 1, 863: 1, 4651:
1, 2569: 1, 733: 1, 2504: 1, 4937: 1, 1655: 1, 2362: 1, 273: 1, 643: 1, 4423: 1, 188:
1, 387: 1, 929: 1, 3368: 1, 3005: 1, 1381: 1, 1543: 1, 3630: 1, 1793: 1, 3: 1, 1340:
1, 519: 1, 972: 1, 912: 1, 1227: 1, 11: 1, 215: 1, 2656: 1, 1929: 1, 764: 1, 4581:
1, 1094: 1, 1238: 1, 123: 1, 4721: 1, 2117: 1, 4216: 1, 1568: 1, 4330: 1, 3636: 1,
4975: 1, 329: 1, 4245: 1, 377: 1, 800: 1, 1945: 1, 146: 1})
```
Dimension=50, Epoch=30

```
Counter({0: 5009, 2389: 2, 4: 2, 201: 2, 2304: 2, 3089: 2, 1034: 2, 1511: 2, 371: 2,
541: 1, 3977: 1, 3170: 1, 821: 1, 1575: 1, 515: 1, 1397: 1, 2814: 1, 2921: 1, 421:
1, 3378: 1, 1948: 1, 1799: 1, 3814: 1, 404: 1, 1046: 1, 2688: 1, 702: 1, 2988: 1, 863:
```

```
1, 3268: 1, 4714: 1, 1308: 1, 1637: 1, 2727: 1, 4256: 1, 126: 1, 1355: 1, 2749: 1,
144: 1, 3375: 1, 659: 1, 3868: 1, 3412: 1, 3455: 1, 3512: 1, 3948: 1, 849: 1, 922:
1, 2012: 1, 366: 1, 2279: 1, 3811: 1, 397: 1, 1982: 1, 806: 1, 4719: 1, 2796: 1, 414:
1, 2126: 1, 892: 1, 2996: 1, 460: 1, 299: 1, 67: 1, 2854: 1, 850: 1, 234: 1, 1493:
1, 3023: 1, 4529: 1, 4151: 1, 2513: 1, 40: 1, 2991: 1, 1184: 1, 167: 1, 534: 1, 2588:
1, 281: 1, 4277: 1, 4586: 1, 1287: 1, 2237: 1, 1490: 1, 4506: 1, 1295: 1, 11: 1, 712:
1, 4559: 1, 4325: 1, 276: 1, 223: 1, 2583: 1, 2741: 1, 2002: 1, 4574: 1, 280: 1, 3704:
1, 664: 1, 2642: 1, 4964: 1, 4338: 1, 114: 1, 30: 1, 349: 1, 3035: 1, 55: 1, 2777:
1, 70: 1, 377: 1, 1925: 1, 1263: 1, 1026: 1, 1660: 1, 1454: 1, 3679: 1, 683: 1, 1753:
1, 33: 1, 2436: 1, 2091: 1, 1350: 1, 4999: 1, 432: 1, 3407: 1, 419: 1, 2025: 1, 2395:
1, 2951: 1, 3787: 1, 3206: 1, 3770: 1, 1661: 1, 1: 1, 867: 1, 4283: 1, 2472: 1, 2682:
1, 14: 1, 1929: 1, 265: 1, 3405: 1, 3146: 1, 1346: 1, 4035: 1, 841: 1, 319: 1, 1675:
1, 4436: 1, 2723: 1, 1568: 1, 218: 1, 3836: 1, 3720: 1, 2431: 1, 3941: 1, 3457: 1,
2531: 1, 3821: 1, 1692: 1, 1367: 1, 3109: 1, 1586: 1, 4689: 1, 465: 1, 4267: 1, 775:
1, 1213: 1, 4133: 1, 1424: 1, 753: 1, 3578: 1, 872: 1, 330: 1, 2819: 1, 1414: 1, 1468:
1, 1741: 1, 8: 1, 2357: 1, 2393: 1, 4465: 1, 3036: 1, 1327: 1, 2640: 1, 1952: 1, 564:
1, 2429: 1, 1927: 1, 1281: 1, 3180: 1, 5: 1, 3960: 1, 2693: 1, 4803: 1, 2477: 1, 211:
1, 4042: 1, 379: 1, 3064: 1, 1202: 1, 2070: 1, 4290: 1, 2089: 1, 2224: 1, 4295: 1,
1909: 1, 2993: 1, 744: 1, 3366: 1, 3648: 1, 2: 1, 523: 1, 1597: 1})
```

Dimension=50, Epoch=50

```
Counter({0: 5015, 2: 4, 1: 3, 219: 2, 55: 2, 4162: 2, 357: 2, 23: 2, 461: 2, 1873:
1, 1653: 1, 360: 1, 5006: 1, 1270: 1, 4123: 1, 2111: 1, 3843: 1, 3323: 1, 1334: 1,
4610: 1, 3613: 1, 2606: 1, 939: 1, 3165: 1, 657: 1, 3819: 1, 1775: 1, 699: 1, 3285:
1, 499: 1, 3400: 1, 2228: 1, 3882: 1, 1936: 1, 973: 1, 72: 1, 3051: 1, 17: 1, 902:
1, 246: 1, 57: 1, 1847: 1, 334: 1, 1740: 1, 2817: 1, 1184: 1, 2651: 1, 2364: 1, 810:
1, 3580: 1, 4832: 1, 4881: 1, 3438: 1, 323: 1, 430: 1, 310: 1, 3657: 1, 1985: 1, 4181:
1, 910: 1, 4874: 1, 265: 1, 2879: 1, 437: 1, 2252: 1, 491: 1, 4736: 1, 38: 1, 4008:
1, 1858: 1, 86: 1, 32: 1, 1684: 1, 4800: 1, 1660: 1, 2912: 1, 3836: 1, 286: 1, 3757:
1, 173: 1, 2833: 1, 3066: 1, 4151: 1, 944: 1, 3119: 1, 4876: 1, 509: 1, 1813: 1, 2066:
1, 2257: 1, 1650: 1, 2830: 1, 1008: 1, 19: 1, 4889: 1, 2622: 1, 543: 1, 4447: 1, 901:
1, 4673: 1, 25: 1, 4344: 1, 4709: 1, 2148: 1, 16: 1, 3423: 1, 2899: 1, 168: 1, 2426:
1, 214: 1, 2142: 1, 232: 1, 1032: 1, 480: 1, 12: 1, 4285: 1, 1480: 1, 2712: 1, 516:
1, 3644: 1, 2225: 1, 2576: 1, 1254: 1, 268: 1, 4195: 1, 501: 1, 1869: 1, 3958: 1, 1357:
1, 4642: 1, 3626: 1, 3216: 1, 365: 1, 1224: 1, 2323: 1, 1596: 1, 2423: 1, 2886: 1,
3929: 1, 1982: 1, 4671: 1, 99: 1, 3589: 1, 1484: 1, 4001: 1, 1140: 1, 2776: 1, 1693:
1, 2741: 1, 3959: 1, 722: 1, 2847: 1, 1536: 1, 4366: 1, 2644: 1, 1577: 1, 102: 1, 4146:
1, 4096: 1, 4247: 1, 4737: 1, 1046: 1, 1235: 1, 273: 1, 3965: 1, 536: 1, 3065: 1, 4358:
1, 4890: 1, 977: 1, 1879: 1, 804: 1, 21: 1, 3599: 1, 1350: 1, 3063: 1, 1878: 1, 2031:
1, 1431: 1, 4764: 1, 4239: 1, 4583: 1, 2749: 1, 4272: 1, 4564: 1, 4467: 1, 11: 1, 1117:
1, 50: 1, 3625: 1, 4481: 1, 2502: 1, 4731: 1, 3919: 1, 4679: 1, 4869: 1, 4040: 1, 3566:
1, 1071: 1, 54: 1, 4807: 1, 1074: 1, 1058: 1, 4602: 1, 4791: 1})
```

Dimension=50, Epoch=80

```
Counter({0: 5020, 1: 6, 220: 2, 4: 2, 44: 2, 16: 2, 842: 2, 3360: 2, 4643: 2, 2940:
2, 397: 1, 3038: 1, 1748: 1, 2970: 1, 4125: 1, 724: 1, 1747: 1, 3573: 1, 3639: 1, 1063:
1, 31: 1, 3589: 1, 1508: 1, 4170: 1, 2812: 1, 2529: 1, 1595: 1, 1411: 1, 500: 1, 2306:
1, 66: 1, 278: 1, 3614: 1, 4877: 1, 2653: 1, 4869: 1, 4216: 1, 5: 1, 3756: 1, 158:
1, 436: 1, 4168: 1, 4852: 1, 2234: 1, 1870: 1, 3136: 1, 74: 1, 1619: 1, 4299: 1, 4953:
```

```
1, 4395: 1, 155: 1, 37: 1, 67: 1, 664: 1, 345: 1, 244: 1, 3158: 1, 1226: 1, 1086: 1,
4018: 1, 2103: 1, 405: 1, 1583: 1, 328: 1, 3383: 1, 484: 1, 2942: 1, 6: 1, 1432: 1,
4634: 1, 2414: 1, 3239: 1, 1322: 1, 1541: 1, 3953: 1, 770: 1, 4147: 1, 265: 1, 4584:
1, 4205: 1, 1087: 1, 4573: 1, 1014: 1, 800: 1, 3637: 1, 1234: 1, 488: 1, 4676: 1, 469:
1, 195: 1, 4384: 1, 2369: 1, 2929: 1, 1556: 1, 124: 1, 1305: 1, 4961: 1, 795: 1, 497:
1, 2790: 1, 1180: 1, 36: 1, 10: 1, 1081: 1, 4281: 1, 4081: 1, 2219: 1, 486: 1, 167:
1, 3112: 1, 453: 1, 4985: 1, 4095: 1, 4398: 1, 2093: 1, 435: 1, 11: 1, 4974: 1, 2472:
1, 4888: 1, 4545: 1, 2947: 1, 3746: 1, 1470: 1, 2578: 1, 3507: 1, 2538: 1, 3893: 1,
3081: 1, 1023: 1, 950: 1, 612: 1, 164: 1, 4470: 1, 2358: 1, 82: 1, 2800: 1, 63: 1,
2791: 1, 2763: 1, 509: 1, 3445: 1, 1815: 1, 3408: 1, 752: 1, 1745: 1, 1841: 1, 4641:
1, 4209: 1, 1948: 1, 26: 1, 1809: 1, 1750: 1, 1531: 1, 3220: 1, 2038: 1, 4540: 1, 102:
1, 4257: 1, 2741: 1, 3373: 1, 1493: 1, 623: 1, 2889: 1, 3652: 1, 19: 1, 8: 1, 3285:
1, 1922: 1, 1675: 1, 3014: 1, 667: 1, 4740: 1, 2131: 1, 3823: 1, 3718: 1, 1849: 1,
3263: 1, 3776: 1, 4161: 1, 3778: 1, 2185: 1, 18: 1, 3993: 1, 871: 1, 1723: 1, 1276:
1, 2752: 1, 3129: 1, 963: 1, 3933: 1, 4244: 1, 1313: 1, 2690: 1, 3724: 1, 3170: 1,
672: 1})
```