ESC472 Capstone - PPD6

Efficient Document Retrieval Through Topic Modelling

Group 1

Daniel Pinheiro Leal (1003635248), Jeremy Hunt (1002969120),

Kamran Ramji (1002934284), Saad Hussain (1003444276)

Client: Infera AI

Division of Engineering Science

University of Toronto

Canada

April 29, 2021

Contents

List of Figures

List of Tables

1 Executive Summary

This report outlines the design process of a software tool designed for the company Infera AI which provides search functionality for local PDF documents, presented in a browser UI. Our final product is primarily meant as a proof-of-concept. This report walks through our team's end-to-end design process, considerations, and lessons learned while developing the proof-of-concept, which will be useful to future engineers who are interested in picking up this work.

The product's target market is engineering design companies that have large corpora of technical documents, possibly containing internal documents, which are difficult to search through without prior knowledge of a document's contents. The product provides functionality to categorize documents into topics and allows searching on document title, author, or topic, in addition to visualizations of how closely related document topics are, through an interactive UI in the browser. Thus, users can search for topics they are interested in and explore the space of such documents in their custom corpus. This information can be used to inform design decisions with the use of internal documentation that may otherwise be difficult to find. Further project scoping and framing is presented in the report and key stakeholders are identified.

In learning the topics of documents and categorizing them, our product uses popular topic-modelling techniques which analyze word frequencies in documents and derive relationships between documents based on similarities in the analysis. The intuition is that words that are frequently present together are likely related, and that these relationships can be put under a "topic". For our proof-of-concept, this is the primary topic-modelling technique we investigated, but we outline other (potentially more involved) possible approaches for future extensions of the product.

We visualize the space of document topics by plotting documents as points on a graph, such that documents that are closer would be more similar in their topics. This allows visible clusters of documents, indicating documents of a related topic. If users find a document of interest, they can explore nearby documents in the visualization to find other documents that can provide further insight in the topic of interest. The visualizations will be shown when opening a particular document's page in the UI.

Also in this report, we discuss in detail the high-level functionality of our system as a whole as well as break it down into its individual subsystems. We define goals, metrics, and criteria that a successful version of our product must satisfy so that the product's performance can be assessed. We then discuss some design considerations we looked into before settling on our final subsystem

design decisions. Due to time constraints, these design considerations were not exhaustive, but provide insight on many baseline considerations. In addition, we provide a section for potential future considerations that we were unable to test but may provide merit if looked into further.

Overall, we demonstrate the feasibility and potential use of our product through a proof-of-concept. We highlight some design considerations and methods used to assess the design, and provide direction for future work on this product.

The code for the product is made available through a GitHub repository `https://github.com/DanielPinheiroLeal/InferaCapstone` [1] and a quickstart guide is provided in Section 4.2.

2 Introduction

2.1 Document Outline

This section presents an outline of the content and structure of this design document.

Section 2 (Introduction) presents a problem statement and motivates the need for the product described in the rest of the document. It also discusses the multiple stakeholders of this project, and the project's potential environmental and social impacts.

Section 3 (Background) provides background information on technical terminology and concepts present in the report.

Section 4 (Function) provides a high-level description of the product and the functions it provides, breaks down the design into its three subsystems, and provides design specifications. It also includes a "Quickstart Guide" section with specific details and instructions on how to install and how to use the product.

Section 5 (Performance) presents the design goals, metrics, and constraints that the product must meet and against which the design is later evaluated.

Section 6 (Subsystem Design Considerations) presents design considerations and explorations made in arriving at the final design solution for each subsystem, as well as potential future considerations that could not be explored at this stage. It also presents reference designs relevant for this project.

Section 7 (Subsystem Design Specifics) presents lower-level implementation details of the design, prototypes, and presents discussions on the verification and validation of each of the subsystems, as well as of the integrated design solution.

Section 8 (Resources) shows project management and planning considerations, such as task allocation, time and risk planning.

Section 9 (Structure) introduces the team members and the team charter.

Section 10 (Dynamics) considers stakeholder interactions and communication, and other sources of information for the team.

2.2 Problem Statement

Many engineering companies have large databases of technical documents, often written by their employees. These documents oftentimes do not have much metadata attached to them, which prevents indexing and searching on them. This, in turn, means that the documents are left largely unused and do not inform design decisions, owing to the difficulty of searching for relevant papers.

For example, the NASA Technical Reports Server hosts the following scanned PDF from 1967 with minimal metadata: `https://ntrs.nasa.gov/citations/19690075962`. Only the title, author, publication date, and an incorrect document type are given - at a less sophisticated engineering company, one might expect even less available metadata. The document content could still be valuable to NASA engineers but there would be no way for them to find this document and its relevancy without having prior knowledge of its title and contents.

Infera AI is a new startup that aims to provide data-driven software tools to engineering design companies, such as tooling for probabilistic forecasting of system performance, fault diagnosis and predictive maintenance, etc. In this capstone, our team is working alongside them to help create a new tool to solve the aforementioned document-searching problem. The end product would take the form of a web app which can connect to a large database of technical documents, extract relevant information from documents for topic modelling, find relevancy between documents, and allow for documents to then be searched by their topics in a UI. We also aim to incorporate a means of visualizing the relatedness of documents to each other based on their topics.

The creation of such a tool would allow for engineers to search for topics relevant to projects they are working on and find papers related to their topic of interest. These papers would no longer go unused and would help in informing design decisions for their work.

2.3 Process and Product Dimensions

2.3.1 Mode of Operation

This project is client-based. The client is Infera AI, a startup that develops software to assist decision-making to allow engineers to make optimal and robust design decisions. Our market is the space of document search tools. From looking at other examples of visualization tools for technical documents, there seem to be a limited number of tools in this area, particularly tools that can be used on any corpus (i.e. with no or little metadata), and that provide good visual search features. Our contribution to this space is thus focused on leveraging technologies that can be used for general corpus (i.e. without human created metadata), and providing high-quality visualization. Our contribution to this market will also have value to the client, as it will serve as a proof of concept for potential future products.

2.3.2 Nature of Product

The product being developed is a software application. As our product is focused on providing search tools for documents without human-generated labels, we will leverage machine learning tools to generate those labels using natural language processing. At a high level, our product is therefore made up of a database system, a front-end application (two necessary ingredients in web-app designs), and a learning model to train off the stored documents and provide topic data to the front-end.

2.3.3 Nature of Knowledge Leveraged

The knowledge needed for this project relates to building web applications, and creating a learning model to model topics of documents. Creating a web application requires good system design and access to effective hardware to ensure a smooth experience. Human interaction design is also required for the front-end application. Creating a learning model requires effective design and testing in order to produce an accurate model that helps users find information they are interested in.

2.3.4 Nature of Contribution

The contribution we hope to make to the market is an application domain expansion. All of the systems being built are based on existing technologies (i.e. NLP models, databases, web-app design), but the area in which the application is being developed is not very mature, so the project aims to expand it.

2.4 Stakeholders

2.4.1 Infera AI

Infera AI is a new startup that aims to provide data-driven software tools to engineering design companies, such as tooling for probabilistic forecasting of system performance, fault diagnosis and predictive maintenance, etc. This capstone project involves creating an automatic topic modelling tool for technical document search & retrieval which Infera can then provide to engineering companies, as part of its overall company mission. The end product is not a core feature that Infera AI offers and thus should emphasize modularity to facilitate Infera incorporating it as a fully featured service in the future. This product is a proof-of-concept for Infera, to allow them to see if this is an avenue worth pursuing in the future. Therefore, our design should be centered on demonstrating the advantages and potential drawbacks of using visualization, NLP and topic modelling techniques to search troves of technical documents.

2.4.2 Engineering Design Companies

Our product's target audience are engineering design companies (i.e. potential clients of Infera AI). Oftentimes, these companies have large corpora of technical documents which currently go unused instead of informing decisions on future projects because they are nigh impossible to search through. Our client provided the following example to demonstrate the need for this product: an engineer is more likely to turn to Google than to search through their colleague's (more relevant) work. Oftentimes, technical reports even from sophisticated companies and agencies, such as the 1967 PDF hosted on NASA's Technical Reports Server previously mentioned in the Problem Statement section, contain little to no correct metadata which prevents being able to search for it, wasting its potential value.

The primary purpose of our product is to help engineering companies make better use of their internal document databases along with the relevant engineering literature. We aim to create a product that can scan through large quantities of technical documents with little metadata and extract the key ideas and topics from them. This would allow documents to be indexed and searched, which in turn would allow engineers to conveniently view all their peers' work for their topic of interest. A unique feature that we aim to provide on top of this is the visualization of connections between documents. This will allow users to see documents related to the topic/document that they are currently interested in, in order to provide an avenue for further discovery of the topic.

2.4.3 Document Authors

A secondary stakeholder are the authors of the technical documents themselves. These may be past/present engineers working at the company of interest or others in academia. Our product can increase the visibility of many academic papers and technical documents for various companies, giving more traction to the paper and its author(s).

An interview with a graduate student, representative of this stakeholder group, has been conducted by the team and the logs from it can be seen in Appendix A.2. From this interview it was possible to gather that, given the graduate student's past experiences with searching for academic and technical papers, it is hard to find good results when searching based on a general topic, as opposed to searching for a specific paper. We believe that our design, by providing a topic modelling-based search-by-topic feature, can enhance the discoverability of academic and technical papers, thus providing benefits both for the users performing the search as well as for the authors of documents.

2.5 Environmental and Societal Impacts

The intent of this project is to make researching and investigating new topics easier using topic modelling and visualization. The impact this will have on society is analogous to the impact of larger, general search engines, if on a smaller scale. However, the creation of the "search index" in this case is significantly more computationally expensive per amount of data (and number of searches), due to the customized nature of the model (i.e. it models each specific trove of documents in a standalone manner), so the environmental impact of the energy consumption for these searches will be greater, proportionally. This product will also have an effect on what technical documents and academic papers are seen more often. It is unclear what the overall effect of this will be at this moment.

There is also the question of intellectual property. Our model will extract features from documents written by other people, and it might be ambiguous to whom the intellectual property of these extracted features will belong. For now, the feature extraction is completely standalone, so a company will only be running this program on documents to which they own the IP (or have some rights to access), and therefore the assumption would be that they would own any features extracted (since these are extracted entirely from the company's own documents). However, in the future, it may be valuable to extract features in a more cumulative fashion (i.e. use the model learned at Company A to boost the performance of the model used at Company B), at which point this question will become more pertinent.

3 Background

This section aims to provide technical background on concepts, technologies, and tools that are relevant and directly related to the presented product.

3.1 NeurIPS Dataset

NeurIPS [2] is an influential conference on neural information processing systems and machine learning, held every December. A dataset provided by the project's client and containing papers presented at NeurIPS throughout many years was used as the main reference and testing benchmark during the design of the product described in this document. It contains approximately 11,500 research papers published between the years of 1987 and 2020, and its size is approximately 15 GB. From now on, this dataset will be referred to as the "NeurIPS dataset".

3.2 Database Technologies

The relational database model, originally introduced by E.F. Codd [3] in 1970, consists of individual data entries structured as tuples combined to form relations (tables). Since its introduction, the relational database paradigm has become pervasive and it has been the basis for many of the most commonly used database systems. The Structured Query Language (SQL) is a domain-specific language that provides an interface for users to manage, insert, and query data from a relational database. SQL follows an ISO technical standard, the most recent of which has been published in 2016 [4]. Implementations of this standard from multiple vendors exist, such as OracleDB, MySQL, and PostgreSQL.

However, recent applications driven by big-data and machine learning, have demanded alternative storage structures for increased performance in specific workload scenarios. This demand gave rise to new paradigms, referred to as NoSQL, which provide flexible, at times schema-less, data models [5].

An example of NoSQL databases are graph databases, which are employed in the product described in this document. In graph databases, each data entry is stored as a graph node instead of a tuple as in relational databases. Relationships between graph nodes can be established and built-in to the database, enabling easy and fast querying of nodes and their related peers.

3.3 Natural Language Processing Models

In order to provide the reader with an understanding of the context of the rest of the work in this project, a background in fundamental natural language processing techniques will be necessary. This subsection will begin with a discussion of principal component analysis (PCA), a key tool from linear algebra that underlies many of the subsequent algorithms that will be discussed. Then, latent semantic indexing (LSI), a popular technique for information retrieval problems that leverages PCA on document-term frequency matrices, will be introduced. Two non-linear transformations of the document-term matrix, the tf-idf model proposed in [6] and the log-entropy evaluated in [7], will then be discussed. Finally, two clustering techniques, Latent Dirichlet Allocation and k-means, will be compared and contrasted, in order to provide the reader with an understanding of the two methods.

Beginning with principal component analysis (PCA), consider a matrix $\mathbf{X} \in \mathbf{R}^{\mathbf{n}\times\mathbf{k}}$, where n is the number of repetitions of an experiment, and k is the number of measurements produced by any one such repetition. In effect, this matrix $\mathbf{X}$ can be thought of as the result of "stacking" the measurements from an experiment into a row vector of length k, and "stacking" the numerous row vectors produced by each repetition of the experiment vertically into a matrix of dimensions $n \times k$. The measurements (which form the columns of the matrix) are often referred to as "features" or "terms", and the repetitions (which form the rows of the matrix) as "samples" or "documents".

PCA is a method to solve the problem of shortening the row vectors of length k, which form the rows of $\mathbf{X}$, into vectors of length p, while minimizing information lost. The intuition here is that by reducing the number of features, while preserving information, patterns in the data will become more evident. There are two common ways this problem can be formalized: preserving the linear components with the highest variance or minimizing reconstruction loss. It is an interesting mathematical fact that these two objectives are equivalent, and so the reconstruction loss objective is provided below:

$$\mathbf{V}^* = \underset{\mathbf{V}^\top\mathbf{V}:\,\mathbf{I}}{\arg\min} \sum_{i=1}^{N} ||x^i - x^i\mathbf{V}\mathbf{V}^\top||^2 \tag{1}$$

Thus, the goal of PCA is to find the matrix $\mathbf{V} \in \mathbf{R}^{k\times p}$, such that when the original matrix $\mathbf{X}$ is projected into the lower-dimensional space by applying the linear transformation $z^i = x^i\mathbf{V} \in \mathbf{R}^{1\times p}$ to the rows of $\mathbf{X}$, the reconstructed vectors $\hat{x}^i = z^i\mathbf{V}^\top$ are as "close" as possible, in the Euclidean sense, to the original rows of $\mathbf{X}$. It can be shown that the columns of $\mathbf{V}$ are in fact the eigenvectors of the sample covariance matrix $\frac{1}{n}\mathbf{X}^\top\mathbf{X}$. These eigenvectors can be efficiently computed using the singular value decomposition (SVD) of X, by recognizing that the right-singular vectors of the matrix $\mathbf{X}$ are, in fact, eigenvectors of the sample covariance matrix:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top \tag{2}$$

$$\mathbf{X}^\top\mathbf{X} = \mathbf{W}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top = \mathbf{W}\mathbf{\Sigma}^2\mathbf{W}^\top \tag{3}$$

PCA becomes useful for natural language processing when combined with another tool, the document-term matrix. Constructing the document-term matrix for a collection of n documents, with a vocabulary of size k, is simple. Each row in the document-term matrix corresponds to a document (this can be a book, an article, or a sentence), and each column corresponds to an element of the vocabulary. The value in each entry in the document-term matrix is the number of occurrences of the term corresponding to that entry's column, in the document corresponding to that entry's row. A non-linear transformation is applied to the raw term counts, either a simple normalization of the columns or rows, or a more complex transform, such as tf-idf or log-entropy. Finally, PCA is performed on the transformed matrix, reducing the number of features from k (the vocabulary size, which is the number of unique words in the document), to p, which is a predetermined constant. The ideal value for p, which is referred to as the "size of the latent space", must be determined by experimentation, and depends on a number of factors, such as the nature of the collection of documents, the desired application, and the quality of the input data. The combination of document-term matrix construction (often called a "bag-of-words"), non-linear transformation, and principal component analysis constitutes latent semantic indexing (LSI).

There are two common non-linear transformation of which the reader should be aware. The term-frequency, inverse document frequency (tf-idf) method transforms the raw term counts by applying the following transformation:

$$\begin{aligned}
local(i,j) &= \frac{X_{i,j}}{\sum_{j=0}^{k} X_{i,j}} \\
global(\cdot,j) &= \log\frac{n}{|d\colon X(d,j) \neq 0|} \\
tfidf(i,j) &= local(i,j) * global(\cdot,j)
\end{aligned}$$

This transformation has two components. First, the term-frequency component, which normalizes the raw term counts into a proportion of the total number of words in the document. The second is the inverse document frequency, which gives greater weight to words which appear in fewer documents.

The log-entropy transformation has a similar structure, with two factors, one of which is a transformation of the raw frequency counts, the second of which applies a factor which is a heuristic for the relevance of that term relative to others [8]:

$$\begin{aligned}
&\mathit{local}(i,j) = \log(X_{i,j}+1)\\
&P(i,j) = \frac{X_{i,j}}{\sum_{i=0}^{n} X_{i,j}}\\
&\mathit{global}(\cdot,j) = 1 + \frac{\sum_{i=0}^{n} P(i,j)\log(P(i,j))}{\log(n+1)}\\
&\mathit{logentropy} = \mathit{local}(i,j) * \mathit{global}(\cdot,j)
\end{aligned}$$

The global weight used in the log-entropy transformation is a “better” approximation of the relative importance of a term, because it uses the entropy of the distribution of that term over the documents, rather than a simple binary test. Therefore, whereas a term appearing many times in a small number of the documents in the collection, and appearing with very low frequency in all of the other documents in the collection, would be assigned a low importance weight by the td-idf method (because the term appears in every document), the log-entropy formulation would capture the low entropy of the distribution overall.

Latent semantic indexing looks to create a low-dimensional latent space in which common vector operations (addition, subtraction, and cosine distance) have semantic meaning. This allows search queries, in the case of information retrieval, to be transformed into co-ordinates in a vector space into which the documents are also mapped. Unsupervised clustering algorithms, in contrast, look to identify clusters of documents, and do not provide a vector space with semantic structure. Latent dirichlet allocation (LDA) is one such unsupervised clustering algorithm. In order to understand the advantages of LDA, it is useful to draw comparisons to k-means, a simple unsupervised clustering algorithm which provides a useful conceptual framework. While there are many variants of the algorithm, the following provides the necessary starting point from which to discuss LDA:

1. Randomly select k datapoints for the dataset to initialize the k-clusters
2. Assignment step: assign each data point in the set to the cluster closest to it (in the

Euclidean sense)

3. Update step: take the mean (i.e. find the centroid) of each cluster, and assign the new centroid to that cluster
4. Repeat the assignment and update steps until convergence

The k-means algorithm, upon convergence, thus assigns every document in the collection to a cluster (these clusters are referred to as "topics" in the natural language processing literature). The disadvantage of the k-means approach is that a single cluster is assigned to each document. LDA solves this problem by finding a distribution over the topics for each document. This is achieved by modelling each word as an observation of the underlying topic distribution, and allocating the topic distribution so as to maximize the posterior probability of the collection of documents.

3.4 Web Technologies/Standards

3.4.1 Client-Server Model

The client-server model [9] is a distributed application framework that defines a server and clients where multiple clients can make requests to a server, the server listens to the requests, and gives an appropriate response. The client and server can either communicate on the same hardware or, commonly, across a computer network/Internet. Clients and servers communicate in a request-response fashion, adhering to particular communication protocols. For example, the common communication protocol for web applications is TCP/IP (see following sections for more information). Some of the benefits of client-server models are as follows [9]:

- Single server hosts all data for easy facilitation of authentication, authorization, protection of data, etc
- Additional servers and intermediaries can be added without significant interruption
- Data can be accessed efficiently without requiring clients and server to be in close proximity
- All nodes in a client-server model are independent and allow easy upgrades/replacement
- Data transferred is platform-agnostic

Clients and servers come in different forms. The most common for web applications is to have a *thin client*, one that relies heavily on the resources of the host computer and the server, along with an application server, database server, and web server. *Application servers* host web applications and perform all necessary data processing, isolated from the client. *Database* *servers* provide access to database(s) for programs that ingest well-organized data. *Web servers* host web pages that an application can run on, serving as the backbone of the World Wide Web. A web app developer mainly works on the application server and its interactions with the client, as well as the database server and retrieving/processing/updating data.

3.4.2 TCP/IP and HTTP

TCP/IP is the de-facto standard internet protocol suite which defines a set of communication protocols for machines to communicate information with each other across the internet. The TCP/IP protocol suite communicates data as "packets" which consist of multiple layers:

1. Application layer: user interface to use the services of the network and application
2. Transport layer: end to end communication and delivery of data
3. Network layer: logical transmission of data over the entire network
4. Link layer: physical transmission of data across hardware

The TCP/IP protocol suite consists of many potential protocols at each of the four layers (see Figure 1) and is named after two of the most common protocols - TCP at the transport layer and IP at the network layer. TCP and IP are often used in conjunction with the HTTP protocol at the application layer for the web. [10]

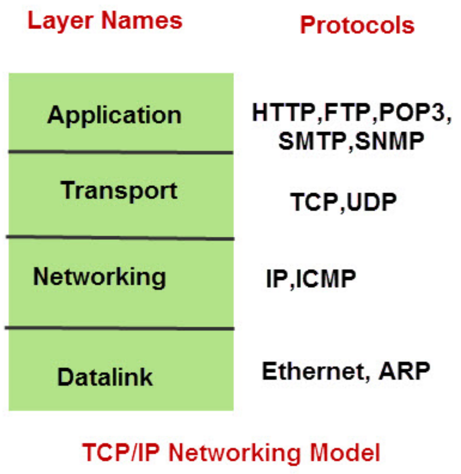


Figure 1: TCP/IP internet protocol suite

IP (Internet Protocol) delivers packets of data from the source host (machine) to the destination host (machine) by looking at the IP address in the packet header. There are two versions: IPv4 (more common) and IPv6 (growing in popularity as IPv4 becomes obsolete). [11]

TCP (Transmission Control Protocol) provides reliable and error-free communication between end systems. It performs sequencing and segmentation of data, requires acknowledgement of segments to ensure no lost data, provides control flow mechanisms, etc. This robustness comes at the cost of higher overhead than other methods such as UDP, which does not guarantee reliable and error-free communication. [11]

HTTP (HyperText Transfer Protocol) is used across the World Wide Web to manage communication between web browsers and servers. It is a stateless protocol based on requests and responses, where request headers contain all information needed to construct a response. It provides various request methods, such as `GET` (retrieve data), `POST` (create data), `PUT` (update data), `DELETE` (delete data), etc. [12] HTTP responses come with status codes; responses with status codes in the range 100-199 are informational, 200-299 are successful, 300-399 are redirects, 400-499 are client errors, and 500-599 are server errors [13]. Its request and response headers contain other information to similarly contain all necessary information and provide seamless interaction between clients and servers.

3.4.3 ReST API

An API (application programming interface) is a set of definitions which governs how two pieces of software can interact with each other. In the context of a webapp, the frontend interacts with the backend code through an API that serves particular requests at particular endpoints. A ReST (Representational State Transfer) API is one that conforms to a set of architectural constraints and has become a popular API design methodology. The rules for an API to be considered RESTful are [14] [15]:

1. Client-server architecture with requests managed through HTTP
2. Stateless client-server communication (each response is dependent only on its corresponding request, previous requests are unrelated)
3. Cacheable responses to prevent stale/inappropriate responses to further requests
4. Uniform interface so that information is transferred in a standard form. Requires:
   (a) Resources are identified in requests, e.g., using URIs. Resource is conceptually different from representation sent to client (e.g., PDF resource vs. JSON representation as response)
   (b) Resource manipulation by client via representation because it has enough information to do so
   (c) Self-descriptive messages include information to describe how to process the message

(d) Hypermedia is present for client to reach all actions that the server can provide

5. Layered system to organize server type (e.g., security vs. load balancer) which is invisible to client. Changes to layered system does not affect communication between client and server.
6. Optional: code on demand which allows the server to send executable code to the client to extend functionality

When requests are made to a RESTful API, it transfers a representation of the resource to the requester, typically as JSON (JavaScript Object Notation), as it is language-agnostic (despite its name) and readable by both humans and machines [14].

3.4.4 JSON

JSON is a lightweight format for storing and transporting data. JSON data types consist of [16]:

- Numbers: signed decimal numbers that may contain a fractional part and may use exponential notation
- Strings: sequence of unicode encoded characters delimited with double-quotation marks (“ ”)
- Booleans: `true` / `false`
- Arrays: ordered lists which contain elements of any type using square bracket notation with comma separated elements (`[..., ...]`)
- Objects: collection of key-value pairs where keys are strings and each key is unique within an object. Objects are delimited with curly brackets (`{}`), commas separate each pair, and colons separate keys from values
- Null: empty value as keyword `null`

Whitespace is ignored around or between syntactic elements, except within strings. An example JSON is provided below:

```
{
  "name": "John Smith",
  "alive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

4 Function

4.1 Product Description for End Users

As mentioned, our target audience is engineering companies that have many documents which they would like to search through in order to guide design decisions. These documents may include private, internal documents written by past/present employees.

Our end product takes the form of a web application that users can access through their browsers. A user can access the page and be greeted by a search box where they can search for documents by title, author, or topic. These documents come from a pre-uploaded source that is managed perhaps by a company's system administrator, which facilitates the inclusion of internal documents. When searching for a document, a list of the top results will be returned to the page. A user can select one of the search results to access more information about that document, including the full PDF as well as suggested related documents, along with a visualization of the relatedness of the suggested related papers to each other. If searching by title or author, documents that contain that search term as a substring for that metadata type will be returned to the page. For example, searching "latent" by title may return a document titled "Latent Semantic Analysis" (if it exists in the database). If searching by topic, documents that are related to the search query by topic will be returned. For example, searching "topic modelling" by topic may return a document on "Latent Semantic Analysis" as well (assuming that document is related to topic modelling).

Behind the scenes, our product will include functionality to load in a corpus of PDF documents and extract topic information and other available metadata for searching. These documents can simply be saved in the file system. Our document processing component will be able to be run with a simple script that takes in as an argument the directory which contains all documents to process. As long as the database software that we rely on is running (see the *Storage and Search* section), our machine learning models will automatically take care of the rest. Our model may take on the order of a few hours to process the documents and populate the database, depending on how many documents are provided; this process can easily be done overnight. This aspect may need to be managed by a company's system administrator or equivalent.

4.1.1 Topic Relatedness

A key aspect our product revolves around is the ability to find related papers to any particular search, as this allows someone to find documents that may assist their design decisions for a given project. Our model uses state of the art techniques in topic modelling such as latent semantic analysis [17] and latent dirichlet allocation [18] to extract topics from each document and give each document an embedding vector in latent space (a mathematical representation of the space of topics). The more similar documents are in latent space, the more similar (i.e., "related") their topics are. Similarity in latent space is calculated using the cosine similarity of the documents' embedding vectors.

A simplified but intuitive view of the notion of relatedness is that documents that are related in topic will likely have content that uses similar, unique wording for that topic. As a rudimentary example, documents that more frequently contain words such as "neural", "embedding", "vector", "machine", "learning", etc. are likely related to the field of machine learning. Depending on what other words they contain, e.g., "semantic", "topic", etc., further inferences on their topic can be drawn such as being related to natural language processing or topic modelling.

4.1.2 Caveats

There are a few caveats regarding our intended deliverable for this project. First, our deliverable is only targeted to run locally - additional work will be required to host the service online. We are also assuming title and author metadata are available from the papers - we will not be parsing files for these ourselves. However, it is important to note that neither of these two caveats impede the attainment of the central objective of the project, which is to explore the use of topic modelling in this information retrieval setting.

4.2 Quickstart Guide

This section describes how to run the end-to-end integrated system. The complete application is made available through a GitHub repository `https://github.com/DanielPinheiroLeal/InferaCapstone` [1].

4.2.1 Requirements

- Download code from aforementioned GitHub repo [1]
- Python 3.x [19]
- Nodejs [20]
- Neo4j
  - System requirements [21]
  - Neo4j server [22] [23]
  - Neo4j Graph Data Science library [24] [25]
- Python dependencies
  - `pip install gensim` [26]
  - `pip install tika` (requires Java 7+) [27]
  - `pip install flask` [28]
  - `pip install flask_cors` [29]
  - scikit-learn [30]
- Node dependencies
  - Run `npm install` in our code's `webapp/client/` directory to automatically install all node-related dependencies

4.2.2 How to Start

There are three pieces to run in the system: the database, the backend, and the frontend.

Following instructions from [23], run `./bin/neo4j console` in the directory you installed the neo4j server to in order to start the database. The database must be running in order to start the backend.

The backend can be run with the base command `python api.py` in our code's `webapp/server/`

directory, but has many arguments (some of which are required) as follows:

Shortcut	Argument	Required	Default	Description
`-m`	`--model-path`	Yes	*n/a*	Path to directory containing saved topic modelling files
`-r`	`--db-uri`	No	"bolt://localhost:7687"	Database URI
`-u`	`--user`	No	"neo4j"	Database account username
`-p`	`--password`	No	"capstone"	Database account password
`-k`	`--neighbours`	No	25	Number of neighbours in knn graph
`-lsi`	`--numLSI`	No	10	Number of LSI dimensions
`-lda`	`--numLDA`	No	10	Number of LDA topics
`-pdf`	`--pdf-path`	No	""	Path to directory containing pdf files
`-text`	`--text-path`	No	""	Path to directory containing text files
`-t`	`--train`	No	False	Train the model from scratch. True/False
`-n`	`--nodes`	No	False	Rebuild database nodes. True/False
`-c`	`--convert`	No	False	Convert PDFs to text files
`-d`	`--debug`	No	False	Turn debug mode on or off. True/False

Table 1: Backend command line arguments for program start-up

For example, running the backend with all extra functionality (convert PDFs to text, train model, build nodes) can be done with:

```
python api.py -m "path/to/model/" -pdf "path/to/pdfs/" -text "path/to/
    text/" -t -n -c
```

To run the backend with minimal arguments (i.e., load saved model and use pre-built database):

```
python api.py -m "path/to/model/"
```

To start the frontend, simply run `npm start` in our code's `webapp/client/` directory.

Starting the frontend should automatically open a browser at `http://localhost:3000` (which interfaces with the backend at `http://localhost:5000` behind the scenes). The app is now ready for use!

4.2.3 How to Use

In the home page there is a top bar and a search bar with a word cloud. Clicking on "Document Explorer" on any page redirects to the home page. The user can type whatever they would like into the search bar. The user must also select a search mode. Author and Title modes will perform a sub-string search on those attributes within the database. A topic search will convert the submitted text to a vector and find the papers most similar to this vector. The results of a search will display in a list underneath the search bar and word cloud. Clicking on any of these will bring up a dedicated page to this article. This includes a PDF view of the paper, a visualization of the topics of the 100 most related papers, and an ordered list of the 100 most related papers from most related to least. A more detailed overview of all these is in the section "Subsystem Design Specifics".

4.3 Functional Decomposition

Our product is divided into three primary subsystems:

1. **Interaction and Visualization** — presents search results in an interactive manner along with visualization tools to explore a domain of interest
2. **Storage** — stores the database of all documents along with their metadata and contextual information (i.e., topic modelling data), ready for easy and quick retrieval
3. **Natural Language Processing Model** — computes document similarity and correlates user search queries on topic with relevant documents

These subsystems are briefly outlined in the following sections. Figure 2 shows a high level overview of the interaction between subsystems.

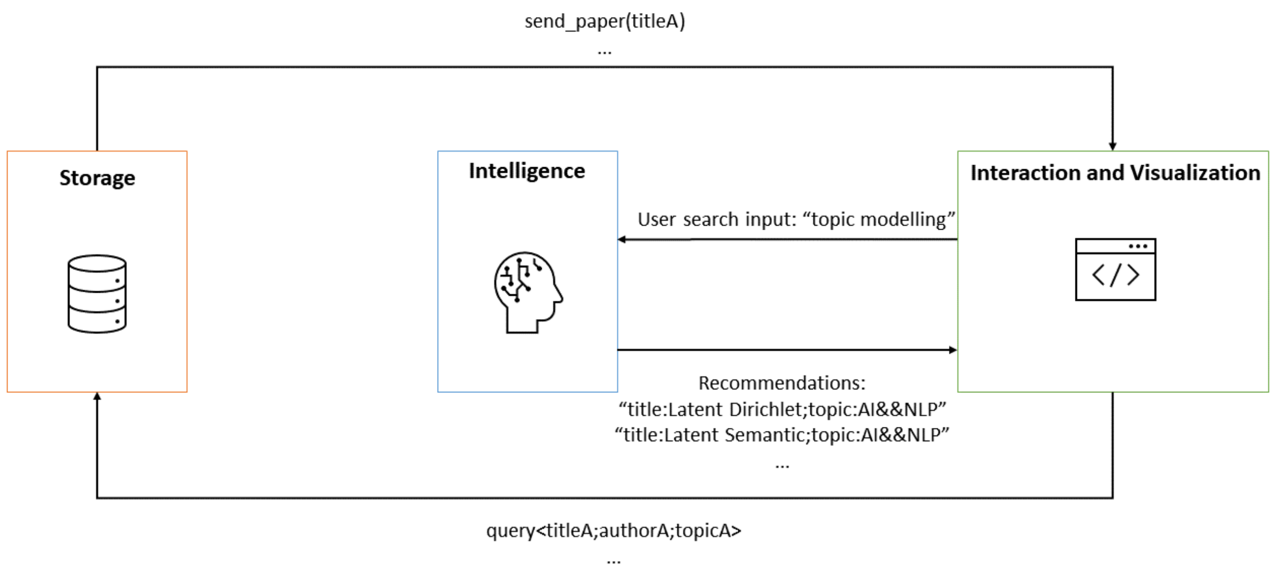

send_paper(titleA)
...
Storage
Intelligence
Interaction and Visualization
User search input: "topic modelling"
Recommendations:
"title:Latent Dirichlet;topic:AI&&NLP"
"title:Latent Semantic;topic:AI&&NLP"
...
query<titleA;authorA;topicA>
...

Figure 2: Interactions between system functions: Storage, Natural Language Processing Model, and Interaction & Visualization.

4.3.1 Interaction and Visualization

The Interaction and Visualization function is responsible for receiving commands from the user and displaying search and recommendation results in a visual and meaningful manner.

In terms of interaction, the system must allow the user to query for results and recommendations based on a document title, author, and topic using, for example, a search bar.

In terms of visualization, it must provide a good display from which the user can clearly understand and navigate the multiple recommendations that the system provides. The ICLR and NeurIPS conferences provide visualization tools [31], [32] where all conference papers (in the order of 1,000 papers) are shown in clusters, grouped based on keywords. These visualization tools can be used as a reference for the required function, but other visualization functions are also desired.

4.3.2 Storage

The Storage function collects the entire set of academic papers and serves requests for individual papers or groups of papers. It should be capable of storing and managing data sets with size on the order of 15 GB. The reason for the 15 GB baseline is that it is the approximate size of the NeurIPS dataset, which was provided by the project's client and is considered to represent well a collection of technical and research documents used internally in an engineering company. It consists of approximately 11,500 research papers in the field of machine learning, published in the NeurIPS conference between the years of 1987 and 2020.

One of the main aspects of the Storage subsystem is that it must serve requests quickly, thus allowing the entire system to interact with the user in a highly responsible manner.

In order to allow for fast servicing of requests, and also support meaningful queries based on the outputs from the intelligent recommendation system, the Storage system must have a carefully designed schema or data storage structure. Otherwise, given the large amount of data it has to store, it can potentially become a performance bottleneck.

4.3.3 Natural Language Processing Model: Making Recommendations

One of the defining features of the system is its ability to provide recommendations of papers related to user search inputs. To achieve this, an intelligent recommendation unit is utilized.

The recommendation unit interacts with the application's database and front-end by providing recommendations to the user based on given inputs in real time. It makes use of topic modelling to determine the topics that best describe each paper in the database. Topic modelling is an unsupervised machine learning technique, and Latent Dirichlet Allocation (LDA) [18] and Latent Semantic Analysis (LSA) [17] are the most import methods to implement it. Implementations of these algorithms such as GenSim [8] in Python and topicmodels in R [33] exist and are available for public use.

Furthermore, it must be able to look through the entire database to build its understanding of the correlation between papers in a reasonable amount of time, as updates to the database may happen frequently.

4.4 Product Specification

The interactions this service should provide are:

- *(S1): Intake a corpus of PDF documents*
- *(S2): Based on these documents, create a database of these papers with relevant topic modelling information*
- *(S3): Provide a web browser based user interface that provides search and visualization features to the user*
- *(S4): Allow users to search papers by paper title, paper author, and topic strings*

4.4.1 Machine Learning Model

- *(S5): Intake a corpus of PDF documents*
- *(S6): Based on the created model, interact with the database so that queries to the database show appropriate papers*
- *(S7): Process user search to convert a query string to a model format so that related papers can be found*

4.4.2 Database

- *(S8): Must store an entry for each paper with topic and available metadata information*
- *(S9): Topic information must be able to be used to determine top descriptive topics*
- *(S10): Topic information must be able to be used to determine a measure of relatedness between papers*

4.4.3 Document Storage

- *(S11): Must provide PDF documents for use in the machine learning model*
- *(S12): Must be accessible by the client so that when the PDF URL is fed to an in-browser PDF viewer, the articles can be viewed in-app*

4.4.4 Browser User Interface

- *(S13): Must allow user to pass single string of search arguments to database*
- *(S14): Must display set of results returned by database in response to search string*
- *(S15): Must be able to intake topic string and provide visualization of multiple articles*
- *(S16): Must provide an article page where the PDF can be viewed in the browser, most relevant topics are presented, and a visualization including the most related articles are shown in relation to the main article*

5 Performance

5.1 Performance Goals and Metrics

Based on communication with the clients at Infera AI, performance goals were established for the design. Note that each of the goals mentioned below come from needs directly expressed by the client, as reported in Appendix A.1. The relationship between each goal and the subsystems it pertains to is also discussed, providing a connection between the goal and the system functions.

- *(G1): Total query response time of 0.5 s.* This goal impacts all three subsystems: database, machine learning model, and frontend. It provides a bound on the time the database takes to search for a specific paper and papers related to it, the time the model takes to process and encode a topic input by the user and the time the frontend takes to communicate with the backend and present query results to the users.
- *(G2): Model training time of at most 12 hours.* This goal applies specifically to the machine learning subsystem. As mentioned in the e-mail from Infera AI in Appendix A.1, this will allow the model to be re-trained over night, adjusting and updating to any changes to the data set.
- *(G3): Better-than polynomial search algorithm for the PDFs.* This goal applies specifically to the database subsystem. Achieving it will enable the system to scale well to increasing

data set sizes.

- *(G4): The whole system should be runnable on a server.* This goal applies to the system as a whole. Even though initial prototypes and designs may be developed to run locally on a PC for testing and demonstration purposes, the system should be easily transferable to a distributed environment, as the end goal of the clients is for a system that runs on a server.

- *(G5): The system should improve the efficiency for users to search through their document data set.* One of the primary goals of the design is to enable efficient document retrieval, being this a goal that applies to the system as a whole. To validate this goal was met, experiments can be set up where users will look for specific papers starting both from specific and well-defined information, and from general topic information, and rate their search experiences when using our tool versus other available solutions, one being manually searching through the document database.

Furthermore, to evaluate the performance of each subsystem, subsystem-specific metrics will be used. The metrics will aid in evaluating the performance of the system as well as inform design decisions and allow the team to differentiate between potential design solutions.

- *(M1): Average and standard deviation of database lookup time.* This measurement will help guide database design, specifically by ensuring that the time bound set by goal (G1) is not violated.

- *(M2): Model accuracy.* For the machine learning model, standard evaluation methods for topic modelling, such as the probability of held-out documents given a trained model [34], will be used to quantify the accuracy of the model.

- *(M3): Model response times.* Also related to goal (G1), this measurement will help ensure that the machine learning model processes inputs fast enough for the latency goal of the system to be met.

- *(M4): Interface responsiveness.* As mentioned by Infera AI in Appendix A.1, an extremely responsive interface is sought. To quantify its responsiveness, the response times for different user actions (e.g. query papers by author, query papers by topic, etc.) will be measured.

- *(M5): User interface and data visualization quality.* A big emphasis has been put by the client on the visualization aspect of the tool. Feedback from the client as well as from proxies of end-users will be sought to evaluate the quality of the proposed visualization functionalities.

5.2 Design Constraints

Key design constraints determined based on communications with the client are outlined below. The constraints are again based on explicit requests from the client, which can be seen in Appendix A.1.

- *(C1): The user must be able to search for particular papers* . Beyond recommendation and visualization, the ability to search for particular papers is a function to be provided by the design. For this, a search bar or other search mechanism is required.
- *(C2): Multiple query types must be supported.* The client has expressed the desire to allow users to query for papers based on, at least, paper author, title, and topic, and thus all of these query formats must be supported by the design.
- *(C3): The design must be able to support data sets over 15 GB.* In our communication, the client has mentioned that a data set size over 15 GB can be assumed. The size of the NeurIPS data set, which is being used as an initial benchmark, is in the 15 GB range.
- *(C4) Cross-platform compatibility*: The system must support both Unix-based and Windows platforms

6 Subsystem Design Considerations

As mentioned in the Function section, the three main sub-functions of our product are storage, interaction & visualization, and natural language processing. In this section we outline higher-level considerations on moving from our specification to specific designs to meet our product's requirements.

6.1 Interaction and Visualization

We will be using React for the front-end of our web app, as it is a common standard for modern interactive web design. We will be using Python's Flask framework for the back-end, since it enables easy interaction between the Neo4j database and our intelligent modelling code.

The design of the user experience is driven by two high-level interactions. The first is the initial search for information, and the second is guiding the direction of a search in a particular direction after more in-depth information is found.

6.1.1 Search Start

For the initial user introduction to data, typical search engines are the current standard, therefore this user experience will be emulated as one way to find documents based on only user input. Given that visualizations are definitely going to be a part of this application, this will be leveraged in this use case as well. A separate view will also be able to intake user input and provide some visual representation of the most relevant results. A mock-up of these views are below. Search suggestions are very useful for searching [35], and given LDA modelling is being done, this word topics can be leveraged as suggestions as part of the search bar.

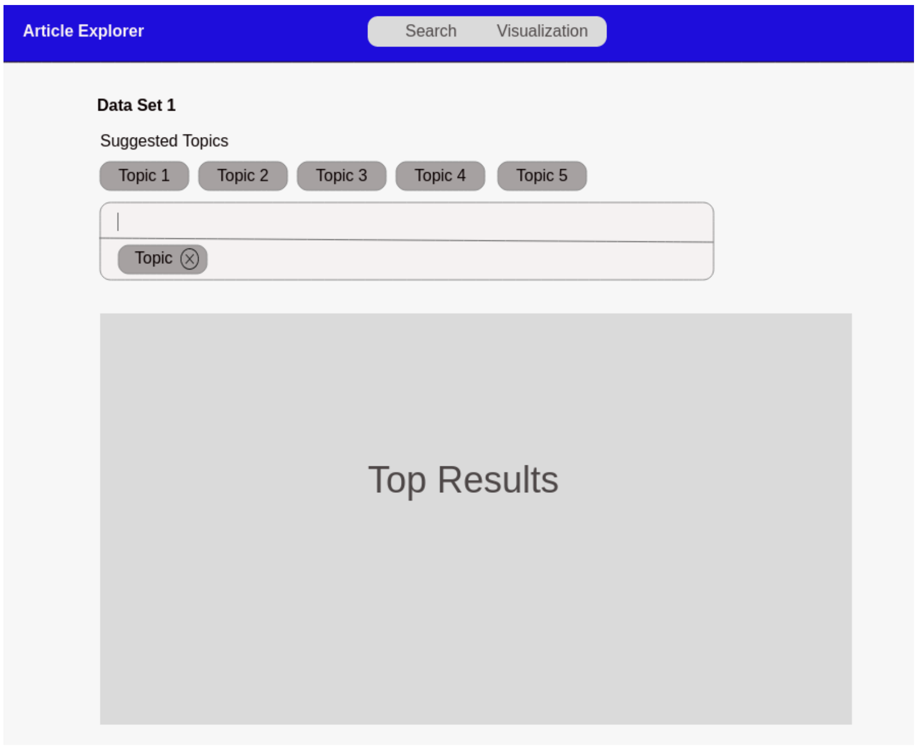


Figure 3: Search Page mock-up

6.1.2 Redirecting Search

The second interaction, guiding a search, is focused on the article view page. The idea is that after the user has found a satisfactory set of results, they will explore a particular result they find interesting. Once the user has determined their opinion of the result they have found, they will either be able to start from scratch, go back to the previous results view, or find results similar to the current one, based on the most relevant topics. Finally the user will be able to view the cluster to which the paper belongs and how it relates to similar papers in its own visualization.

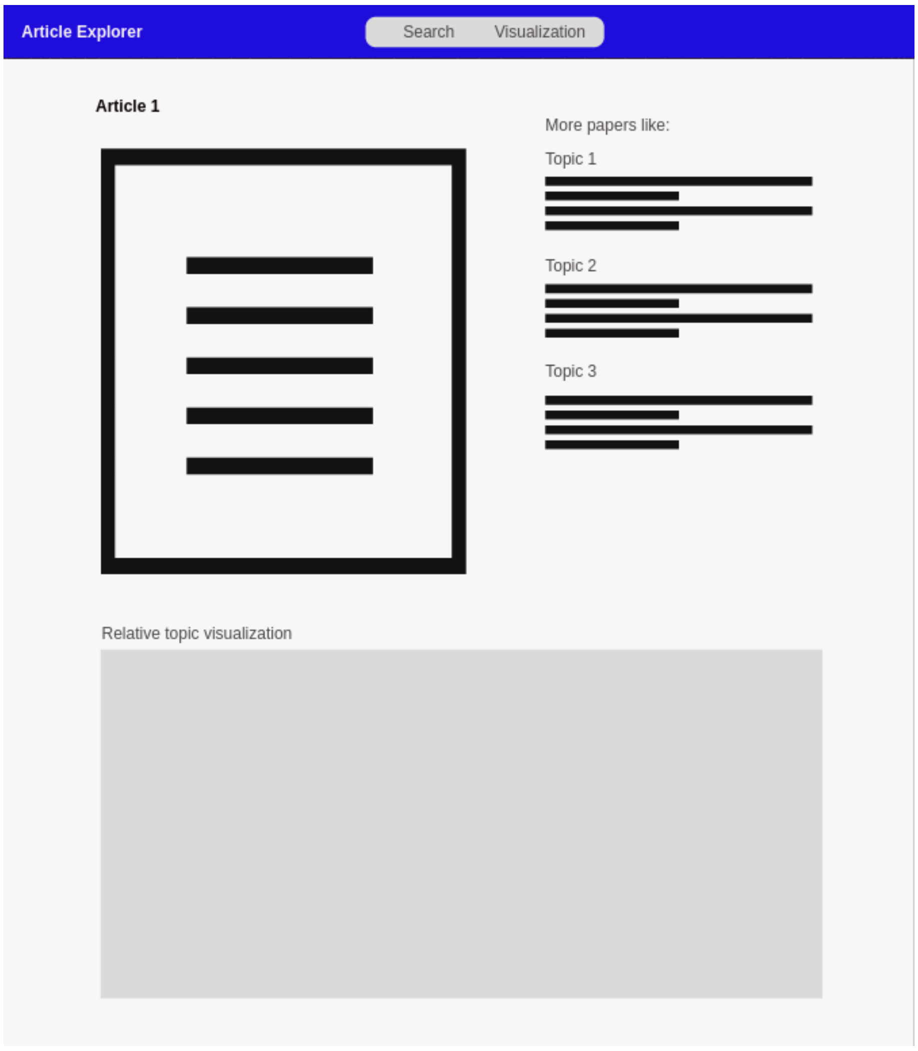


Figure 4: Article Page mock-up

6.1.3 Backend Technologies

“A framework’s main objective is to automate the overhead correlated with software development activities.” [36] We want to choose a backend framework that is time-saving, scalable, robust.

Expressjs is a very popular choice of backend framework used for building APIs and web applications as it allows the use of a single programming language for the frontend and backend (note we are using Reactjs for the frontend), has an easy learning curve, provides high performance with Google’s V8 Javascript engine [36]. It is used in sites such as MySpace and Geeklist [36].

Ruby on rails is another popular framework with many ready-made modules and plugins for time-efficient implementation of many features. It has many features for testing to ensure bug-free code, promotes the use of web standards and software engineering patterns for qulaity development, and is very scalable for a high volume of customers [36]. It is used in sites such as Shopify and Github [36].

Django is another strong choice as it is written in Python which interfaces well with our other code (database driver) since it is also written in Python. It is feature-rich, has optimal security, high scalability, and is versatile [36]. It is used in sites such as Instagram, Pinterest, and Coursera [36].

While these frameworks are all viable choices, we ultimately chose to go with the Flask framework. Flask uses Python, which interfaces very well with our neo4j database driver, also written in Python. Querying the database from the backend becomes trivial with such an approach. In addition, the model for topic-modelling is written in Python as well; a developer/team that works on this project does not require multiple language skillsets. Flask is a very lightweight backend framework that gives a lot of control to the developer and does not have a steep learning curve. As such, it is perfect for this proof-of-concept product and can be scaled up as appropriate. Flask is used in notable cases such as RedHat and Reddit [36].

6.2 Database: Storage and Search

While exploring and looking for the appropriate database solution, we considered four main criteria:

1. Structure – Our database will need to store metadata for each paper such as title, author, etc. In addition, it will need to store information (likely in the form of embedding vectors) on the topic in its topic space.
2. Size – We will need to store data on the order of 15 GB, so smaller database systems are appropriate.
3. Speed and Scale – Our database does not require fast write speeds, as the documents will be stored beforehand. Our database will, however, require fast read speeds as fast responses from our application will be critical for user experience. It also needs to scale well to a high number of entries.
4. Usability - The database driver should preferably be easy to implement, considering both the complexity of writing the necessary queries as well as the previous experience of the team.

The main types of databases we considered were:

- Relational – Stores data in classification tables; allows data to be identified by keys and then structured based on categories. Requires a fixed schema for each table. May work well for our documents which have multiple classification categories such as topic(s), date, author, title, etc.

- Document-oriented – Documents stored with varying fields for each document (unlike fixed schema for relational). Flexible and efficient as a search-based data store as they are typically optimized for data querying across multiple fields, sorting by relevance, etc.
- Graph – Stores papers as nodes and the relationships between them to any depth. May be useful when deriving relationships between the topics of academic papers.

Table 2 presents a decision matrix built considering the three potential database solutions and the four criteria.

Table 2: Database decision matrix comparing the three considered database solutions.

Criteria	Weight	Relational	Graph	Document
Structure	0.15	10	9	9
Size	0.1	10	10	10
Speed and Scale	0.6	5	10	6
Usability	0.15	8	9	5
Total Score		6.7	9.7	6.7

In terms of structure, relational databases received a grade of 10 due to their well-structured paradigm, which is able to accommodate all of the required properties for each paper entry. While graph and document databases are also able to do so, they offer unnecessary flexibility, as all paper entries have the same properties.

In terms of size, all database types can be made to accommodate the required size of 15 GB comfortably, and thus they all received a grade of 10.

In terms of speed and scale, relational databases do not rank well, as when querying for papers and their most related peers, relatedness metrics have to be computed for all papers in the database at query time. Graph databases, on the other hand, can pre-compute relatedness information at build time and store it built-in to the database, thus providing an optimal solution for our use case. The flexibility provided by document databases can aid in the query time computations, but it is still far from the built-in solution provided by graph databases.

In terms of usability, relational databases were the only ones the team had previous experience with, however the need to perform relatedness computation at query time harms makes implementing this solution harder. The team was new to graph databases, but given the favourable structure for the use-case, writing queries with them is expected to be much easier than with relational databases.

In the end, we chose to use a **graph database** since one of the primary functions of our product is to provide relatedness between a user's topic search and various papers. A graph database provides a convenient means to store the relations and similarity of topics, by constructing a graph on the topic embedding.

The particular graph database we are using is Neo4j [37]. Our Neo4j database has in-built capabilities to store relations between entries in the form of k-nearest neighbours. Papers can be queried by any key they are associated with such as author, title, or topic (written by the machine learning model described above). Direct matches as well as k-nearest neighbours can be returned.

6.3 Natural Language Processing Model

There are two important categories of tasks that the machine learning model will have to perform: training, and responding to real-time queries. Training includes all of the activities which can be performed before the user input is available. This includes: loading all stored PDF documents, converting them into text, and storing the plain text versions for use throughout; training the NLP models and storing the model parameters to disk; and extracting any corpus-level information that will be required by the other subsystem components at query-time. It is desirable for the model to front-load as much of the computation as possible, so that responses to queries can be provided quickly. However, some computation can only performed once user input is available. Most notably, this includes any processing of the user input.

The client has communicated that this project should be an experiment in using LDA and LSI as tools for information retrieval. However, there are alternative tools, of a similar level of complexity, which could be used. The first is pre-trained GloVe embeddings from [38], which essentially learns a vector representation for words by performing LSI on much larger corpuses, such as English Wikipedia or the publicly-accessible internet. It would be possible, then, to take the sum of the GloVe embeddings for all the words in a document (i.e. find the centroid in the embedding space), and use this centroid to quantify the relationships between documents, and between documents and queries. This approach will not tailor the learned subspace to the corpus, however, and because the corpus may focus on a particular subset of topics (within the larger space of topics which would be represented in English Wikipedia, for example), this may result in the entire corpus forming a single cluster, without being able to extract the nuance from within the corpus.

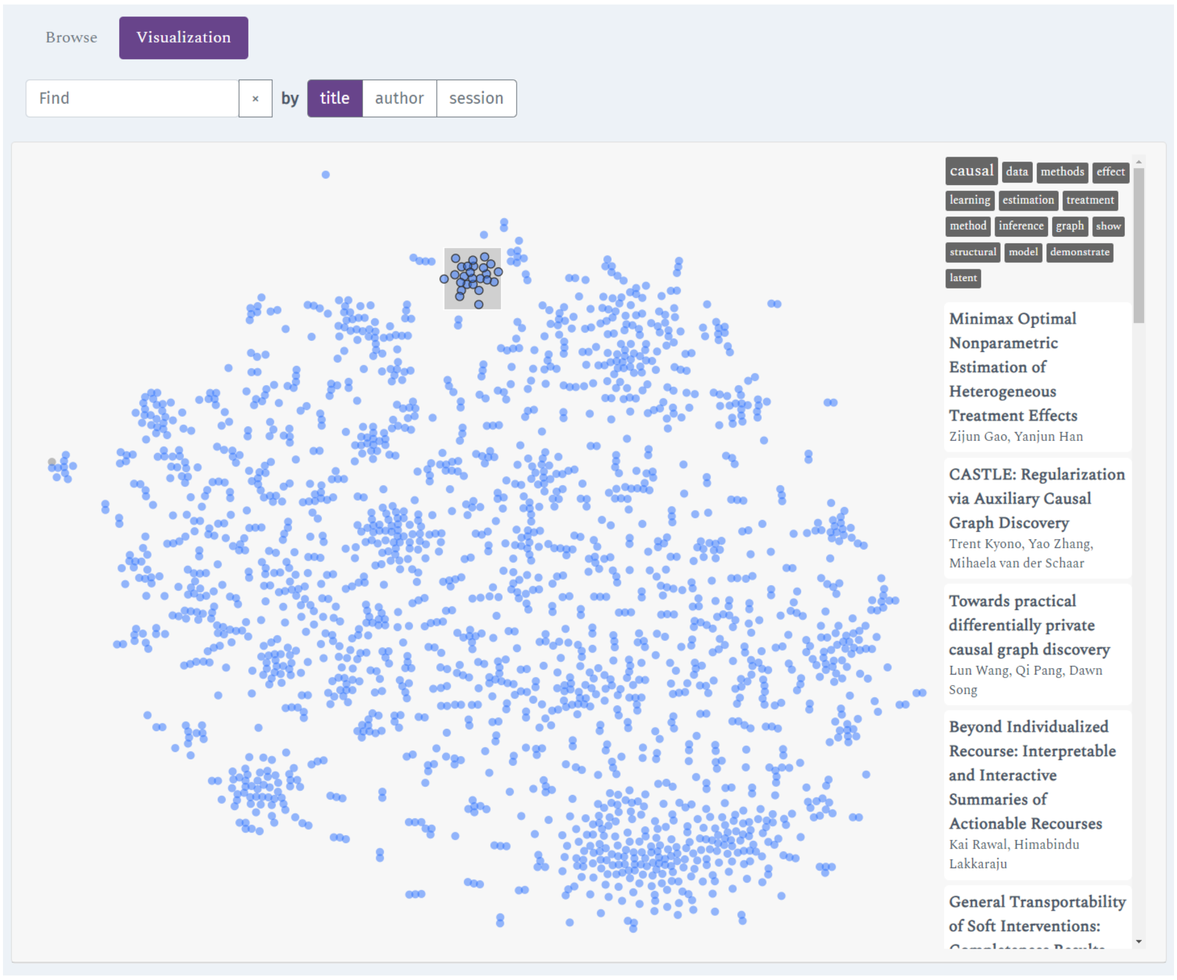
The second approach would use raw word counts, without using a non-linear transformation or dimension reduction. This approach would take the user query as input, and return documents containing words in the query, perhaps ranked by the number of search query terms they contain. This approach does not work as well as latent semantic indexing, however, because it is extremely brittle and does not generalize. In this model, there is no notion of ”similarity”, but rather a binary result if a search term is contained within a document, or not. Therefore, a user could search using many terms related to a particular topic, but if an extremely useful document on that topic happens to not contain the particular search terms the user has entered, it will not be returned.

Looking at the specifics of the LSI implementation that will be used, there are two common non-linear transformations in the literature, tf-idf and log-entropy. The work in [7] suggests that the log-entropy transformation works the best for information retrieval tasks, and so this is the transform that is used. The LSI model will be used to assign an embedding vector to each document in the collection. The purpose of assigning an embedding vector as opposed to a raw topic string is to be able to compute topic similarities in the form of cosine distance between vectors, as well as clustering of topics using a k-nearest neighbours algorithm. The LDA algorithm will be used to generate latent topic probabilities for each paper, and will also be stored in the database. We will be using the Python library Gensim [8] for its ease-of-use topic modelling capabilities.

6.4 Reference Designs

The two primary reference designs are the ICLR [31] and NeurIPS [32] 2020 conference sites.

These provide a good view of what our end product may look like. The difference, however, is that we would like a stronger suite of visualization capabilities and the ability for users to search on internal documents. In addition, we do not have access to any LaTeX source nor document meta-data unlike these conferences.



Minimax Optimal Nonparametric Estimation of Heterogeneous Treatment Effects
Zijun Gao, Yanjun Han

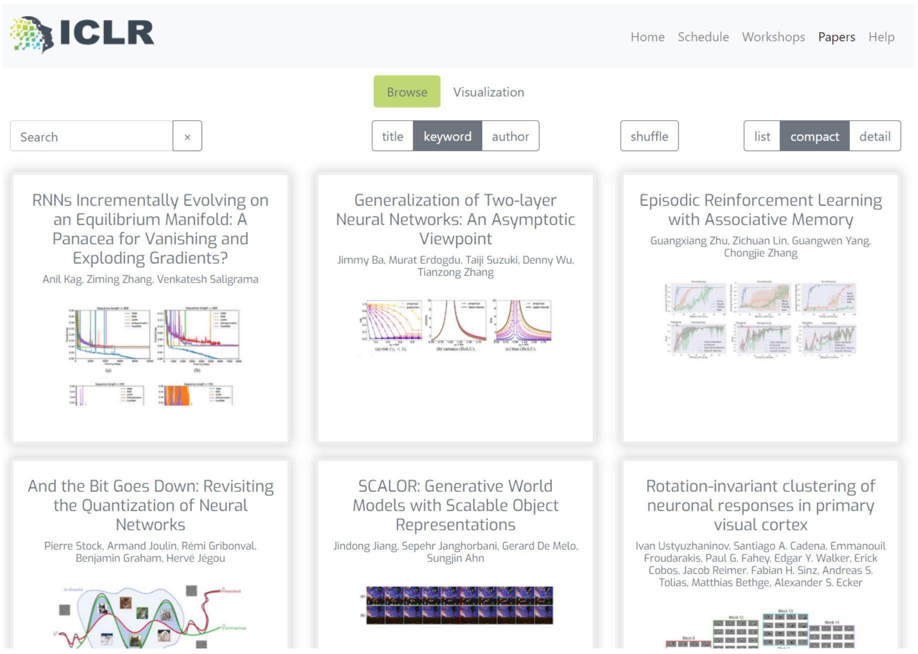
CASTLE: Regularization via Auxiliary Causal Graph Discovery
Trent Kyono, Yao Zhang, Mihaela van der Schaar

Towards practical differentially private causal graph discovery
Lun Wang, Qi Pang, Dawn Song

Beyond Individualized Recourse: Interpretable and Interactive Summaries of Actionable Recourses
Kai Rawal, Himabindu Lakkaraju

General Transportability of Soft Interventions:

Figure 5: NeurIPS 2020 conference site paper visualization [32]



RNNs Incrementally Evolving on an Equilibrium Manifold: A Panacea for Vanishing and Exploding Gradients?
Anil Kag, Ziming Zhang, Venkatesh Saligrama

Generalization of Two-layer Neural Networks: An Asymptotic Viewpoint
Jimmy Ba, Murat Erdogdu, Taiji Suzuki, Denny Wu, Tianzong Zhang

Episodic Reinforcement Learning with Associative Memory
Guangxiang Zhu, Zichuan Lin, Guangwen Yang, Chongjie Zhang

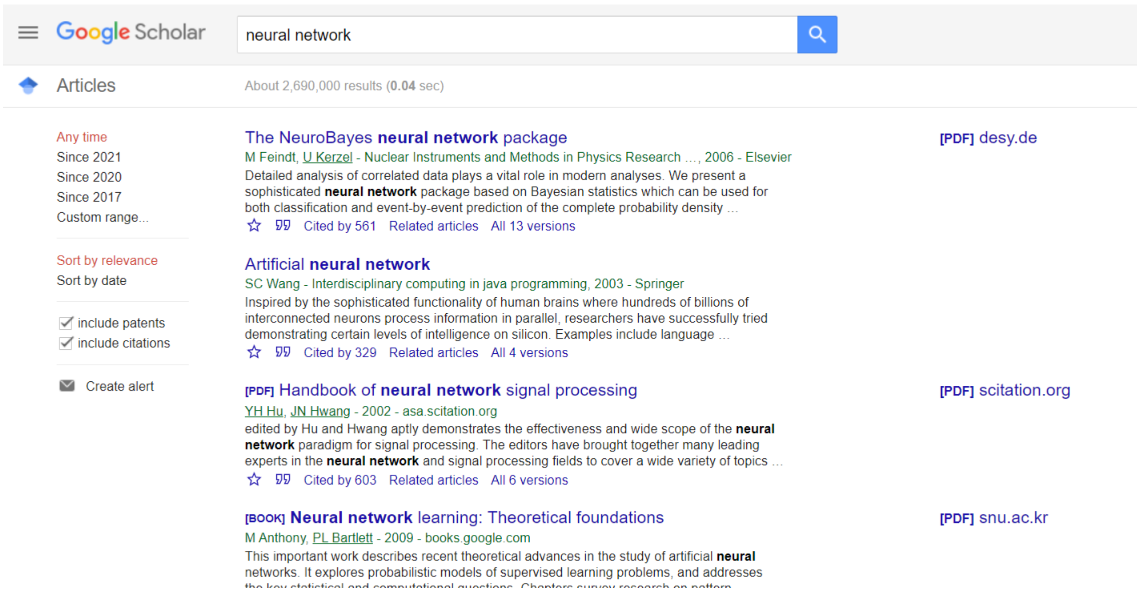
And the Bit Goes Down: Revisiting the Quantization of Neural Networks
Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, Hervé Jégou

SCALOR: Generative World Models with Scalable Object Representations
Jindong Jiang, Sepehr Janghorbani, Gerard De Melo, Sungjin Ahn

Rotation-invariant clustering of neuronal responses in primary visual cortex
Ivan Ustyuzhaninov, Santiago A. Cadena, Emmanouil Froudarakis, Paul G. Fahey, Edgar Y. Walker, Erick Cobos, Jacob Reimer, Fabian H. Sinz, Andreas S. Tolias, Matthias Bethge, Alexander S. Ecker

Figure 6: ICLR 2020 conference site paper browsing [31]

Two additional reference designs for their pure search capabilities are Google Scholar [39] and NASA's technical report server [40]. They possess no visualization capabilities but have appear to have strong searching mechanisms.



About 2,690,000 results (**0.04** sec)

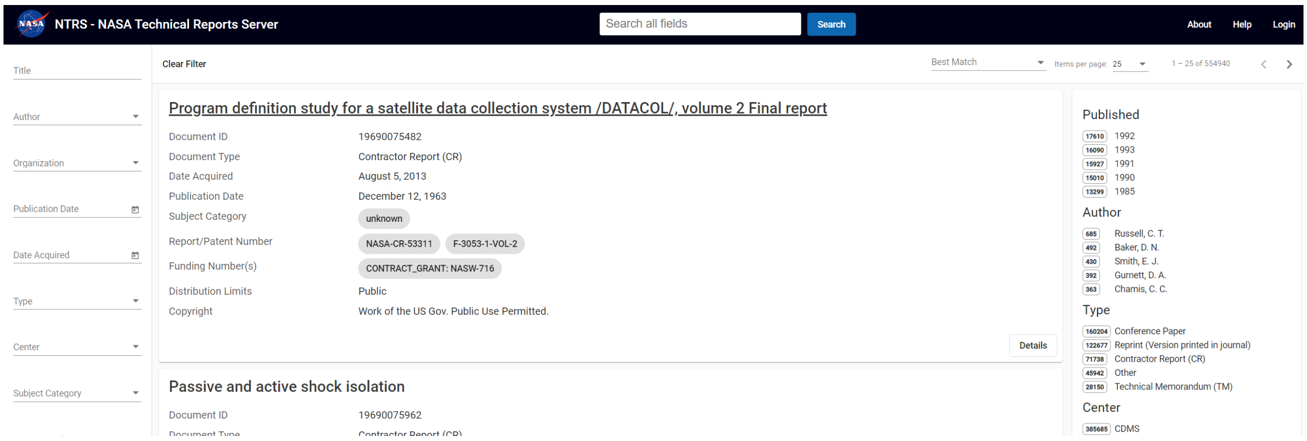
The NeuroBayes **neural network** package — [PDF] desy.de
M Feindt, U Kerzel - Nuclear Instruments and Methods in Physics Research ..., 2006 - Elsevier
Detailed analysis of correlated data plays a vital role in modern analyses. We present a sophisticated **neural network** package based on Bayesian statistics which can be used for both classification and event-by-event prediction of the complete probability density ...
Cited by 561 Related articles All 13 versions

Artificial **neural network**
SC Wang - Interdisciplinary computing in java programming, 2003 - Springer
Inspired by the sophisticated functionality of human brains where hundreds of billions of interconnected neurons process information in parallel, researchers have successfully tried demonstrating certain levels of intelligence on silicon. Examples include language ...
Cited by 329 Related articles All 4 versions

[PDF] Handbook of **neural network** signal processing — [PDF] scitation.org
YH Hu, JN Hwang - 2002 - asa.scitation.org
edited by Hu and Hwang aptly demonstrates the effectiveness and wide scope of the **neural network** paradigm for signal processing. The editors have brought together many leading experts in the **neural network** and signal processing fields to cover a wide variety of topics ...
Cited by 603 Related articles All 6 versions

[BOOK] **Neural network** learning: Theoretical foundations — [PDF] snu.ac.kr
M Anthony, PL Bartlett - 2009 - books.google.com
This important work describes recent theoretical advances in the study of artificial **neural** networks. It explores probabilistic models of supervised learning problems, and addresses

Figure 7: Google Scholar academic paper searching



Program definition study for a satellite data collection system /DATACOL/, volume 2 Final report

Document ID	19690075482
Document Type	Contractor Report (CR)
Date Acquired	August 5, 2013
Publication Date	December 12, 1963
Subject Category	unknown
Report/Patent Number	NASA-CR-53311 F-3053-1-VOL-2
Funding Number(s)	CONTRACT_GRANT: NASW-716
Distribution Limits	Public
Copyright	Work of the US Gov. Public Use Permitted.

Passive and active shock isolation

Document ID	19690075962
Document Type	Contractor Report (CR)

Published
17610 1992
16090 1993
15927 1991
15010 1990
13299 1985

Author
685 Russell, C. T.
492 Baker, D. N.
430 Smith, E. J.
392 Gurnett, D. A.
363 Chamis, C. C.

Type
160204 Conference Paper
122677 Reprint (Version printed in journal)
71738 Contractor Report (CR)
45942 Other
28150 Technical Memorandum (TM)

Center
385685 CDMS

Figure 8: NASA technical reports server

Microsoft Teams can integrate with other Microsoft Office products. For companies who have all their documents written in Microsoft Word, Powerpoint, Excel, etc. and also use Teams, they have a way to search through their documents based on keywords. This provides the functionality of local file search that we aim to provide, but can only search for keywords in the files and provides no notion of topic relatedness/visualization of such. This also requires a company to fully integrate with Microsoft products to make the most advantage of this.

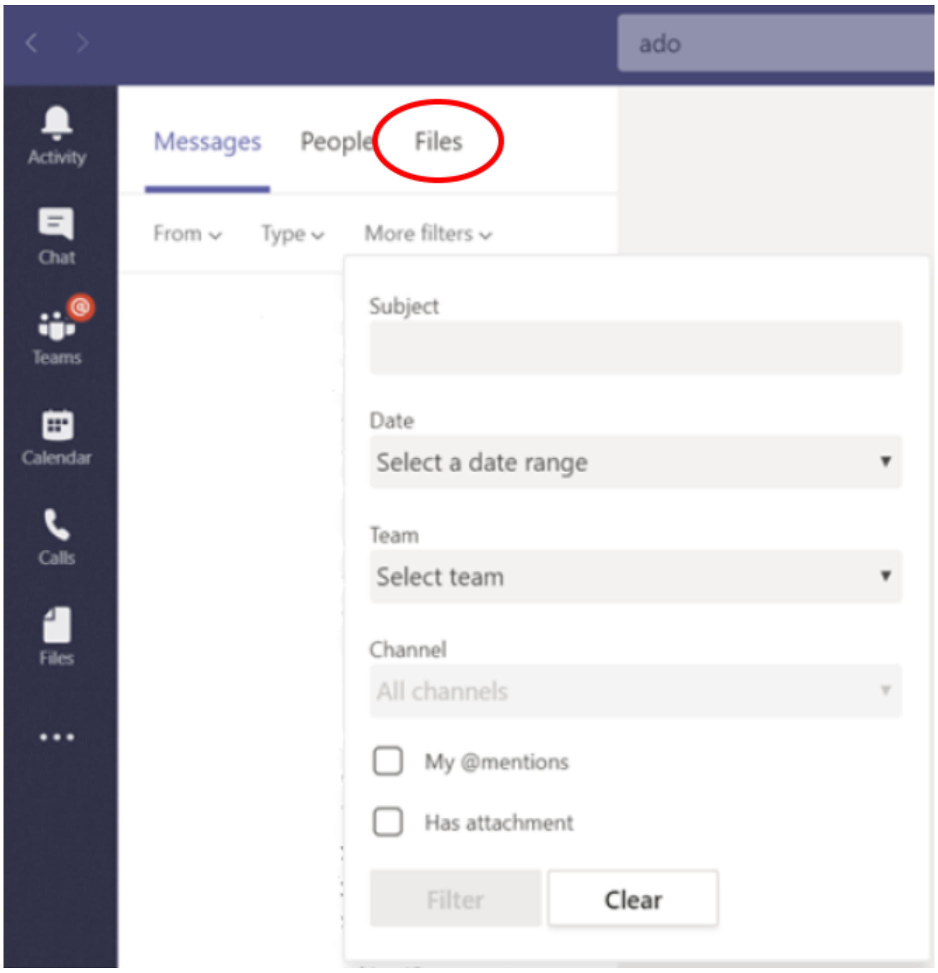


Figure 9: Microsoft Teams file search

6.5 Future Design Considerations

6.5.1 Optical Character Recognition (OCR)

An important additional feature would be the inclusion of OCR. Some of the PDFs (especially if they are older) may exist as a collection of scanned images. The NLP model requires text as input and so such PDFs must be converted to text through the use of OCR technologies. Python's popular OpenCV [41] library has tools for such a task.

6.5.2 Transformers

Transformers have gained much traction in recent times and are an extremely popular state-of-the-art for natural language processing (NLP) tasks. In particular, self-attention transformer models work by applying learned "attention" weights to pre-trained word embeddings. The attention weight can thus be used to find the centroid of a document in the semantic space more effectively than a simple sum of the embedding space vectors for the individual words. The attention weight is computed using the following transformation [42]:

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V \tag{4}$$

These transformers can be used to learn features from words in documents for further analysis with other ML techniques. PyTorch has pre-trained implementations for notorious transformer models such as BERT, GPT, XLM, etc. [43]

6.5.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are another popular choice of deep-learning for NLP. Due to their recurrent nature, they are able to capture past information when creating embeddings for their inputs. This is particularly useful in NLP tasks, as language has a very sequential structure. LSTMs and GRUs are much more widespread adaptations to the traditional RNN as they better capture long-term history of inputs. The figures below from [44] illustrate the specific mechanisms of RNNs, and the modifications made by LSTMs and GRUs.

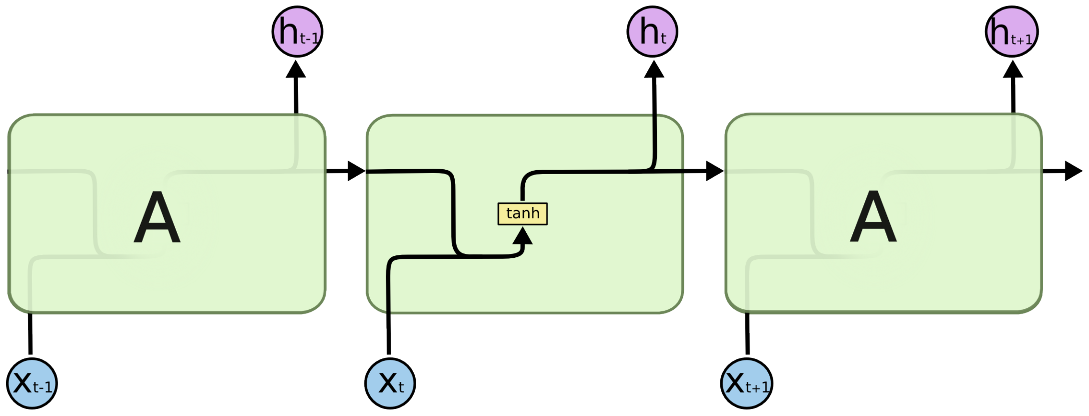


Figure 10: Original RNN Architecture

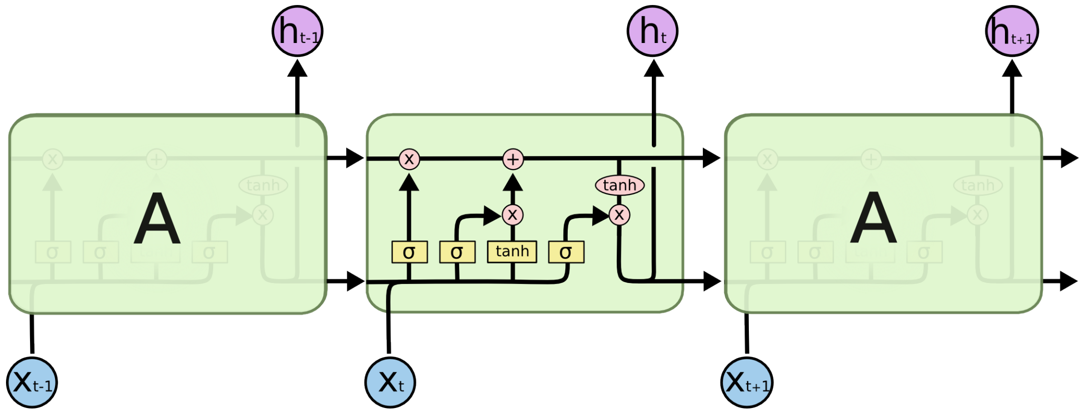
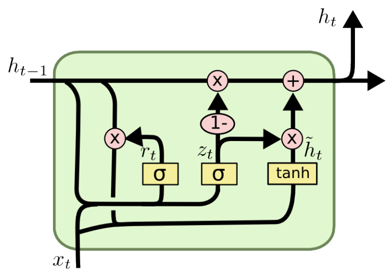


Figure 11: Long Short Term Memory - A Popular Variant

These models can be used in a similar way to transformers; however, transformers have out-popularized such neural networks in NLP.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 12: Gated Recurrent Unit - Another Popular Variant

7 Subsystem Design Specifics

In this section we outline implementation details and specific aspects of our product.

For the scope of this project, our web app and all associated services (i.e., database, model) are run locally. In this software-based project, we are not using any special tools that require purchase. All development work is done on our personal laptops.

For the machine learning model development, Gensim is used for its optimized, built-in natural-language processing capabilities.

Furthermore, for the database, Neo4j [37], a standard framework for graph database development, is used for the design of the database that stores the relationships between the academic papers.

For the front-end application there is a tool for prototyping, an overall framework, and a library for dynamic visualization. For prototyping, InVision is used. ReactJS is used as a front-end framework.

The back-end is written in Python using the Flask framework for easy compatibility with database/model code.

7.1 Front-End Client

React is being used as the framework for the front-end application. To build this application, Node.js, and within that, React, needs to be installed. Within the repository, the source code for the client is in “webapp/client/src”. The web elements that are common to every page are included in App.js within this folder, and every other component is coded in the react_components folder within “webapp/client/src”. Anytime dynamic information relating to documents is needed, a GET request is made to the server in the background “localhost:5000/[args]” in which the arguments of the URL specify what information is needed, then the application waits for a response, which will be in the form of a JSON object, and then processes this response in order to display it in an appropriate manner to the user.

7.1.1 App Description

The overall application is described in “webapp/client/src/App.js”. It describes constant elements throughout the application. It also has a switch element which will be populated based on what URL extension of the application the user is currently on. In the case of the search results and article view, the URL will take an argument that affects what search results, or article, will be shown.

In order to use the HomePage, ResultPage, and ArticlePage elements, they must be imported:

Listing 1: Including React Components

```
import HomePage from './react_components/HomePage/HomePage.js';
import ResultPage from './react_components/ResultPage/ResultPage.js';
import ArticlePage from './react_components/ArticlePage/ArticlePage.js';
import SearchBar from './react_components/SearchBar/SearchBar.js';
```

Home Page

Code in “webapp/client/src/react_components/HomePage”. The home page is simply an instance of the search bar on an otherwise empty page. We can see the base page is located at localhost:3000. The search bar is described in more detail in a following section.

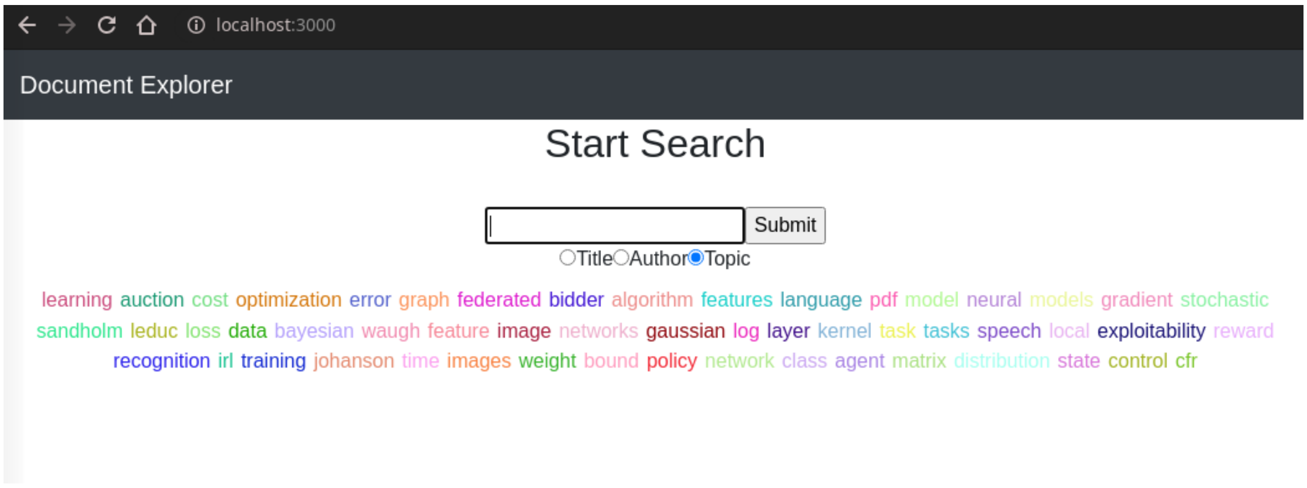
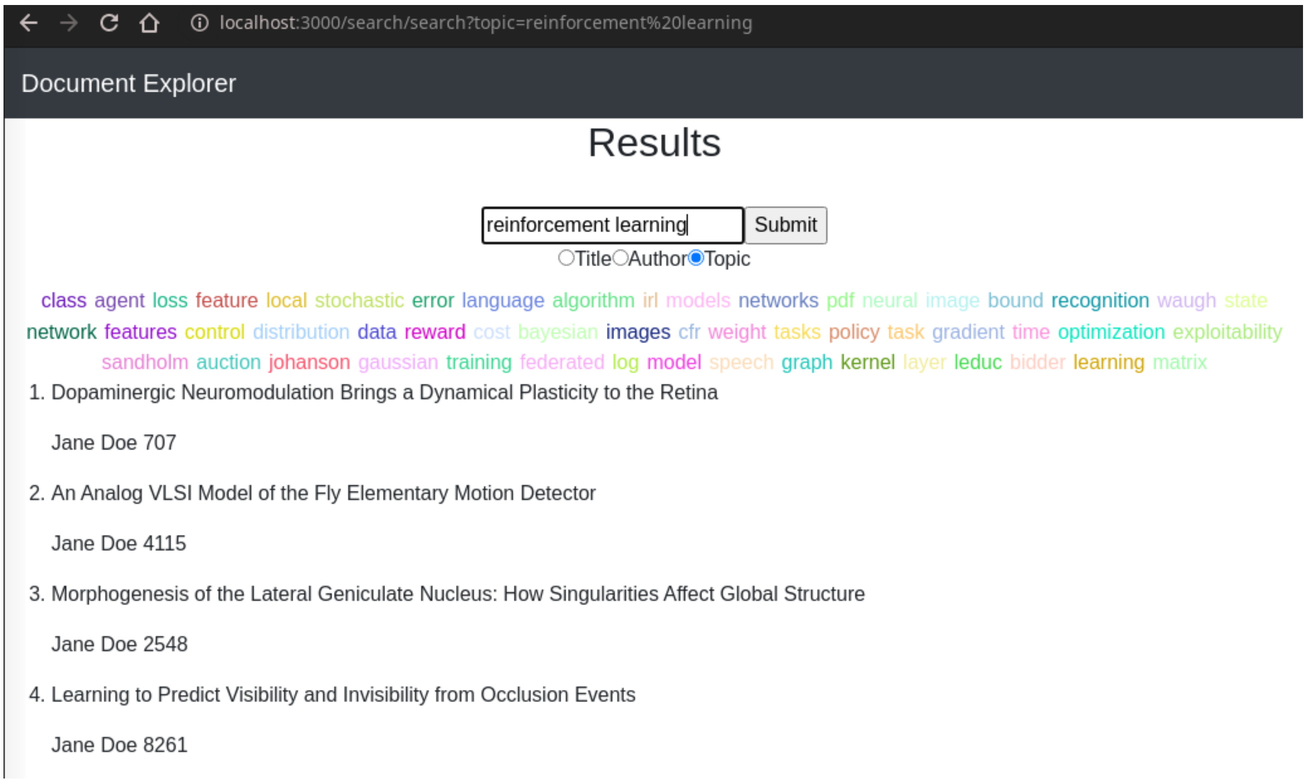

localhost:3000
Document Explorer
Start Search
Submit
Title Author [✓ Topic]
learning auction cost optimization error graph federated bidder algorithm features language pdf model neural models gradient stochastic sandholm leduc loss data bayesian waugh feature image networks gaussian log layer kernel task tasks speech local exploitability reward recognition irl training johanson time images weight bound policy network class agent matrix distribution state control cfr

Figure 13: App home page

Search Result Page

Code in “webapp/client/src/react_components/SearchPage”. When a search result page is loaded, the search query from its url is extracted and used to create a fetch request to the server as shown below. In the below example, the URL specifies that papers with a topic related to “reinforcement learning” should be queried for. The page consists of a search bar to enable further searches (word cloud belongs to search bar), as well as search results.



Results

reinforcement learning

Title Author [✓ Topic]

class agent loss feature local stochastic error language algorithm irl models networks pdf neural image bound recognition waugh state network features control distribution data reward cost bayesian images cfr weight tasks policy task gradient time optimization exploitability sandholm auction johanson gaussian training federated log model speech graph kernel layer leduc bidder learning matrix

1. Dopaminergic Neuromodulation Brings a Dynamical Plasticity to the Retina

   Jane Doe 707

2. An Analog VLSI Model of the Fly Elementary Motion Detector

   Jane Doe 4115

3. Morphogenesis of the Lateral Geniculate Nucleus: How Singularities Affect Global Structure

   Jane Doe 2548

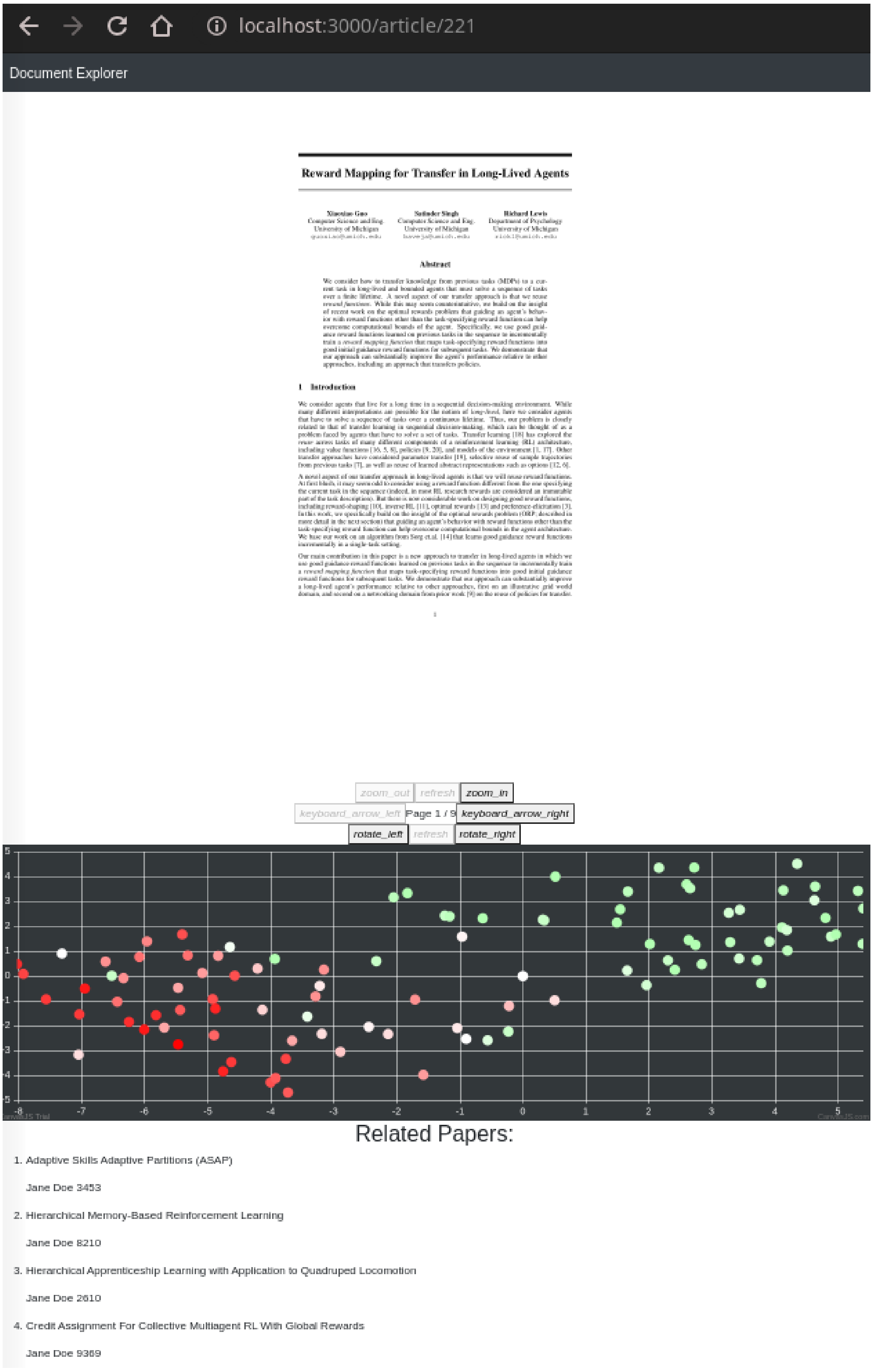
4. Learning to Predict Visibility and Invisibility from Occlusion Events

   Jane Doe 8261

Figure 14: App result page

Article Page

Code in “webapp/client/src/react_components/ArticlePage”. The article page consists of 3 components, the PDF viewer, a visualization of the 100 most LSI-related papers, and a list of the same 100 most LSI-related papers. In the visualization, each dot represents one of the papers in the list below it. The dot at coordinate (0,0) always belongs to the paper corresponding to the current page. The shade of the point represents a corresponding time for the paper. In this case, each paper corresponds to the year that the paper was presented at the NeurIPS conference. Papers in white represent papers with the same year of the paper of the current web page. Papers are then shaded green or red depending on if it was presented after or before the current paper respectively. The full range of white to red/green is used to the maximum of the full range of the paper that is published most before or after the current one, or 8 years before or after the current paper. This is done to use as much of the range of shades as possible, but to show a limited range of colours when all papers are within a 2-3 year period.



Reward Mapping for Transfer in Long-Lived Agents

Xiaoxiao Guo
Computer Science and Eng.
University of Michigan
guoxiao@umich.edu

Satinder Singh
Computer Science and Eng.
University of Michigan
baveja@umich.edu

Richard Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Abstract

We consider how to transfer knowledge from previous tasks (MDPs) to a current task in long-lived and bounded agents that must solve a sequence of tasks over a finite lifetime. A novel aspect of our transfer approach is that we reuse reward *functions*. While this may seem counterintuitive, we build on the insight of recent work on the optimal rewards problem that guiding an agent's behavior with reward functions other than the task-specifying reward function can help overcome computational bounds of the agent. Specifically, we use good guidance reward functions learned on previous tasks in the sequence to incrementally train a *reward mapping function* that maps task-specifying reward functions into good initial guidance reward functions for subsequent tasks. We demonstrate that our approach can substantially improve the agent's performance relative to other approaches, including an approach that transfers policies.

1 Introduction

We consider agents that live for a long time in a sequential decision-making environment. While many different interpretations are possible for the notion of *long-lived*, here we consider agents that have to solve a sequence of tasks over a continuous lifetime. Thus, our problem is closely related to that of transfer learning in sequential decision-making, which can be thought of as a problem faced by agents that have to solve a set of tasks. Transfer learning [18] has explored the reuse across tasks of many different components of a reinforcement learning (RL) architecture, including value functions [16, 5, 8], policies [9, 20], and models of the environment [1, 17]. Other transfer approaches have considered parameter transfer [19], selective reuse of sample trajectories from previous tasks [7], as well as reuse of learned abstract representations such as options [12, 6].

A novel aspect of our transfer approach in long-lived agents is that we will reuse reward functions. At first blush, it may seem odd to consider using a reward function different from the one specifying the current task in the sequence (indeed, in most RL research rewards are considered an immutable part of the task description). But there is now considerable work on designing good reward functions, including reward-shaping [10], inverse RL [11], optimal rewards [13] and preference-elicitation [3]. In this work, we specifically build on the insight of the optimal rewards problem (ORP; described in more detail in the next section) that guiding an agent's behavior with reward functions other than the task-specifying reward function can help overcome computational bounds in the agent architecture. We base our work on an algorithm from Sorg et.al. [14] that learns good guidance reward functions incrementally in a single-task setting.

Our main contribution in this paper is a new approach to transfer in long-lived agents in which we use good guidance reward functions learned on previous tasks in the sequence to incrementally train a *reward mapping function* that maps task-specifying reward functions into good initial guidance reward functions for subsequent tasks. We demonstrate that our approach can substantially improve a long-lived agent's performance relative to other approaches, first on an illustrative grid world domain, and second on a networking domain from prior work [9] on the reuse of policies for transfer.

1

Related Papers:

1. Adaptive Skills Adaptive Partitions (ASAP)

   Jane Doe 3453

2. Hierarchical Memory-Based Reinforcement Learning

   Jane Doe 8210

3. Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion

   Jane Doe 2610

4. Credit Assignment For Collective Multiagent RL With Global Rewards

   Jane Doe 9369

Figure 15: App article page

Search Bar

Described in "webapp/client/src/react_components/SearchBar". This component will be used both in the home page, and in the search results page. The list of words is taken from the unique set of words from the LDA model description of the corpus of documents. Clicking on a word in the word cloud simply adds a space and the word to the search bar. It is recommended this is used for creating a topic search, but the string in the search bar can be used to search in any mode. There are 3 modes of search title (C1), author, and topic (C2).

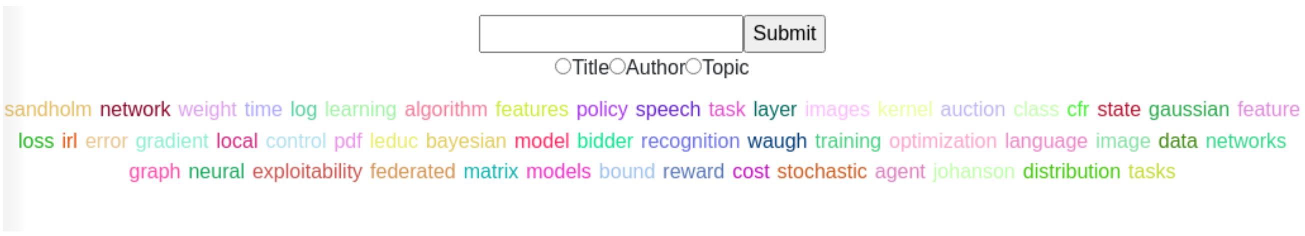

Submit
Title
Author
Topic
sandholm network weight time log learning algorithm features policy speech task layer images kernel auction class cfr state gaussian feature
loss irl error gradient local control pdf leduc bayesian model bidder recognition waugh training optimization language image data networks
graph neural exploitability federated matrix models bound reward cost stochastic agent johanson distribution tasks

Figure 16: App article page

7.1.2 Front-End Verification

First of all, the prototype satisfies all the specifications outlined for this application (S13, S14, S15, S16).

The paper "Verification Templates for the Analysis of User Interface Software Design" [45] in conjunction with "Verification of User Interface Software: The Example of Use-Related Safety Requirements and Programmable Medical Devices" [46] are used as a guide, keeping in mind these practices are intended for safety-critical systems, thus our product is not required to adhere to the stringent methodology outlined in the paper, but can incorporate some its principles where it is practical and useful.

"Verification Templates for the Analysis of User Interface Software Design" describes templates for model-based analysis of usability and safety of UI software design. These ensure principles common user safety requirements such as ability to undo actions, and predictability of behaviour.

The focus of the second paper is to demonstrate that software is compliant with use-related requirements using testing, inspections, analysis during the development cycle, and more specifically, "A methodology is presented based on the use of formal methods technologies to provide guidance to developers about addressing three key verification challenges: 1) how to validate a model, and show that it is a faithful representation of the device; 2) how to formalize requirements given in natural language, and demonstrate the benefits of the formalization process; and 3) how to prove requirements of a model using readily available formal verification tools."

There are three formal techniques provided to analyze the user interface software of a user interface product, model checking, theorem proving, and simulation.
The methodology the authors of the paper suggest as a guideline when using formal methods to the analysis of UI software against user-related requirements. This includes showing:

1. the model accurately describes the interactions of the product
2. natural language requirements are accurately translated into formal properties
3. model satisfies the formal properties
4. failures in the previous point results in a redesign of the product of a modification to the property

For our purposes, we will create a model of the application, and verify the design/prototype against design templates.

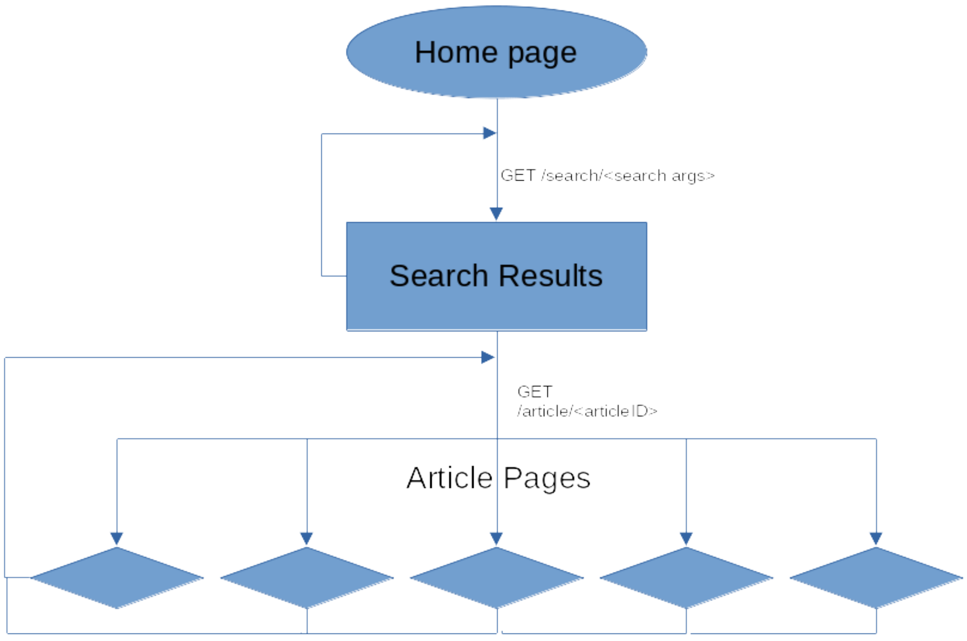


Figure 17: UI Model

Templates:

Completeness This describes how easily, or how many interactions are needed to access any part of the functionality. Typically any interaction should be within 3 interactions of the base view.

This is satisfied by this model. Looking at the figure above, we can see any page is within 2 interactions of the home page.

Feedback This describes that any change to the state of the application is perceivable by the user. Below is a list of possible interactions, and corresponding visual feedback the user should be able to see.

Action	Feedback
Click on logo	Home page rendered
Type search text	Characters displayed in search bar
Click on search mode	Circle by mode is filled
Click on keyword for search	Word added to text in search bar
Submit Search	Result page appears or changes to show new results
Click on article	Article page is rendered
Click on article point	Article page is rendered

Table 3: Action-Feedback Table

Consistency Users are able to develop an idea of how an application functions, keeping as much consistent throughout the experience as possible is important such as layout, structure, navigation, terminology and control elements to ensure predictable behaviour.

There are 3 views. For each, there is a common element, the top bar, which includes a logo that redirects to the home page. For the home page and result page, the search bar is sourced from the exact same code, so the layout is the same. The same is true for the results articles in the result page and article page. No other elements are common between views.

Reversibility Users should be able to reverse, stop, modify, restart a previous action when there are problems or abnormalities.

Care was made so that the page URL includes all the information needed to describe the state of the application. This means any interaction submitted to the application to change the state is reversed by pressing the back button in the browser. This also means that a URL can be passed from one user to another to render the same page without the need for any other state information.

Visibility Visual or auditory cues should be included to draw the user's attention to important information necessary for correct operation.

This is not totally applicable to this scenario since this is not a critical system, and undue sights and sounds are not helpful for such and application. That being said, a form of visibility is implemented when no search mode is selected. If a search is attempted without a search mode selected, an alert is triggered that informs the user that a mode must be selected, and this message must be acknowledged.

Universality Guidelines regarding certain design choices should be the same throughout, such as knob rotation or "submit" button placement.

This is related to consistency, where possible the same code is used in different situations so that design is as universal as possible throughout the application.

7.2 Backend Server

The backend is written in Python using the Flask framework for easy integration with our Neo4j database driver (written in Python). It is written as a ReST API, incorporating standards from Section 3.4. Information on starting the backend is present in Section 4.2 (Quickstart Guide).

The backend currently listens on port 5000 (Flask default) for GET requests to the following routes:

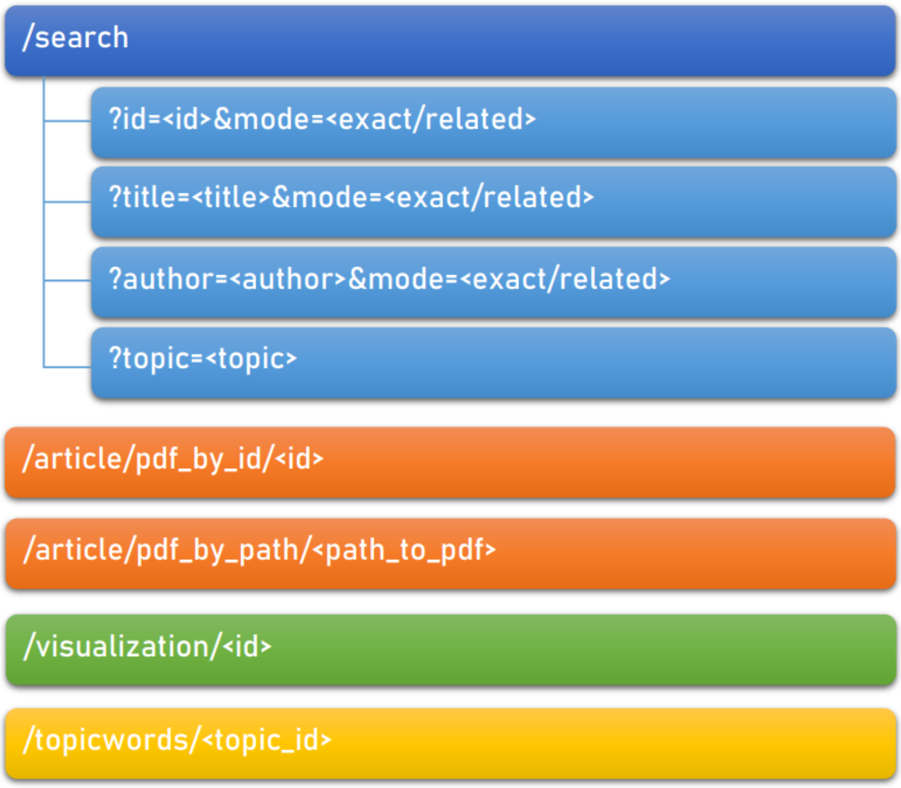

/search
?id=<id>&mode=<exact/related>
?title=<title>&mode=<exact/related>
?author=<author>&mode=<exact/related>
?topic=<topic>
/article/pdf_by_id/<id>
/article/pdf_by_path/<path_to_pdf>
/visualization/<id>
/topicwords/<topic_id>

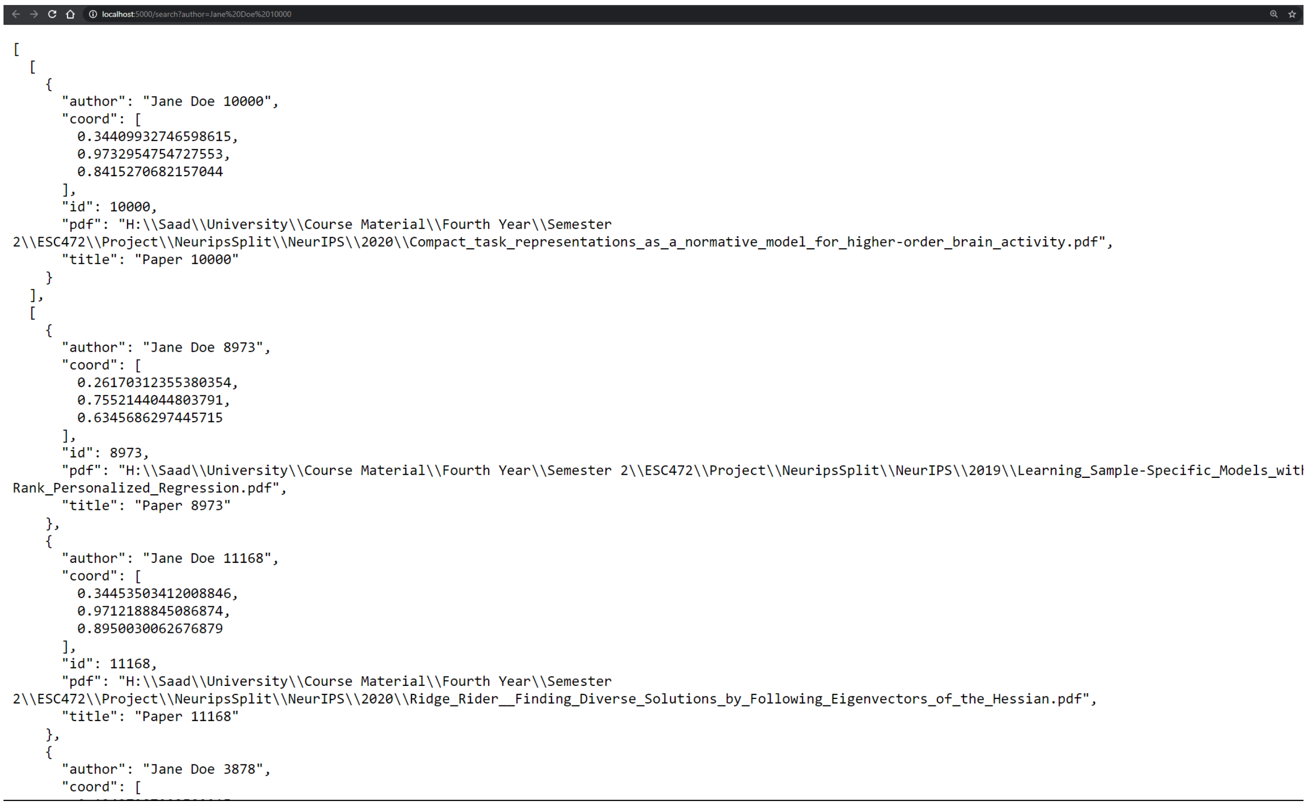
Figure 18: Backend API routes

- `/search` – Main search route. Looks for URL query string arguments `id`, `author`, `title`, or `topic`. Unless querying by topic, the additional `mode` query string argument must be provided as one of `exact` or `related` to return exact matches of the search or related matches. Returns a list of document objects (object with key-value pairs, see Database

section below for more info) based on the query. Extending to new queries is trivial once they are implemented in the database. Helps achieve product specifications *(S4), (S13), (S14)* and constraints *(C1), (C2)*. Example queries:
`http://localhost:5000/search?author=Jane%20Doe%20100&mode=exact`,
`http://localhost:5000/search?title=Phasor%20Neural%20Networks&mode=exact`,
`http://localhost:5000/search?id=0&mode=exact`,
`http://localhost:5000/search?topic=reinforcement%20learning`

- `/article/pdf_by_id/<id>` – Serves PDF contents with HTTP Content-Type `application/pdf`. Requires document ID in database as `<id>`. Queries database for the document's PDF path and reads PDF bytes from that path. Helps achieve specification *(S16)*.
- `/article/pdf_by_path/<pdf_path>` – Serves PDF contents with HTTP Content-Type `application/pdf`. Requires file system path to PDF as `<pdf_path>` and reads PDF bytes from that path. Currently unused, but included for a potential secondary method if getting PDF by ID is found to be unideal.
- `visualization/<id>` – Queries the database for paper ID `<id>` and its nearest neighbours, adds a key `"processed_coord"` to each document which represents a t-SNE 2d projection of the high dimensional LSI vectors to preserve euclidian distance, and returns these documents as a list (first document is the exact match). Helps achieve specification *(S15), (S16)*.
- `topicwords/<topic_id>` – Gets the most prominent words from each topic from the LDA model. `<topic_id>` must be $0 - \texttt{numLDA} - 1$ (inclusive) where `numLDA` is the number of LDA dimensions passed in when starting the backend (see Section 4.2 (Quickstart Guide)). Helps achieve specification *(S16)*.

The server connects to and queries the Neo4j database and returns the response as JSON. Querying the database takes on the order of 100 ms but connecting takes on the order of 2 s. Since fast query responses is an important consideration of our design (*(G1)*), the server establishes an open connection with the database when it starts and leaves the connection open, as opposed to opening and closing the connection on every request. Thus, queries can be made in 100 ms. The response from the database is always a list of dictionaries, where each dictionary contains stored information for one paper (e.g., its path in the file-system, its topic coordinates, etc.).



```
[
  [
    {
      "author": "Jane Doe 10000",
      "coord": [
        0.34409932746598615,
        0.9732954754727553,
        0.8415270682157044
      ],
      "id": 10000,
      "pdf": "H:\\Saad\\University\\Course Material\\Fourth Year\\Semester
2\\ESC472\\Project\\NeuripsSplit\\NeurIPS\\2020\\Compact_task_representations_as_a_normative_model_for_higher-order_brain_activity.pdf",
      "title": "Paper 10000"
    }
  ],
  [
    {
      "author": "Jane Doe 8973",
      "coord": [
        0.26170312355380354,
        0.7552144044803791,
        0.6345686297445715
      ],
      "id": 8973,
      "pdf": "H:\\Saad\\University\\Course Material\\Fourth Year\\Semester 2\\ESC472\\Project\\NeuripsSplit\\NeurIPS\\2019\\Learning_Sample-Specific_Models_wit
Rank_Personalized_Regression.pdf",
      "title": "Paper 8973"
    },
    {
      "author": "Jane Doe 11168",
      "coord": [
        0.34453503412008846,
        0.9712188845086874,
        0.8950030062676879
      ],
      "id": 11168,
      "pdf": "H:\\Saad\\University\\Course Material\\Fourth Year\\Semester
2\\ESC472\\Project\\NeuripsSplit\\NeurIPS\\2020\\Ridge_Rider__Finding_Diverse_Solutions_by_Following_Eigenvectors_of_the_Hessian.pdf",
      "title": "Paper 11168"
    },
    {
      "author": "Jane Doe 3878",
      "coord": [
```

Figure 19: Server JSON response from mock search on paper title

7.3 Database

As mentioned previously, a graph database developed using Neo4j is used to store the documents as well as relationships between them.

Figure 20 illustrates the structure of the graph database. Each node corresponds to a particular paper and stores information about it. Each paper is connected to its K-nearest neighbours, based on the cosine similarity between their latent space vectors. This allows the design to easily express queries for a paper as well as its most related papers, and greatly improves the speed of related paper-lookup, as the relationship information is pre-computed and built-in to the database. In our initial prototype, each paper was connected to its 100-nearest neighbours, but this is a parameter that can be changed in the design.

Each database node contains the following properties about a paper:

- `paper_id`: Unique integer ID of the paper in the database.
- `pdf`: File system absolute path to the paper's PDF file.

  Sample value: `"/home/user/ml_paper.pdf"`.
- `title`: Paper or document title. Sample value: `"ML Paper"`.

- `author`: Paper or document author(s). Sample value: `"Jane Doe"`.
- `year`. Year the paper was published. Sample value: `"2018"`.
- `coord`: LSI latent space coordinates of the paper. Sample value (5-D latent space): `[0.133, 0.024, 0.566, 0.786, 0.111]`.
- `topic_prob`: LDA probabilities that the paper is in each of the possible topics. Sample value (5-D latent space): `[0.139, 0.001, 0.344, 0.475, 0.040]`.

Each entry in the database contains information about a particular paper, including relevant topic information, through the LSI vector and LDA probabilities, and also available metadata information, such as paper title and author. Therefore, the database design meets specifications *(S2) Create a database of papers with relevant topic information*, and *(S8) Must store an entry with topic information and available metadata.* Furthermore, the topic information stored in the LDA vector can be used to determine the most relevant topics for a particular paper, and the LSI vector can be used to compute the similarity between different papers, and thus the design meets specifications *(S9) Topic information must be able to be used to determine top descriptive topics* and *(S10) Topic information must be able to be used to determine a measure of relatedness between papers.* PDFs are provided by the database to the ML model at training time, and the absolute path to the PDF file for each particular paper is given in the database through the `pdf` property of each entry, which can be used by the front-end to load a PDF from the file system and display it to the user. Thus, the proposed database design meets specifications *(S11) Must provide PDF documents for use in the ML model* and *(S12) Must be accessible by the client so that when the PDF URL is fed to an in-browser PDF viewer, the articles can be viewed in-app.*

Neo4j supports Python as one of its official development languages. Python was chosen by the team as the language for the Neo4j database driver due to its ease of development, and ease of integration with the system's frontend.

Besides offering storage, the database allows users to query for papers by paper title, paper author, topic string, topic index, and paper ID.

A Python-based database driver has been developed, called `db_driver.py`, located in the `db_and_model/` folder of the product's repository, and provides functions to build and tear down a database, build KNN relationships, and query the database by author name, paper title, topic string, topic index, and paper ID. The NeurIPS dataset, which consists of 11,572 academic papers, was used as a reference when testing the driver.

The API provided by the database driver is as follows:

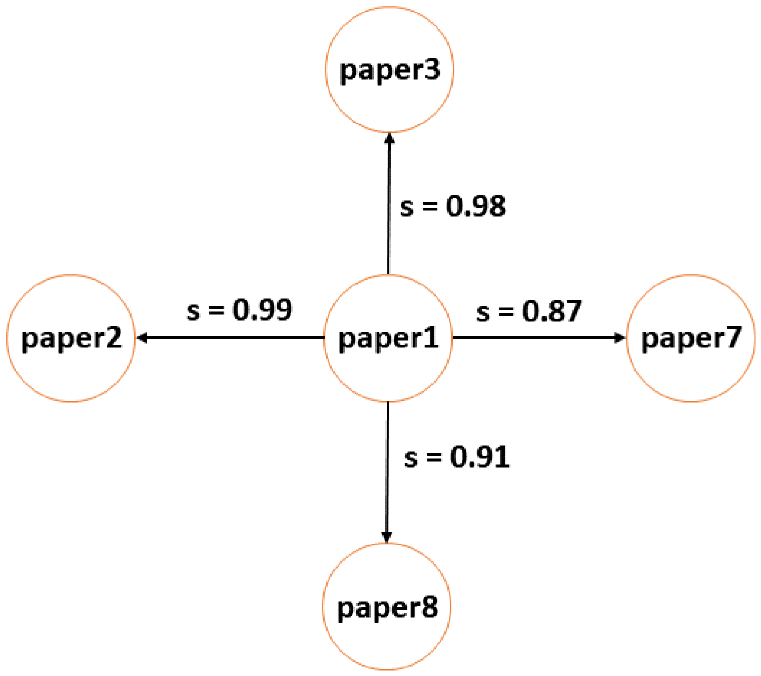


Figure 20: Diagram illustrating the relationships between a paper node in the database and its K-nearest neighbours.

1. `__init__(self, uri, user, password, numK, numLSI, numLDA, pdf_path, text_path, model_path, debug_info)`: Driver constructor. Takes in a Neo4j resource identifier (e.g. `"bolt://localhost:7687"`); user and password information for authentication; `numK`, a parameter indicating how many relationships each paper should have; `numLSI`, a parameter indicating the desired LSI dimensions; `numLDA`, a parameter indicating the desired LDA dimensions; `pdf_path`, a variable indicating the file system location where the driver should recursively look for PDFs; `text_path`, a variable indicating the file system path to the directory that stores the .txt files with the raw text content of the documents; `model_path`, a variable indicating the file system path to the directory that stores the machine learning model files; and `debug_info`, a variable that turns on/off debug information.

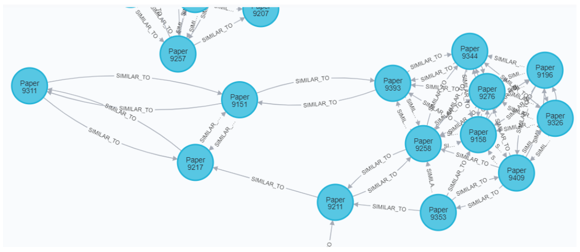
2. `build_db(self, train_model, create_db_nodes, convert_pdfs)`: Builds the database by searching for PDFs recursively starting from the given `pdf_path`. The parameter `train_model` is a boolean that indicates whether the ML model should be re-trained or loaded from a previously trained model stored in `model_path`; `create_db_nodes` is a boolean that controls whether the DB nodes should be re-populated; and `convert_pdfs` is a boolean that controls whether the raw text files obtained from the PDFs should be

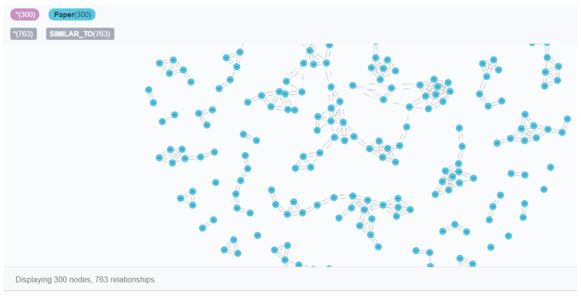
re-created by using the model's `pdf_to_text()` function or re-utilized.

3. `destroy_db()`: Tears down the database and the GDS graph, if existent.
4. `insert_node(self, paper_id, pdf, author, title, year, coord, topic_prob)`: Insert a single node on the database.
5. `build_knn_graph()`: Builds the GDS graph, and runs the KNN algorithm to build the relationships between papers.
6. `query_by_author(author, mode)`: Query the database by paper author, and return any matching papers or their K-nearest neighbours. The `mode` argument should be either `"exact"` for queries that should return exact matches or `"related"` for queries that should return recommendations.
7. `query_by_title(title, mode)`: Query the database by paper title, and return any matching papers or their K-nearest neighbours. The `mode` argument should be either `"exact"` for queries that should return exact matches or `"related"` for queries that should return recommendations.
8. `query_by_string(string)`: Query the database by topic string, and returns the top K papers. The string is converted to a latent space representation vector by the ML model, and the K papers with highest cosine similarity to the vector are returned.
9. `query_by_topic_index(topic_idx)`: Query the database by topic index, and returns the top K papers. Topic indices go from `0` to `numLDA-1`.
10. `query_by_paper_id(paper_id, mode)`: Query the database by paper ID, and return any matching papers or their K-nearest neighbours. The `mode` argument should be either `"exact"` for queries that should return exact matches or `"related"` for queries that should return recommendations.

During development, Neo4j server community edition 4.2.3 was used. It can be downloaded for free at [22], and installation instructions can be found for different OSes at [23]. Furthermore, in order to build the KNN relationships, Neo4j's Graph Data Science (GDS) library version 1.4.1 is used. It can be downloaded at [24], and installation instructions can be found at [25].

A sample KNN graph database was built using the 11,572 papers from the NeurIPS dataset, 3D vectors with random values for the semantic space representation of each paper, and mock values for the paper author and paper title fields of each paper. A relationship was built between each paper and its 100 nearest neighbours, as determined by the cosine similarity of the 3D vectors of each paper. Figure 21 shows visualizations of the nodes of the database and the relationships between them. Note that only 300 nodes were used in the rendering, and thus the entire database is not shown in the images.


(a) Zoomed-in view of the graph database



(b) Zoomed-out view of the graph database

Figure 21: KNN graph database built for the NeurIPS dataset. Only 300 nodes are shown in the database rendering, and thus not all relationships are depicted

7.3.1 Database Verification

Using the developed prototype driver script, the sample NeurIPS database was built and queried, and the latency of database accesses and build times were recorded. The database was re-built and destroyed 5 times, and queries for papers by author (using both "exact" and "related" query modes), title (also with both "exact" and "related" query modes), and topic were issued 1,000 times each so that relevant latencies could be measured. Results are shown in Table 4. These results correspond to the measurements required by metric *(M1) Average and standard deviation of database lookup time.*

Table 4: Database access, build, and tear down times. Accesses were performed 1,000 times, connect was performed 100 times, build and tear down latencies were measured 5 times.

Action	Mean Latency	Latency Std. Dev.
Tear Down	31.997 s	2.052 s
Build	244.596 s	7.208 s
Connect	2.025 s	4.14 ms
Author Query (Exact)	14.182 ms	1.496 ms
Author Query (Related)	27.002 ms	2.762 ms
Title Query (Exact)	14.365 ms	1.448 ms
Title Query (Related)	27.178 ms	2.643 ms
Topic Query	74.042 ms	6.507 ms

As seen in the table, the query type that takes longest is query-by-topic, taking on average 74.042 ms. Note that this time is much lower than the goal outlined in *(G1) Total query response time* of 500 ms query response time, and thus the database solution meets this goal. Furthermore, the coefficient of variation (ratio between standard deviation and mean) for the query latencies is near or below 10% for all query modes, indicating that access times do not have much variability. These results were measured on one of the team member's PC. Query times are expected to be even lower when the system runs on a server.

The time to build the database was found to be only slightly over 4 minutes on average, thus being much lower than the 12-hour update time goal set by goal *(G2) Model training time.*

In terms of characteristics of the database system, query times are approximately constant even with growing database sizes. This happens because the relationships between related papers are embedded in the database at build time, and thus do not need to be computed at query time. Therefore, the design meets the better-than-polynomial search complexity outlined in goal *(G3) Better-than polynomial search algorithm.* Furthermore, Neo4j is runnable on a server environment, therefore meeting goal *(G4) System should be runnable on a server.* The proposed design allows users to search for specific papers with its query-by-author and query-by-title functionalities, meeting constraint *(C1) The user must be able to search for particular papers*, and supports all three query types outlined in constraint *(C2) Multiple query types must be supported.* Moreover, Neo4j can handle data sets well over 15 GB as required by constraint *(C3) Support data sets over 15 GB*, with total storage only being limited by server capacity constraints.

Therefore, initial validation indicates that the proposed graph-based database developed with Neo4j is a very appropriate solution for the design, as it meets all criteria relevant to the database subsystem and presents access latencies well below the required threshold, even when running on a local computer.

7.4 Natural Language Processing Model

The model is implemented in Python, using the Gensim natural language processing library. The library defines the following objects (under the object-oriented programming model), which are imported into our Python script:

- Dictionary: provides a string $\leftrightarrow$ integer mapping for every unique string provided on instantiating the object
- Corpus: bag-of-words representation of a collection of documents. Maps each document to a vector of word frequency counts.
- LSI model: this takes the corpus as input, and provides dimensional-reduction for docu-

ment similarity computation.

- LDA model: similar to LSI model, but provides the Latent Dirichlet model to generate topics.
- Log-Entropy model: applies the log-entropy non-linear transformation given the raw document-term matrix as input (in the form of a corpus object)

Aligning with G2 & G4, the desire for overnight training (i.e. training is not done at query-time), and to be able to run on a server, the model provides an interface to load and store the model parameters. This ensures that if the computer loses power, or the program is restarted, the user can restart the program by loading model files stored on disk, and not retrain from scratch.

The initial prototype of the intelligent model is implemented as a SimilarityModel object and provides the following methods as an interface to the users:

1. `.build()`: generates the raw document-term matrix from scratch, trains and applies the log-entropy transformation, trains the LSI and LDA models from scratch, and saves all the model parameters to the disk
2. `.load()`: loads the necessary model parameters from the disk: the dictionary, the LSI model, the LDA model and the log-entropy model
3. `.set_topic_terms()`: outputs a list of lists of the top 10 words for each LDA topic cluster, for the front end to generate a word-cloud
4. `.document_map(filename)`: outputs the LSI co-ordinates and the LDA topic probabilities for the document with the provided filename
5. `.string_lookup(input_string)`: processes the provided string into a vector in the embedding space using the trained LSI model

The `.build()` and `.load()` methods provide the necessary interface to train and store, and subsequently reload, the model parameters, and build needs to be run to train the model for the first time. `.document_map(filename)`is run once upon start up to provide the word cloud information to the newly instantiated database driver object. The final two interface functions are run at query-time, and thus the user is directly affected by the speed at which those functions run.

The model also implements a simple function to transform the corpus of PDF documents into text files that can be parsed by Gensim. This functionality makes use of the tika Python package. The interface accepts an input directory of PDFs, and writes the text file output to the provided directory: `pdf_to_text(input_path,output_path)`.Writing the text files to disk, as opposed to simply storing the data in dynamic memory, ensures that this process only needs to be performed once (or whenever the corpus is updated).

7.4.1 Natural Language Processing Model Verification

The machine learning model allows use to extract the highest weighted words in the embedding space, which allows us to get a sense of the information that is being extracted by the model. The following top 10 keywords have been extracted for the 10 topics, from the NeurIPS database of papers, using the LSI model:

Topic 1	image	matrix	training	loss	gradient	algorithm	model	log	network	task
Topic 2	image	regret	bounds	visual	images	convex	object	theorem	bound	lemma
Topic 3	neurons	neuron	spike	stimulus	activity	firing	adversarial	synaptic	deep	cells
Topic 4	policy	reward	action	agent	reinforcement	actions	policies	rewards	mdp	environment
Topic 5	neurons	neuron	graph	sgd	posterior	descent	bayesian	gradient	latent	inference
Topic 6	variational	posterior	regret	latent	likelihood	bayesian	inference	object	monte	carlo
Topic 7	policy	graph	classifier	adversarial	regret	matrix	risk	posterior	classifiers	agent
Topic 8	node	graph	nodes	graphs	images	tree	image	object	edges	visual
Topic 9	svm	classifier	training	classifiers	kernel	classification	margin	graph	units	hidden
Topic 10	regret	rank	image	bandit	images	belief	energy	pixel	object	propagation

A previous iteration of this exercise, before the log-entropy tranformation was added to the training pipline, produced markedly worse results. The words "latexit", "sha" and "_base", which are clear artifacts of converting the PDFs into plain text, were picked up by the model. In the output above, it is clear that no spurious words have surfaced, which suggests that the model is now creating a much more effective latent space respresentation.

Looking now to the LDA model, the following keywords are extracted:

Topic 1	network	matrix	image	training	model	task	map	networks	time	data
Topic 2	model	training	graph	networks	log	algorithm	data	learning	classification	noise
Topic 3	task	matrix	gradient	learning	model	algorithm	convex	network	features	training
Topic 4	learning	function	loss	set	log	data	algorithm	distribution	training	model
Topic 5	stdp	depression	potentiation	ltp	tpre	neuromorphic	poo	photoreceptor	receptors	capacitor
Topic 6	matrix	model	image	training	algorithm	network	loss	gradient	task	log
Topic 7	network	image	feature	features	networks	data	model	training	graph	time
Topic 8	policy	state	stochastic	action	reward	gradient	network	optimal	control	convergence
Topic 9	network	training	model	image	networks	arxiv	layer	tasks	gradient	task
Topic 10	model	matrix	image	network	graph	training	networks	neurons	models	time

Here, as in the LSI model, there are some words which appear in both the singular and plural form (i.e. "network" and "networks"). This is both reassuring from a validation perspective (in general, the singular and plural forms of the same word have similar semantic meaning), but will require more thought in order to avoid confusing or frustrating the user (one idea would be to only show words in the singular form, if there are duplicates). Another artifact that appears here is the word "arxiv", which is a popular open-source publishing site for machine learning papers. It is unclear if this keyword was extracted due to true relevance (perhaps papers discussing the merits of open publishing), or if it simply extracted from the reference section of the papers, in which case it may be spurious.

Looking at the timing requirements, the average training time for the model is 632.1 seconds (over 5 training iterations), and the average query response time is 214 milliseconds. These results are in line with the specifications for the model.

To validate the usefulness of the model, the following procedure was used, based on the techniques presented in [17]: five topics related to the test corpus were searched, and the top five papers were examined to verify if they were related to the searched topic, based on the expert knowledge of the tester.

Topic	Score (out of 5)
Bayesian regression	5
Meta-learning	5
Actor-critic methods	0
Neural networks	5
Graph networks	5

7.5 System-Level Integration

The final product is a system composed by integrating the front-end, server, database, and machine learning model subsystems. This section describes the interactions between these subsystems, allowing the design to achieve its desired functionalities.

Figure 22 show the system interactions that happen upon startup. The server's `api.py` script performs a call to `database.py`, which is responsible for calling the DB driver's `build_db()` function with the appropriate parameters, as specified by the user. Depending on the startup parameters, the DB driver will call wither `model.build()` to train a new machine learning model, or `model.load()` to load a pre-existing model.

Figure 23 shows the system interactions when the application is terminated. The server calls the DB driver's `close()` function through `database.py` to finish the driver's connection.

Figure 24 shows the system interactions that happen when building the visualization features. The front-end `ArticlePage.js` sends a `FETCH` request to `api.py` passing the reference paper's ID. The server then queries the database by paper ID, using both `"exact"` and `"related"`

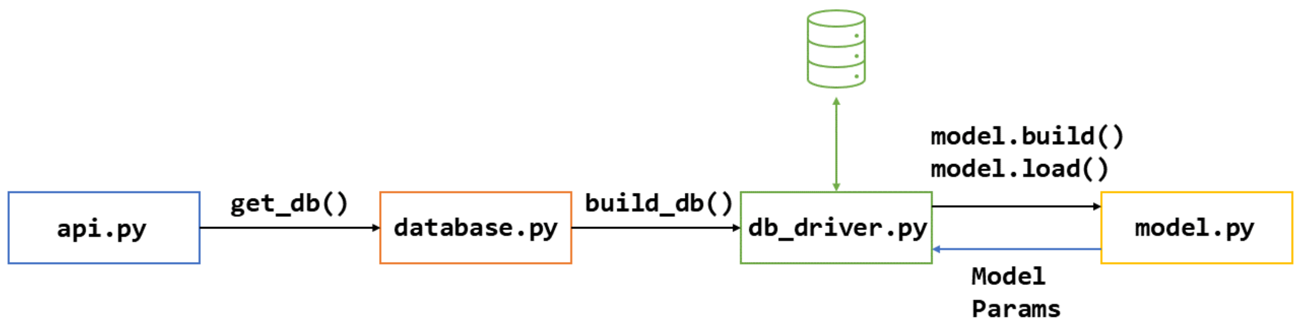


Figure 22: System interactions upon startup.

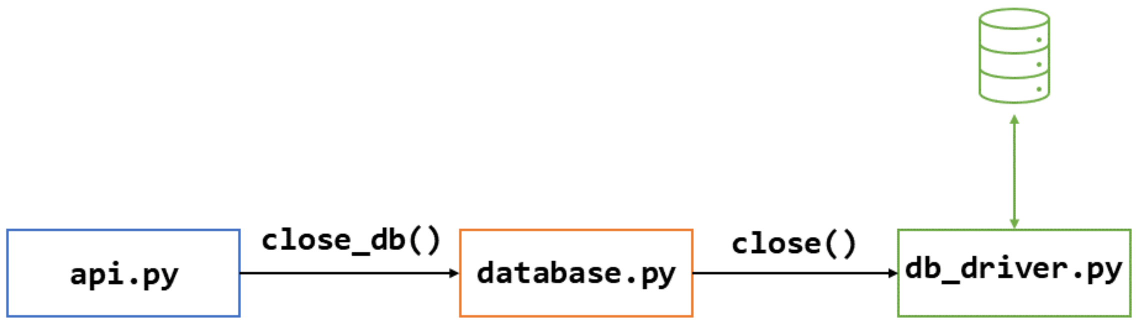


Figure 23: System interactions upon process finish.

queries, in order to get the desired paper, as well as its most closely related neighbours. The fetched papers are then returned to the front-end.

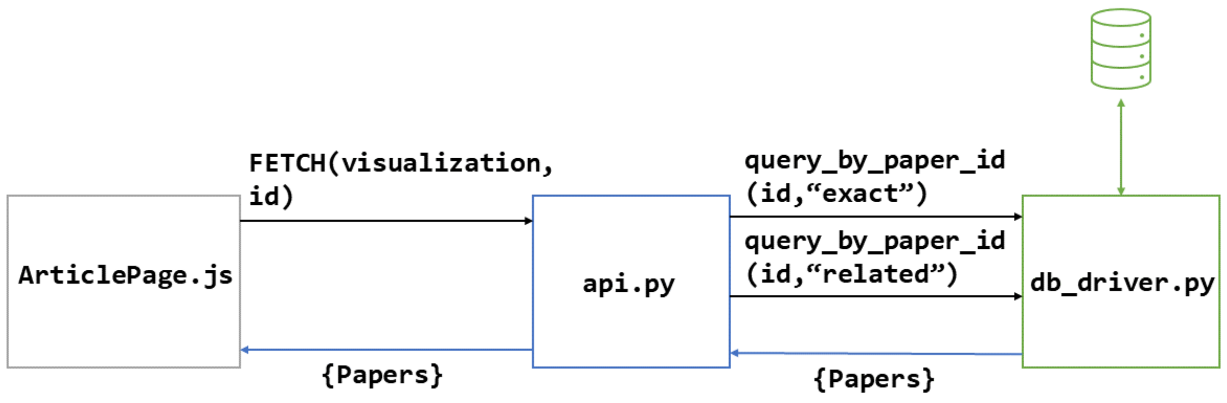


Figure 24: System interactions for visualization.

Figure 25 shows the system interactions that happen when queries are sent from the front-end, be it a query by author, title, paper_id, topic string, or topic index. The queries start from `ResultPage.js` and go through the server's `api.py`. The server is able to resolve the appropriate DB driver function to call depending on the query type requested by the front-end.

The requested papers are then sent from `db_driver.py` back to the front-end through `api.py`.

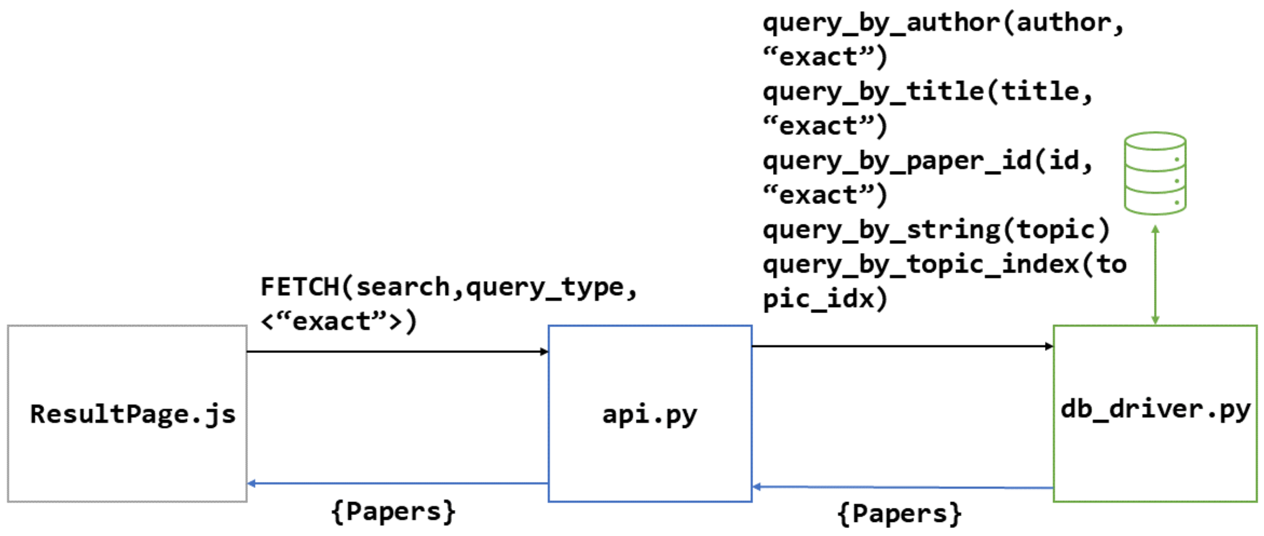
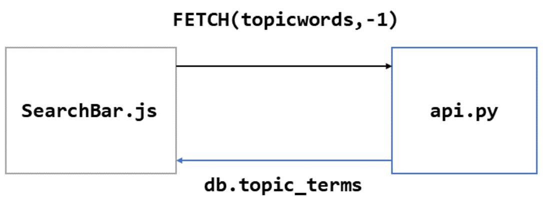
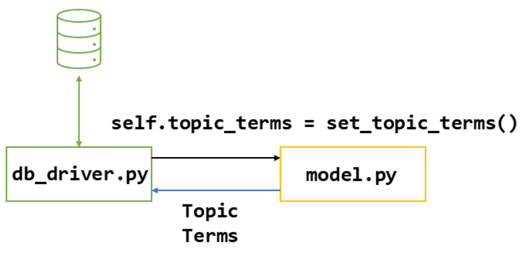

query_by_author(author, “exact”)
query_by_title(title, “exact”)
query_by_paper_id(id, “exact”)
query_by_string(topic)
query_by_topic_index(topic_idx)
FETCH(search,query_type, <“exact”>)
ResultPage.js
api.py
db_driver.py
{Papers}
{Papers}

Figure 25: System interactions during queries for papers in the database.

Figure 26 shows the system interactions necessary for obtaining and displaying the LDA topic terms in the front-end. The front-end `SearchBar.js` sends a “topicwords” query to `api.py` with an argument of -1. The argument -1 tells `api.py` to retrieve topic words for all of the topics. Queries for words of specific topics are also supported, although not used in the application. The topic terms are obtained by the DB driver upon initialization by making use of the model's `set_topic_terms()` function, and storing the topic terms in a class variable called `topic_terms`.



(a) System interactions when front-end queries for topic terms at run time.



(b) System interactions when the DB driver obtains the topic terms from the model when upon initialization.

Figure 26: System interactions necessary to set and query the LDA topic terms so they can be displayed in the front-end.

Finally, Figure 27 show the interactions that happen to allow a PDF file content to be loaded, in order for it to be displayed in the paper's article page. The front-end makes a request to the server, which directly accesses the file system in order to fetch the PDF file. It then reads its content, and sends it to the front-end, where the PDF file's content is displayed.

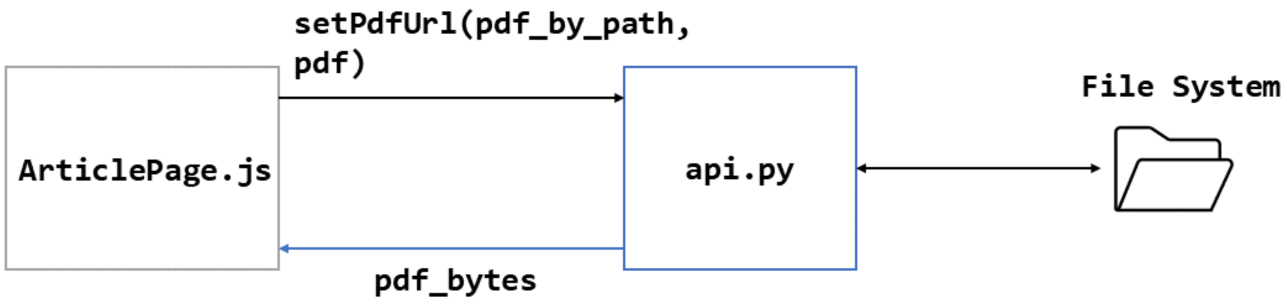
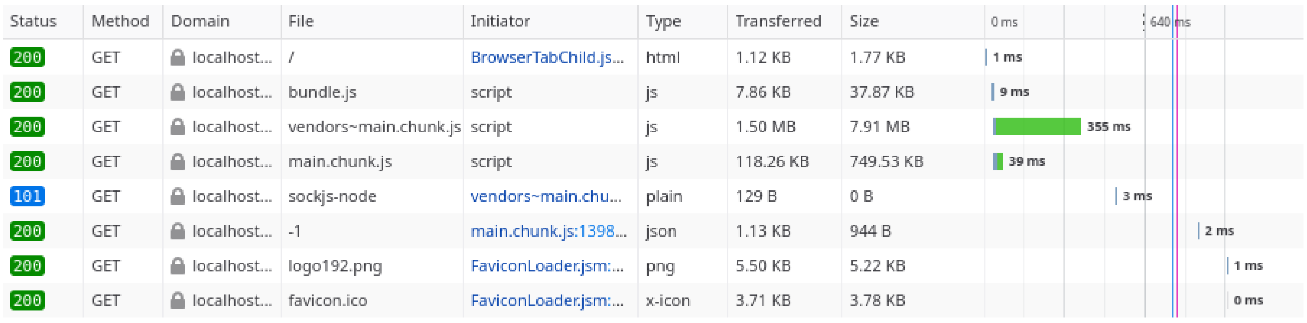


Figure 27: System interactions when obtaining a PDF file content.

7.5.1 System-level Performance

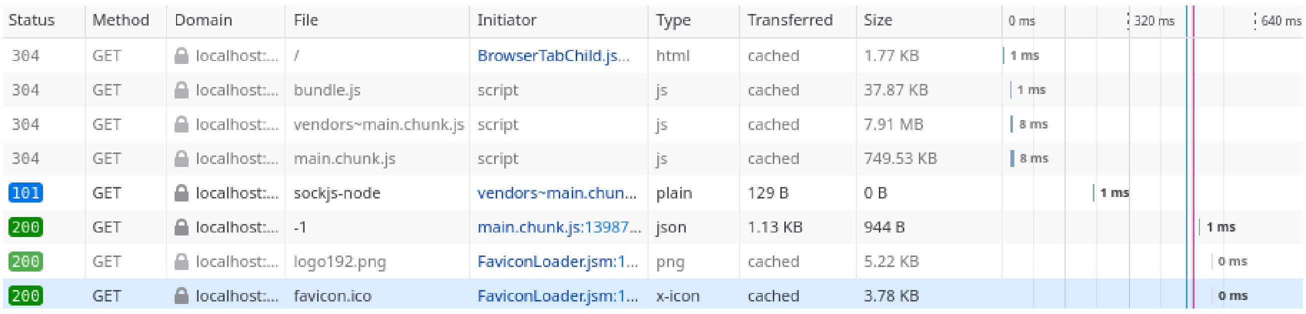
To analyze the network requirements for the functionality in the demo, the Firefox network analyzer was used to demonstrate network activity for sample pages of the home page, result page, and article page. It is important to keep in mind this is running all locally, so network times are neglected. The number of requests and size of data being transferred are looked at to ensure excessively large loads will not be applied to a network. Each page was loaded with caching and without to show worst-case and typical performance, many of the larger files such as bundle.js, vendors main.chunk.js, and sockjs-node would only need to be loaded once when navigating the application. This is a measurement of metric M4.

Home Page



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms … 640 ms
200	GET	localhost...	/	BrowserTabChild.js...	html	1.12 KB	1.77 KB	1 ms
200	GET	localhost...	bundle.js	script	js	7.86 KB	37.87 KB	9 ms
200	GET	localhost...	vendors~main.chunk.js	script	js	1.50 MB	7.91 MB	355 ms
200	GET	localhost...	main.chunk.js	script	js	118.26 KB	749.53 KB	39 ms
101	GET	localhost...	sockjs-node	vendors~main.chu...	plain	129 B	0 B	3 ms
200	GET	localhost...	-1	main.chunk.js:1398...	json	1.13 KB	944 B	2 ms
200	GET	localhost...	logo192.png	FaviconLoader.jsm:...	png	5.50 KB	5.22 KB	1 ms
200	GET	localhost...	favicon.ico	FaviconLoader.jsm:...	x-icon	3.71 KB	3.78 KB	0 ms

Figure 28: Home Page Network activity

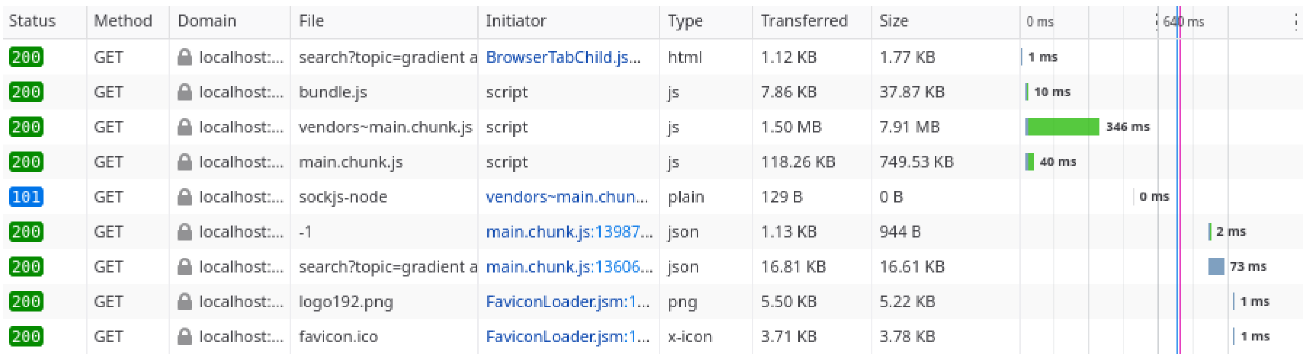


Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms 320 ms 640 ms
304	GET	localhost:...	/	BrowserTabChild.js...	html	cached	1.77 KB	1 ms
304	GET	localhost:...	bundle.js	script	js	cached	37.87 KB	1 ms
304	GET	localhost:...	vendors~main.chunk.js	script	js	cached	7.91 MB	8 ms
304	GET	localhost:...	main.chunk.js	script	js	cached	749.53 KB	8 ms
101	GET	localhost:...	sockjs-node	vendors~main.chun...	plain	129 B	0 B	1 ms
200	GET	localhost:...	-1	main.chunk.js:13987...	json	1.13 KB	944 B	1 ms
200	GET	localhost:...	logo192.png	FaviconLoader.jsm:1...	png	cached	5.22 KB	0 ms
200	GET	localhost:...	favicon.ico	FaviconLoader.jsm:1...	x-icon	cached	3.78 KB	0 ms

Figure 29: Home Page Network activity with caching

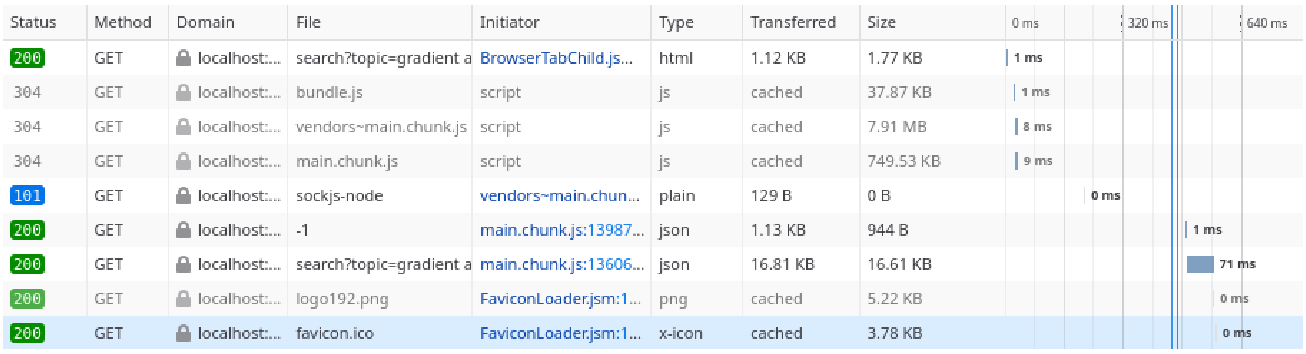
We can see that the file transfer that is by far the largest is vendors main.chunk.js at 7.91 MB, luckily this file can be cached and only needs to be downloaded once. The variable data that is retrieved from the server is file "-1" at around 1 KB, this is well within the limits of reasonable.

Result Page



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms 640 ms
200	GET	localhost:...	search?topic=gradient a	BrowserTabChild.js...	html	1.12 KB	1.77 KB	1 ms
200	GET	localhost:...	bundle.js	script	js	7.86 KB	37.87 KB	10 ms
200	GET	localhost:...	vendors~main.chunk.js	script	js	1.50 MB	7.91 MB	346 ms
200	GET	localhost:...	main.chunk.js	script	js	118.26 KB	749.53 KB	40 ms
101	GET	localhost:...	sockjs-node	vendors~main.chun...	plain	129 B	0 B	0 ms
200	GET	localhost:...	-1	main.chunk.js:13987...	json	1.13 KB	944 B	2 ms
200	GET	localhost:...	search?topic=gradient a	main.chunk.js:13606...	json	16.81 KB	16.61 KB	73 ms
200	GET	localhost:...	logo192.png	FaviconLoader.jsm:1...	png	5.50 KB	5.22 KB	1 ms
200	GET	localhost:...	favicon.ico	FaviconLoader.jsm:1...	x-icon	3.71 KB	3.78 KB	1 ms

Figure 30: Result Page Network activity

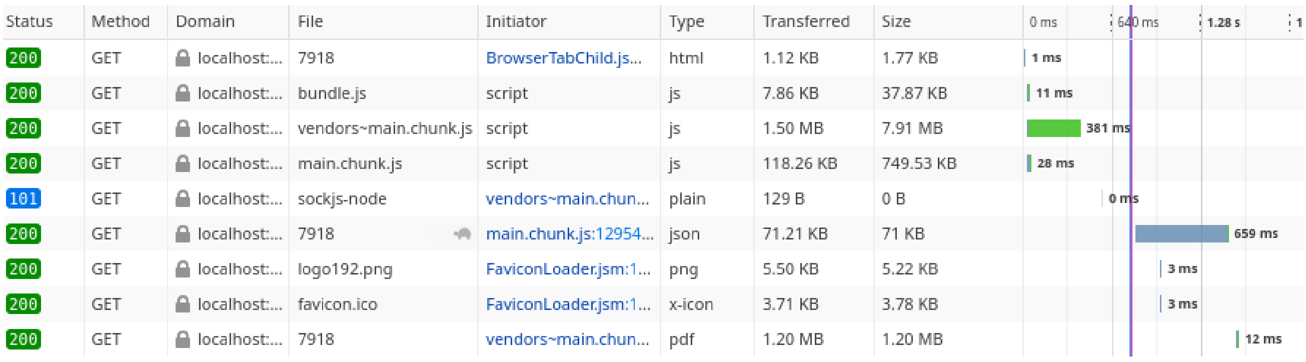


Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms 320 ms 640 ms
200	GET	localhost:...	search?topic=gradient a	BrowserTabChild.js...	html	1.12 KB	1.77 KB	1 ms
304	GET	localhost:...	bundle.js	script	js	cached	37.87 KB	1 ms
304	GET	localhost:...	vendors~main.chunk.js	script	js	cached	7.91 MB	8 ms
304	GET	localhost:...	main.chunk.js	script	js	cached	749.53 KB	9 ms
101	GET	localhost:...	sockjs-node	vendors~main.chun...	plain	129 B	0 B	0 ms
200	GET	localhost:...	-1	main.chunk.js:13987...	json	1.13 KB	944 B	1 ms
200	GET	localhost:...	search?topic=gradient a	main.chunk.js:13606...	json	16.81 KB	16.61 KB	71 ms
200	GET	localhost:...	logo192.png	FaviconLoader.jsm:1...	png	cached	5.22 KB	0 ms
200	GET	localhost:...	favicon.ico	FaviconLoader.jsm:1...	x-icon	cached	3.78 KB	0 ms

Figure 31: Result Page Network activity with caching

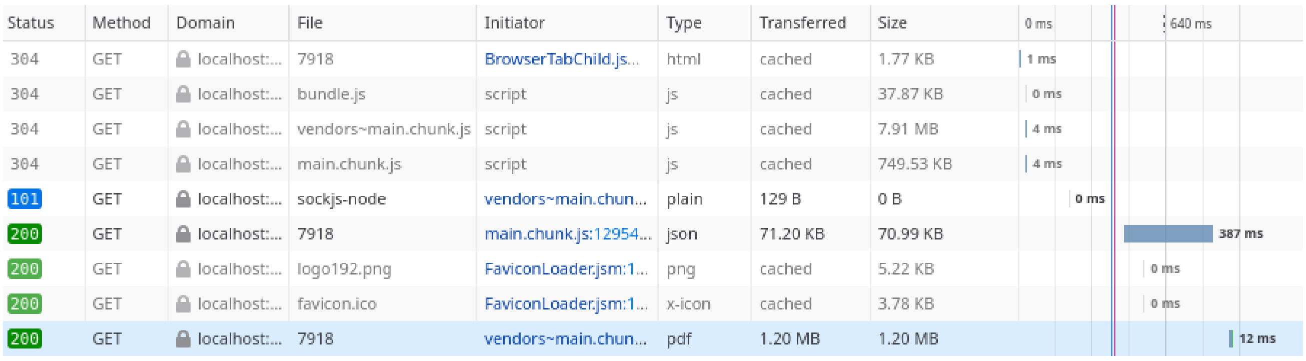
Again we see the common network activity, in addition there is "-1" and "search?*=*". 16.61 KB is transferred, which depends on the result size returned, in this case it is 25 results. This size should be approximately linearly related to the number of results returned.

Article Page



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms 640 ms 1.28 s
200	GET	localhost:...	7918	BrowserTabChild.js...	html	1.12 KB	1.77 KB	1 ms
200	GET	localhost:...	bundle.js	script	js	7.86 KB	37.87 KB	11 ms
200	GET	localhost:...	vendors~main.chunk.js	script	js	1.50 MB	7.91 MB	381 ms
200	GET	localhost:...	main.chunk.js	script	js	118.26 KB	749.53 KB	28 ms
101	GET	localhost:...	sockjs-node	vendors~main.chun...	plain	129 B	0 B	0 ms
200	GET	localhost:...	7918	main.chunk.js:12954...	json	71.21 KB	71 KB	659 ms
200	GET	localhost:...	logo192.png	FaviconLoader.jsm:1...	png	5.50 KB	5.22 KB	3 ms
200	GET	localhost:...	favicon.ico	FaviconLoader.jsm:1...	x-icon	3.71 KB	3.78 KB	3 ms
200	GET	localhost:...	7918	vendors~main.chun...	pdf	1.20 MB	1.20 MB	12 ms

Figure 32: Article Page Network activity



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms 640 ms
304	GET	localhost:...	7918	BrowserTabChild.js...	html	cached	1.77 KB	1 ms
304	GET	localhost:...	bundle.js	script	js	cached	37.87 KB	0 ms
304	GET	localhost:...	vendors~main.chunk.js	script	js	cached	7.91 MB	4 ms
304	GET	localhost:...	main.chunk.js	script	js	cached	749.53 KB	4 ms
101	GET	localhost:...	sockjs-node	vendors~main.chun...	plain	129 B	0 B	0 ms
200	GET	localhost:...	7918	main.chunk.js:12954...	json	71.20 KB	70.99 KB	387 ms
200	GET	localhost:...	logo192.png	FaviconLoader.jsm:1...	png	cached	5.22 KB	0 ms
200	GET	localhost:...	favicon.ico	FaviconLoader.jsm:1...	x-icon	cached	3.78 KB	0 ms
200	GET	localhost:...	7918	vendors~main.chun...	pdf	1.20 MB	1.20 MB	12 ms

Figure 33: Article Page Network activity with caching

Again there is the common network activity. For this page a request to "(server)/visualization" is made, which returns all the information required to create the visualization and the list of related papers. It also provides the information to acquire the PDF document from the server. The network analyzer shows a turtle by this activity suggesting it is slow to respond, i.e. response time is over 500 ms. This makes sense as in addition to finding the 100 most related papers to the current paper, the vectors in these papers must be processed to reduce each one to a 2-D vector so that they can be plotted on screen. This should not vary very much since there will always be only 100 papers to process, so this is acceptable.

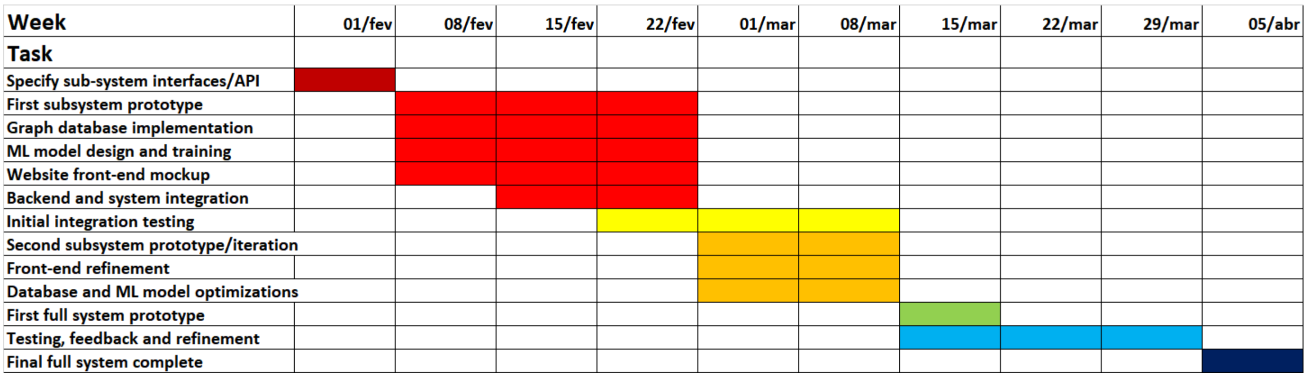
Each page with exception of the article page has a total response time under 500 ms when caching is used. We find this is acceptable as the article page response time is not significantly over this threshold, and this response time should not vary unless more than 100 papers are displayed in the visualization. So G1 is considered passed.

Each component of the total application communicates through ports, meaning that they can be run on a server, meaning G4 is achieved.

8 Resources

8.1 Time Management

One of the most valuable resources the team has available, in a project with such a short duration, is time. Below is a Gantt Chart illustrating the initial breakdown for each of the major project stages.



Week	01/fev	08/fev	15/fev	22/fev	01/mar	08/mar	15/mar	22/mar	29/mar	05/abr
Task										
Specify sub-system interfaces/API	🟥									
First subsystem prototype		🟥	🟥	🟥						
Graph database implementation		🟥	🟥	🟥						
ML model design and training		🟥	🟥	🟥						
Website front-end mockup		🟥	🟥	🟥						
Backend and system integration			🟥	🟥						
Initial integration testing				🟨	🟨	🟨				
Second subsystem prototype/iteration					🟧	🟧				
Front-end refinement					🟧	🟧				
Database and ML model optimizations					🟧	🟧				
First full system prototype							🟩			
Testing, feedback and refinement							🟦	🟦	🟦	
Final full system complete										⬛

Figure 34: Gantt Chart allocating time between different project stages

As shown in the chart, the project can be broken down into three major phases: first subsystem prototype, second subsystem prototype, and final testing and refinement.

In the first subsystem prototype phase, each subsystem - database, ML model, and front-end - will be developed to an early functional stage and they will be integrated, leading to the first project deliverable: the first design prototype. It is estimated that this phase will take approximately 3 weeks.

Next, in the second subsystem prototype phase, the website display and front-end will be refined based on feedback from the client, and the database and ML model subsystems will be refined and optimized. Notice that the main goal of the first subsystem prototype phase is functionality, while in the second phase the focus will be on optimization, e.g. making the database queries faster, improving the accuracy of the ML model, ensuring the communication between the subsystems happens with little overhead, and optimizing any bottlenecks that are identified. The second prototype deliverable will be ready by the end of this phase.

Finally, in the final testing and refinement phase, extensive testing of the design will be conducted to correct any bugs and flaws that may still be present in the system. Another round of feedback from the client and users will be obtained and used as a reference for final changes to the design. Any remaining changes and fixes to the design will be performed, and the product will be presented in its final form to the client.

8.2 Role Breakdown

In the early-mid stages of the project, each team member is responsible for one particular aspect of the end product (see Section 9 for info on team members). Daniel L. is in charge of the database system; this includes researching for an ideal database type for our application and setting it up. Saad H. is in charge of the back-end server for the application; this includes interfacing between the database/model and the front-end. Jeremy H. is in charge of the front-end; this includes creating a functional UI and displaying our product's capabilities to the end user. Kamran H. is in charge of the machine learning; this includes extracting relevant information (particularly topic information) from a corpus of documents.

While this is the initial role breakdown, we expect the UI design and machine learning aspects to include the most work. As such, as the project progresses, Saad will flex to assist Jeremy in the front-end work while Daniel will flex to assist Kamran in the machine learning. Implicitly, everyone will be involved in the integration aspect between components.

Furthermore, while each member owns a particular aspect of the project, each member shares their thoughts, ideas, accomplishments, and plans with the rest of the team in regular meetings (see Section 10 for more on team dynamics).

8.3 Risk Management

Each of the three phases presented above is associated with potential risks.

In the first phase, it is assumed that each of the three subsystems: database, ML model, and website front-end will be in an early functional form within 3 weeks, leading to the first prototype. This phase is crucial as the early prototype is to be used to acquire valuable and indispensable feedback from the client. While the likelihood that any of the three systems will not be in an early functional form in three weeks is considered low to moderate, its impact would be considerable as it would prevent a functional initial prototype. To mitigate this risk, the team will closely monitor the progress of each subsystem and allocate its human resources accordingly.

As the second phase of the project focuses on optimization, the main risk associated with it is that one of the database or ML subsystems will present a subpar performance. The likelihood of this risk is considered low as the database design chosen by the team - a graph database - has been already validated by the user as a feasible, efficient, and well-suited solution to the problem, and thus is expected to perform well, and standard existing ML implementations will be used, thus reducing the risk of low performance. In the unlikely case any of these subsystems performs badly and cannot be optimized, the design is still expected to be functional and will only suffer from latency problems, thus making this risk a low-impact one.

Finally, in the third phase of the design, a risk is associated with any changes made in the final stages of the design, as these may introduce failures and little time may be left to correct any such failures. To mitigate such risk, changes at this stage will be limited to essential and low-risk modifications.

8.4 Change Management

Some of our project goals may be ambitious and as such it is possible that changes to the scope and timeline of the project may happen. To account for this, our design will be made as modular as possible so that many features do not directly rely on the existence of another feature (unless perhaps that feature is core to our product). In regular meetings (see Section 8 for more on team dynamics) we consistently reassess whether our goals are feasible to allow for re-scoping as early as possible.

We identified three main design features early on that would be nice to have but are not crucial to the product, with verification of the latter by our client. The first was the use of optical character recognition technology to extract text from PDFs, as some of them may be scanned images. The second was an all-in-one search bar, that could determine whether a user's search query was for a title, author, or title, even though these are separate queries in the database. The third was to host our database, machine learning model, and web app on a dedicated server for access online, as opposed to simply hosting everything locally.

While these features add value to the design, they are more "nice-to-haves" and are not critical for the scope of our project. We originally planned to try to incorporate these features in our final design, but as the project progressed we have decided to not include these features as their anticipated benefit-time ratio is low. Owing to our change management plan and the modular implementation of our design thus far, the removal of these aspects did not cause any setbacks in development.

Furthermore, in terms of changes to the timeline, the second prototype iteration can be shortened in case the first prototype development takes longer than planned. While this phase is needed to enhance the performance and refine the appearance of the design, it can be removed if absolutely necessary in favor of the first prototype iteration, where functionality is to be achieved.

8.5 Hardware Resources

For the actual deployment of this service, the servers would likely be hosted by the stakeholders, given the potentially sensitive information that the application is dealing with.

For the purposes of this proof-of-concept product, all development and testing will be done locally. The PDF data set is not excessively large (15 GB) and the machine learning model is not too intensive so this is feasible. This project focuses on providing useful features so time was invested on that front as opposed to getting the product running on 3rd party servers and/or across the internet.

9 Structure

9.1 Meet the Team - Skills and Expertise

The team has four members: Saad Hussain, Jeremy Hunt, Daniel Leal and Kamran Ramji. Each team member brings a critical set of skills to the team.

Saad brings extensive software development experience. His skill set includes the languages JavaScript, Python and Golang, as well as tools such as Pytorch, Express, Docker and Kubernetes. Through his PEY experience at IBM, he brings extensive experience with distributed, cloud-computing systems, including the tools Jaeger and OpenTracing. His machine learning background includes an autonomous lane-tracking program, a data imputation program, and a real/fake news classifier, all built using PyTorch.

Jeremy comes to the team with experience in scripting, data processing and web development. He is familiar with Python and JavaScript, two scripting languages central to many machine learning and web development libraries. Jeremy has used SQL, which is critical in database design. Furthermore, he is well-acquainted with React, a JavaScript library that will help us build a seamless user interface. During his PEY, Jeremy scripted quality-assurance testing in Perl. While not directly related to this project, the debug and critical analysis skills he developed will be a pillar of the team's performance.

Conducting a thesis on pre-training deep neural networks, Daniel brings an extensive machine learning background. He has experience with Python, as well as SQL. As previously mentioned, SQL is essential for database design, and so this skill will be particularly useful to the team. Daniel is familiar with TensorFlow, which is another commonly used machine-learning toolkit. During his PEY, he worked at Intel in the Programmable Solutions Group.

Kamran brings a strong machine learning background, as well as a strong data processing skillset. He has extensive experience in Python, and some experience in Java. He is familiar with PyTorch and TensorFlow, two common machine-learning toolkits. Furthermore, he has the fundamentals of SQL from his coursework. During his PEY at Apple, on the iPhone Hardware team, Kamran developed strong scripting skills for test automation, as well as strong debug and critical analysis abilities.

Overall the team is familiar with the following programming languages that may be useful during the course of this project:

- Python
- C
- C++
- Java
- JavaScript
- SQL
- MATLAB

The team has experience with the following specific tools, that can be leveraged:

- Pytorch
- TensorFlow
- node.js
- React
- Numpy
- Pandas
- jQuery
- AngularJS
- Flask
- Express
- Docker
- Kubernetes

All of these skills are related to some component of the project, in either distributed systems, data-processing, machine learning, web and UI development.

Lastly, to highlight the diversity of the team, the following natural languages are spoken:

- English
- French
- Spanish
- Portuguese
- Urdu
- Cantonese

9.2 Team Charter

The purpose of this team charter is to put in place measures to maintain equity, inclusion, accountability and fairness. The core values of the team are:

- Integrity: all team members should act with integrity as outlined in the Code of Ethics in the Professional Engineers Act [47].
- Honesty: in their dealings with other team members, clients and course staff, all team members will be honest and forthright, in order to promote dialogue and the free flow of information.
- Consensus: whenever possible, team decisions are made through consensus, however, in this fast-paced and high-pressure environment, team members are expected to act in the best interest of the team if consultation is not possible.
- Solidarity: decisions made by team members within the scope of this project, represent the interests and intentions of the entire team; team members should refrain from calling out individual team members to external stakeholders for perceived errors; the entire team is accountable as a group.
- Internal accountability: frequent team meetings are an open and informal dialogue for team members to discuss past decisions, lessons learned, and how to proceed in the future.
- Ownership: this team is a group of equal peers who are all responsible for the project as a whole; there is no designated leader or spokesperson, unless one must be appointed for a particular task for the sake of efficiency or clarity when dealing with external stakeholders.

The team has no tolerance for discrimination on the basis of any protected characteristic under federal and provincial law, including race, gender, national origin, sexual orientation or religion (or lack thereof).

10 Dynamics

The team has several sources to lean on, to gather obtain necessary information and develop skill sets. Due to the virtual nature of the team, all of the information communicated and/or made available will be transmitted via the internet. Some of the primary sources are:

- Client: Ph.D. student with extensive experience (including first-hand Capstone experience).
- Other team members: often experts in their respective domains of interest.
- Online documentation: for various libraries and software tools.
- Librarians: for academic research in the various areas of design.
- Teaching team: for course-related information and advice on the design process.

10.1 Communication and Stakeholder Management Plan

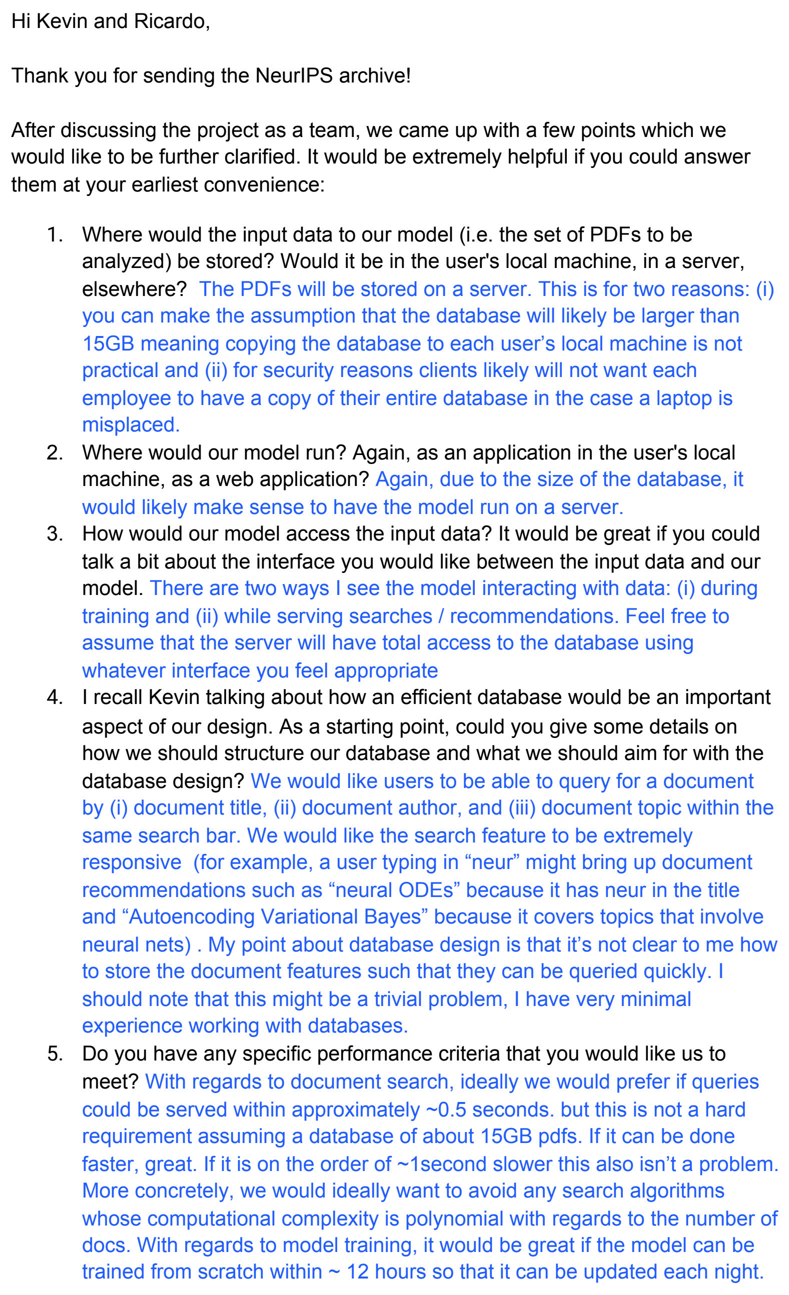
Communication will happen through several channels. The choice of channel depends on the audience, and context. Regular (at least weekly, often more) intra-team communication primarily occurs on Facebook Messenger, both over chat and using the video call functionality for meetings. Communications with the client take place primarily over email, with Zoom video call for meetings. Team members are encouraged to turn on their camera whenever possible during virtual meetings, to match the in-person environment as closely as possible. Additional stakeholders will be contacted primarily through email, and Zoom video conferencing will be used when / if additional meetings are organized.

A Appendix A: Stakeholder Interactions

This appendix presents logs of relevant interactions the group had with stakeholders. These discussions were used to inform design decisions, establish metrics and criteria, and gather feedback on proposed design solutions.

A.1 E-mail from Infera AI: specification of requirements

On January 29th, 2021, the team sent an e-mail requesting for clarifications on aspects of the design, as well as requirements from Infera AI. The responses are shown below, in blue.



Hi Kevin and Ricardo,

Thank you for sending the NeurIPS archive!

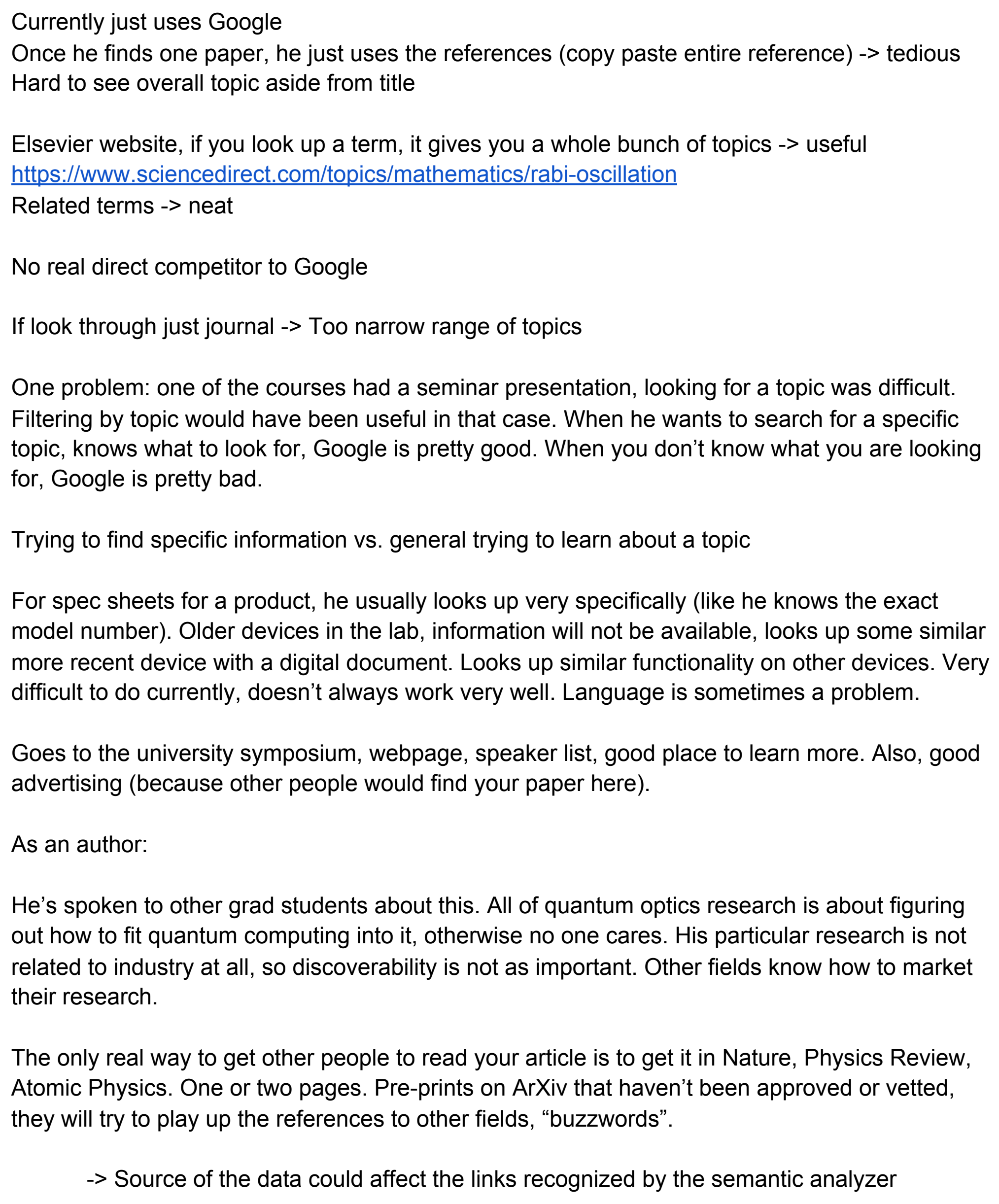
After discussing the project as a team, we came up with a few points which we would like to be further clarified. It would be extremely helpful if you could answer them at your earliest convenience:

1. Where would the input data to our model (i.e. the set of PDFs to be analyzed) be stored? Would it be in the user's local machine, in a server, elsewhere? The PDFs will be stored on a server. This is for two reasons: (i) you can make the assumption that the database will likely be larger than 15GB meaning copying the database to each user's local machine is not practical and (ii) for security reasons clients likely will not want each employee to have a copy of their entire database in the case a laptop is misplaced.
2. Where would our model run? Again, as an application in the user's local machine, as a web application? Again, due to the size of the database, it would likely make sense to have the model run on a server.
3. How would our model access the input data? It would be great if you could talk a bit about the interface you would like between the input data and our model. There are two ways I see the model interacting with data: (i) during training and (ii) while serving searches / recommendations. Feel free to assume that the server will have total access to the database using whatever interface you feel appropriate
4. I recall Kevin talking about how an efficient database would be an important aspect of our design. As a starting point, could you give some details on how we should structure our database and what we should aim for with the database design? We would like users to be able to query for a document by (i) document title, (ii) document author, and (iii) document topic within the same search bar. We would like the search feature to be extremely responsive (for example, a user typing in "neur" might bring up document recommendations such as "neural ODEs" because it has neur in the title and "Autoencoding Variational Bayes" because it covers topics that involve neural nets) . My point about database design is that it's not clear to me how to store the document features such that they can be queried quickly. I should note that this might be a trivial problem, I have very minimal experience working with databases.
5. Do you have any specific performance criteria that you would like us to meet? With regards to document search, ideally we would prefer if queries could be served within approximately ~0.5 seconds. but this is not a hard requirement assuming a database of about 15GB pdfs. If it can be done faster, great. If it is on the order of ~1second slower this also isn't a problem. More concretely, we would ideally want to avoid any search algorithms whose computational complexity is polynomial with regards to the number of docs. With regards to model training, it would be great if the model can be trained from scratch within ~ 12 hours so that it can be updated each night.

Figure 35: E-mail exchange with Infera AI regarding design objectives and requirements.

A.2 Logs from an interview with a graduate student

On February 18th, 2021, the team interviewed a graduate student, as someone that often searches for academic and technical papers and also publishes works of their own. The logs taken by the team during the interview are shown below.



Currently just uses Google
Once he finds one paper, he just uses the references (copy paste entire reference) -> tedious
Hard to see overall topic aside from title

Elsevier website, if you look up a term, it gives you a whole bunch of topics -> useful
https://www.sciencedirect.com/topics/mathematics/rabi-oscillation
Related terms -> neat

No real direct competitor to Google

If look through just journal -> Too narrow range of topics

One problem: one of the courses had a seminar presentation, looking for a topic was difficult. Filtering by topic would have been useful in that case. When he wants to search for a specific topic, knows what to look for, Google is pretty good. When you don't know what you are looking for, Google is pretty bad.

Trying to find specific information vs. general trying to learn about a topic

For spec sheets for a product, he usually looks up very specifically (like he knows the exact model number). Older devices in the lab, information will not be available, looks up some similar more recent device with a digital document. Looks up similar functionality on other devices. Very difficult to do currently, doesn't always work very well. Language is sometimes a problem.

Goes to the university symposium, webpage, speaker list, good place to learn more. Also, good advertising (because other people would find your paper here).

As an author:

He's spoken to other grad students about this. All of quantum optics research is about figuring out how to fit quantum computing into it, otherwise no one cares. His particular research is not related to industry at all, so discoverability is not as important. Other fields know how to market their research.

The only real way to get other people to read your article is to get it in Nature, Physics Review, Atomic Physics. One or two pages. Pre-prints on ArXiv that haven't been approved or vetted, they will try to play up the references to other fields, "buzzwords".

-> Source of the data could affect the links recognized by the semantic analyzer

Figure 36: Logs taken by the team during an interview with a graduate student stakeholder.

References

[1] Daniel Pinheiro Leal, Jeremy Hunt, Kamran Ramji, and Saad Hussain. *InferaCapstone GitHub Repository.* URL: `https://github.com/DanielPinheiroLeal/InferaCapstone`.

[2] Neural Information Processing Systems Foundation. URL: `https://nips.cc/`.

[3] E. F. Codd. "A relational model of data for large shared data banks". In: *Communications of the ACM* 13.6 (1970), pp. 377–387. DOI: `10.1145/362384.362685`.

[4] *ISO/IEC 9075-1:2016 - Information technology: database languages: SQL.* ISO/IEC, 2016.

[5] Ali Davoudian, Liu Chen, and Mengchi Liu. "A Survey on NoSQL Stores". In: *ACM Computing Surveys* 51.2 (2018), pp. 1–43. DOI: `10.1145/3158661`.

[6] Gerard Salton and Michael J. McGill. *Introduction to modern information retrieval.* McGraw-Hill, 1983.

[7] *An Empirical Evaluation of Models of Text Document Similarity.* 2005.

[8] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". In: May 2010, pp. 45–50. DOI: `10.13140/2.1.2393.1847`.

[9] omnisci. *Client-Server.* URL: `https://www.omnisci.com/technical-glossary/client-server`.

[10] Steve Cope. *The TCP/IP Model and Protocol Suite Explained for Beginners.* URL: `http://www.steves-internet-guide.com/internet-protocol-suite-explained/`.

[11] GeeksforGeeks. *TCP/IP Model.* URL: `https://www.geeksforgeeks.org/tcp-ip-model/`.

[12] Mozilla Developer Network. *HTTP request methods.* URL: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods`.

[13] Mozilla Developer Network. *HTTP response status codes.* URL: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Status`.

[14] Red Hat. *What is a REST API?* URL: `https://www.redhat.com/en/topics/api/what-is-a-rest-api`.

[15] the free encyclopedia Wikipedia. *Representational state transfer.* URL: https://en.wikipedia.org/wiki/Representational_state_transfer.

[16] the free encyclopedia Wikipedia. *JSON.* URL: https://en.wikipedia.org/wiki/JSON.

[17] Susan T. Dumais. "Latent semantic analysis". In: *Annual Review of Information Science and Technology* 38.1 (2004), pp. 188–230. DOI: https://doi.org/10.1002/aris.1440380105. eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/aris.1440380105. URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440380105.

[18] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent Dirichlet Allocation". In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 993–1022. ISSN: 1532-4435.

[19] Python Software Foundation. *Python downloads.* URL: https://www.python.org/downloads/.

[20] OpenJS Foundation. *Nodejs downloads.* URL: https://nodejs.org/en/download/.

[21] *System requirements.* URL: https://neo4j.com/docs/operations-manual/current/installation/requirements/#deployment-requirements.

[22] *Neo4j Download Center.* Mar. 2021. URL: https://neo4j.com/download-center/?fbclid=IwAR305o9Nv65OWeNQwIDuUnC6cEh-hDG0jALhIBd-HI9ZUfg87o7pl5QWAhE#community.

[23] *Installation - Operations Manual.* URL: https://neo4j.com/docs/operations-manual/current/installation/?fbclid=IwAR2fHhHJEAvUFejSMe5A-gA9rOkiGWUXZtG3T-5XkbT2huKIynRKRZpAmmY.

[24] For trial users of Neo4j Enterprise Edition. *Neo4j Download Center.* Mar. 2021. URL: https://neo4j.com/download-center/?fbclid=IwAR0B1H7aCzBY2HMRsgn0oa5tJRpxUABnfYPiz9T_4J3b9HVUnG_bgeY2OcI#algorithms.

[25] *Chapter 2. Installation.* URL: https://neo4j.com/docs/graph-data-science/current/installation/?fbclid=IwAR1u76m1o3Ps6-3VKIQRVafWxt4XI8zAgQ07Ahxzb644m48Tthw7x6f4pRs.

[26] PyPI. *Python framework for fast Vector Space Modelling.* URL: https://pypi.org/project/gensim/.

[27] PyPI. *Apache Tika Python library.* URL: https://pypi.org/project/tika/.

[28] Pallets. *Installation.*
URL: https://flask.palletsprojects.com/en/1.1.x/installation/#.

[29] PyPI. *A Flask extension adding a decorator for CORS support.*
URL: https://pypi.org/project/Flask-Cors/.

[30] scikit-learn developers. *Installing scikit-learn.*
URL: https://scikit-learn.org/stable/install.html.

[31] *ICLR Paper Explorer.* URL: https://iclr.cc/virtual_2020/paper_vis.html.

[32] *NeurIPS 2020 Paper Explorer.*
URL: https://neurips.cc/virtual/2020/protected/paper_vis.html.

[33] Bettina Grün and Kurt Hornik. “topicmodels: An R Package for Fitting Topic Models”. In: *Journal of Statistical Software* 40.13 (2011), pp. 1–30. DOI: 10.18637/jss.v040.i13.

[34] Hanna M. Wallach, Iain Murray, Ruslan Salakhutdinov, and David Mimno. “Evaluation Methods for Topic Models”. In: *Proceedings of the 26th Annual International Conference on Machine Learning.* New York, NY, USA: Association for Computing Machinery, 2009, pp. 1105–1112. ISBN: 9781605585161. URL: https://doi.org/10.1145/1553374.1553515.

[35] Diane Kelly Xi Niu. “The use of query suggestions during information search”. In: *Information Processing Management* 50 (1 Jan. 2014), pp. 218–234. DOI: 10.1016/j.ipm.2013.09.002.

[36] *Top 10 backend frameworks: Which is the best option for you?* Feb. 2021.
URL: https://blog.back4app.com/backend-frameworks/.

[37] *Neo4j Graph Platform – The Leader in Graph Databases.* Feb. 2021.
URL: https://neo4j.com/.

[38] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP).* 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[39] *Google Scholar.* URL: https://scholar.google.com/schhp?hl=en&as_sdt=0,5.

[40] *NTRS - NASA Technical Reports Server.* URL: https://ntrs.nasa.gov/search.

[41] *Introduction to OpenCV-Python Tutorials.*
URL: https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[43] *PyTorch Transformers - HuggingFace.* URL: https://pytorch.org/hub/huggingface_pytorch-transformers/?fbclid=IwAR09PawVYPP0DeafD1qpnbwviCUc_B8LNDVbx40SKScnCfzTa6i0_iu9drs.

[44] Christopher Olah. *Understanding LSTM Networks.* URL: https://colah.github.io/about.html.

[45] Campos Michael D. Harrison Paolo Masci. “Verification Templates for the Analysis of User Interface Software Design”. In: *IEEE Transactions on Software Engineering* 45 (8 Aug. 2019), pp. 802–822. DOI: 10.1109/TSE.2018.2804939.

[46] José Creissac Campos Michael D. Harrison Paolo Masci and Paul Curzon. “Verification of User Interface Software: The Example of Use-Related Safety Requirements and Programmable Medical Devices”. In: *IEEE Transactions on Human-Machine Systems* 47 (6 July 2017), pp. 834–846. DOI: 10.1109/THMS.2017.2717910.

[47] Government of Ontario. *Professional Engineers Act, R.R.O. 1990, REGULATION 941.* Dec. 2020. URL: https://www.ontario.ca/laws/regulation/900941#BK93.