Logic Mill - A Knowledge Navigation System

Sebastian Erhardt[1,*], Mainak Ghosh[1], Erik Buunk[1], Michael E. Rose[1] and Dietmar Harhoff[1]

[1]Max Planck Institute for Innovation and Competition, Marstallplatz 1, Munich, Bavaria, 80539, Germany

Abstract

Logic Mill is a scalable and openly accessible software system that identifies semantically similar documents within either one domain-specific corpus or multi-domain corpora. It uses advanced Natural Language Processing (NLP) techniques to generate numerical representations of documents. It leverages a large pre-trained language model to generate these document representations. The system focuses on scientific publications and patent documents and contains more than 200 million documents. It is easily accessible via a simple Application Programming Interface (API) or via a web interface. Moreover, it is continuously being updated and can be extended to text corpora from other domains. We see this system as a general-purpose tool for future research applications in the social sciences and other domains.

Keywords

Semantic Similarity, Vector Search, Patents, Publications, Document Encoder

1. Introduction

There is a growing need for tools that allow researchers to identify related documents within the same but also across different domains. With the ever-growing volume of scientific publications and patents, scholars find it burdensome to manage relevant documents and search for important prior contributions efficiently. Finding relevant documents plays a significant role in building coherent scientific arguments but is also important in assessing the use of scientific research outside academia [1, 2]. Patent examination is another field in which finding related documents and identifying prior art is essential.

In ex post analyses, researchers often rely on citation data to identify relations between documents. While citations are helpful in tracing citation networks and in uncovering important patterns in the production and diffusion of knowledge within the same corpus, they are typically limited when searching for relations across different corpora. Even within the same corpus, citations can be selective or even systematically biased (see [3, 4]). Finally, references may not exist for texts that are in the process of being created, so authors are faced with the challenge of identifying relevant references in the first place. Therefore, tools are needed that allow for processing and analyzing the textual contents of high-volume text corpora and establishing measures of relatedness (similarity) between them.

We refer to the system described here as a knowledge navigation system, since it facilitates the tracing of knowledge elements within and across text corpora (e.g., the corpus of all patents or all scientific publications). Previous attempts include [5] for scientific publications and [6] for patent documents, both of which make use of bag-of-words approaches. While such systems are sometimes referred to as recommender systems [7], recommendation of related documents is only one possible application of knowledge navigation (see below for a non-exhaustive list of use cases). However, these systems often use proprietary algorithms, usually focus on one domain corpus, are not openly accessible, or are not continuously being updated. A knowledge navigation system of the kind envisioned here should be capable of efficiently retrieving, storing, and processing hundreds of millions of documents. Moreover, it requires capabilities for fast detection of particularly similar documents within and across corpora.

An important problem that needs to be addressed is how to implement the concept of document similarity. This requires representing documents in a numerical form that computers can process, with the goal of generating similar representations for similar documents at a large scale. In the field of natural language processing (NLP), this process is known as document encoding.

There are various methods for representing documents numerically. Traditional NLP approaches like *TF-IDF* [8] are used widely in the literature. However, these traditional methods are not scalable, since an extension of the vocabulary requires a re-computation of all representations of the corpus. In addition, these approaches do not capture the semantics of the documents; that is, the meaning of words or the interpretation of sentences in context is lost.

Therefore, we propose *Logic Mill*, a software system aiming to satisfy the shortcomings of existing systems and approaches. The system creates document repre-

5th Workshop on Patent Text Mining and Semantic Technologies (PatentSemTech) 2024

*Corresponding author.

sebastian.erhardt@ip.mpg.de (S. Erhardt); mainak.ghosh@ip.mpg.de (M. Ghosh); erik.buunk@ip.mpg.de (E. Buunk); michael.rose@ip.mpg.de (M. E. Rose); dietmar.harhoff@ip.mpg.de (D. Harhoff)

sentations using modern NLP techniques and contains large document sets with pre-calculated encodings. It is easy to use and allows users to access and compare texts from different text corpora. Furthermore, it is scalable and built on non-proprietary algorithms. We regularly update the datasets based on their release schedule. Our objective is to provide a fast system of high accuracy that is openly accessible.

In the current version, the system encodes text documents using the *Pat SPECTER* document encoder [9], which leverages the bi-directional transformer architecture (BERT) [10]. This model, in combination with the database containing numerical representations of documents from different corpora (analogous to a vector search database), is the backbone for the *Logic Mill* system.

At present, we provide the numerical representations for the scientific articles of OpenAlex [11] and for the patent publications provided in DocDB database, maintained by the European Patent Office. To be precise, we use the titles and abstracts of these, since BERT-based models can process only 512 tokens at a time.

An important feature is that users can also feed their own text data to Logic Mill for encoding and obtaining similarity measures, within their data and between their data and the system-provided documents. They can thus link their own curated documents to patents and scientific publications according to textual similarity.

Logic Mill can be used in a number of different research applications, such as:

- Explore literature: Search for research papers, and find the best matches based on textual similarity to a paper in the database or to own text documents.
- Prior art search in patent examination: Look for previously granted patents or (not yet granted) patent applications that are similar to the focal one.
- Link patents to related scientific publications: Search for patents that the scientific publication might be based on or have a strong similarity to.
- Recommend citations and readings for new documents: Find documents that are very similar to a focal one and may be useful as a reference or reading.
- Assess the novelty of patents and publications: Check if a patent or publication is new or not by comparing it to prior texts. Documents that have few highly similar documents may be new or even unique.
- Trace concepts across domains and over time: Identify documents across domains (e.g. publications and patents) that are highly similar and possibly related.

2. Document Encoding

Document encoding in natural language processing (NLP) is a process for representing textual data in a numerical form. There are various approaches to encoding documents.

Bag of words Simple and fast procedures construct a vector whose binary elements indicate the presence of a word in a document ("binary term encoding"), the number of occurrences of a word in a document ("count matrix") or the weighted number of occurrences of a word in a document ("Term frequency-inverse document frequency (TF-IDF)"). These approaches have the drawback of not capturing the meaning or context of the words in the document – hence the common term bag-of-words. Furthermore, they are not scalable since the whole model must be retrained if a new word is added to the vocabulary and the length of each vector equals with the vocabulary size. From a computational perspective, they are inefficient since they generate sparse matrices where most elements equal 0.

Word embeddings Word embeddings are dense, fixed-size, and continuous-valued vector representations of words that capture the meaning of the words in the document. These word representations are learned via training over large corpora of textual data using methods such as *Word2Vec* [12], *GloVe* [13], or *FastText* [14]. The advent of these word embedding methods was a leap towards memory-efficient dense numerical representation for words and documents from the sparse representation of bag of words models. A baseline approach to representing a document is to average or sum the learned word embeddings of the words in a document.

Sentence/Paragraph Embeddings Sentence embeddings can be understood as an extension of the basic idea of word embeddings. Word embeddings are static representations of words that do not change even when multiple contexts are present in the document collection. However, the same word can have different meanings in different contexts. For example, the word *"bank"* can be a financial institute or can relate to a river bank. Hence, the neural network architecture such as Recurrent Neural Networks (*RNNs*) and Long Short-Term Memory (*LSTM*) [15] are used for representing the word to its true context dynamically with a notion that words appearing either before or after the focal word reveal the context around the focal word. *RNN* and *LSTM* also aid the vector representation of the context of the sentences or paragraph [16].

However, *RNN* and *LSTM* architectures cannot appropriately capture the meaning of the words at the beginning of a very long sentence or paragraph in its numerical

representation due to their sequential structure. The rise of deep neural networks and recent advancements in the field of NLP introduced the transformer architecture [17]. Transformers look at each word of a sentence together, unlike RNN and LSTM, and learn the degree to which the words reflect the context of that sentence using the so-called attention mechanism. The transformer architecture can translate the meaning of each word in a sentence through its network into the numerical representation of the sentence, also called sentence embedding. As stated before, a baseline approach to representing a document is to average or sum the learned sentence embeddings of that document.

BERT Language Model Bidirectional Encoder Representations from Transformer, *BERT* [10] is another recent development that uses transformer architecture. It exhibits all the traits of transformer architecture, meaning it learns the sense of the words of an input sentence, the context of that sentence, and the semantic relation between the words and the context. *BERT* being a "bidirectional" model, considers the context of a word from both sides (left and right) at the same time in a sentence, which makes this model effectively process long contiguous text sequences, such as entire paragraph, not limited to short phrase. The *BERT* architecture is designed with a limitation of 512 input tokens. This means that before the text is fed into the model, it is tokenized and truncated if necessary. Any model built on *BERT* must take this into account. Based on our own data, a word usually consists of ≈ 1.2 tokens.

SciBERT *SciBERT* is a *BERT* language model for tasks involving scientific publications [18]. It was trained on a large corpus of 1.14M scientific publications from computer science and the broad biomedical field. This model also outputs numerical representation like *BERT*.

Fine-tuning a general purpose *BERT* model on scientific papers produces more accurate results in this domain [18, 19] because it uses a domain-specific vocabulary. This also extends to similar domains, in this case, patents, where *SciBERT* model outperforms the original *BERT* for tasks such as IPC classification and similar patent finding [20]. *SciBERT* is thus particularly well-suited for tasks such as information extraction, document classification, and text representation in the scientific domain.

SPECTER *SPECTER* is an extension of *SciBERT* to encode scientific publications also with the help of inter-document relatedness [21]. In the scientific literature, citations signal relatedness, but this information is not used by *SciBERT*. *SPECTER* transfers the learned relatedness signal to the representation of a scientific article. During the application of this model, it generates similar embeddings for related scientific documents without knowing citation information. *SPECTER* was used in the initial version of *Logic Mill*.

Pat SPECTER *Pat SPECTER* is a language model to encode scientific publications and patents at the same time [9]. It is a *SPECTER* model (version: 2.0) fine-tuned on a curated patent dataset comprising 1.5M training instances (5 triplets for 300k focal patents) with credible patent citations. Thus, this model combines both domains, scientific publications, and patents. Our tests have shown that this model outperforms the previously mentioned models, as well as the *PaECTER* [9], for predicting patent-to-paper citations. Furthermore, it also outperforms traditional search methods, often used in information retrieval systems like *BM25*.

Pat SPECTER replaces *SPECTER* as our workhorse document encoder model.[1]

3. System Specifics

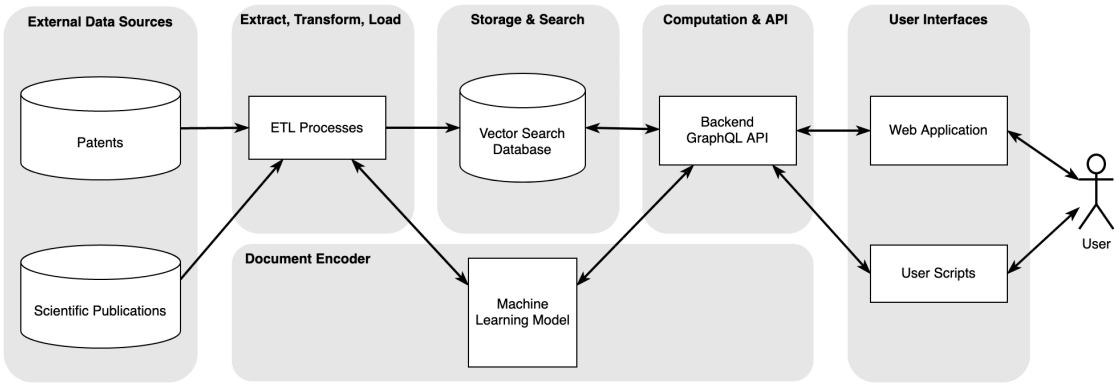
Logic Mill system is designed using a Microservice Architecture. This software design approach breaks down a large, monolithic application into smaller, independent components that can be developed, deployed, and maintained separately. Each microservice is a self-contained unit that performs a specific function and communicates with other microservices through well-defined interfaces, typically using APIs.

Figure 1 shows the multiple services of our system. From a high-level perspective, there are 6 distinct parts:

a) External Data Sources: where patent documents and scientific publications are obtained from
b) *Document Encoder*: transforms the text of the document into a numerical representation using a machine learning model
c) *Extract, Transform, Load (ETL)*: processes move documents from the external sources, process and store them
d) *Vector Search Database*: stores the computed numerical representation along with metadata and the text
e) *Backend with Web API*: propagate the user requests to the database
f) *User-Interface*: provides the users with a web application where they can interact with the API via our website or using their own scripts, for example in *Python* or *R*.

[1] There is one other *BERT*-based language model specifically trained on patents, namely *PatentBERT* [22]. Its task is to classify patents, and therefore, it is not suitable for our purpose.

Figure 1: Overview of the Architecture of Logic Mill



External Data Sources The system can be connected to various external data sources. The release version of the system retrieves patent documents and scientific publications from public and access-restricted sources.

Initially, *Logic Mill* relied on publicly available patent APIs from the EPO and the USPTO. For scientific publications, we relied on Semantic Scholar [23].

However, we decided to change the external data sources to streamline the processes and align with the preferences of the scientific community.

Scientific publications are now obtained from the OpenAlex [2][11] database, including title, abstracts, and additional metadata such as publication date, journal name, or Digital Object Identifier (DOI).

We obtain patent documents from the DocDB[3], the EPO's master documentation database. This database has worldwide coverage and contains bibliographic data, abstracts, and citations. Furthermore, it also maintains the DOCDB simple patent family.

Extract, Transform, Load The system obtains the raw documents from external data sources and processes them through various Extract, Transform, and Load (ETL) processes. In the first step, the textual content and the metadata are extracted and stored in a global data structure. The structure is independent of the document type (patent or scientific publication). In the second step, the document encoder encodes the text parts of the document and generates the numerical representation. Finally, the search database stores the numerical representation of a document along with the metadata and the text.

Document Encoder The document encoder is a machine-learning model that transforms text documents into a numerical representation. We use *Pat SPECTER* [9] to encode documents. The output of this model is a dense vector of 768 dimensions.

Since encoding all documents requires significant computing power, it was conducted on specialized hardware. On the one hand, we used desktop workstations with Nvidia graphics processing units (GPUs). For large bulk sets, we made use of high-performance cloud computing facilities. To allow for real-time inference, a CPU microservice was deployed in the cloud. It is connected to the system and accessible to end-users via the GraphQL API.

Storage & Search For the search and storage of documents with their numerical representation, ElasticSearch is used. This database is capable of full-text search and can be used in a distributed context, which is essential for scalability reasons.

Furthermore, ElasticSearch allows storing dense vectors that nearest-neighbor search algorithms can use. Exact Nearest Neighbor searchers are guaranteed to find the best solution but are inefficient and not scalable. Therefore, we use the approximate nearest neighbor searches (ANN), which trade-off precision for lower computational and resource burdens. ElasticSearch uses the Hierarchical Navigable Small World graphs (HNSW) [24] algorithm (as of version 8.0).[4] HNSW organizes vectors as a graph based on their similarity to each other. Together, this setup finds the most similar documents with a very high probability for any query document within milliseconds.

In our current setting, the cluster consists of 12 nodes with 8 vCPU cores, 128 GB of RAM, and 1 TB of SSD storage each. This setting is needed to allow for fast and efficient computation, because the RAM required by the ElasticSearch database can be distributed over multiple nodes. The database cluster as well as all other

[2]See https://openalex.org/

[3]See https://www.epo.org/en/searching-for-patents/data/bulk-data-sets/docdb

[4]Compared to a wide spectrum of alternative ANN and according to various distance measures, HNSW performs consistently well [25]; see also http://ann-benchmarks.com.

components are running on the GWDG OpenStack Cloud IT infrastructure[5] in Göttingen, Germany.

Computation & API The back-end extends the software stack for more functionality and to handle user interactions. It is written in the language *Go* and provides a plug-and-play Application Programming Interface (API) for end-users using *GraphQL*. This query language for APIs is a strongly typed interface that provides complete and understandable API documentation. Furthermore, it allows users to retrieve the data precisely that they have asked for.

The back-end also connects the document encoder with the client-facing run-time environment. This ensures that end-users can upload texts, which can then be encoded to numerical representations. Finally, the representation can be used to query similar documents from the database.

It can also be used to calculate distances between texts provided by the user using their numerical representation and distance metrics like the Cosine Similarity (Eq. (1)), the Manhattan (L1) Distance (Eq. (2)) or the Euclidean (L2) Distance (Eq. (3)).

$$\cos(\mathbf{a},\mathbf{b}) = \frac{\mathbf{ab}}{\|\mathbf{a}\|\|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} \mathbf{a}_i \mathbf{b}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{a}_i)^2}\sqrt{\sum_{i=1}^{n} (\mathbf{b}_i)^2}} \tag{1}$$

$$l1(\mathbf{a},\mathbf{b}) = \|\mathbf{a}-\mathbf{b}\| = \sum_{i=1}^{n} |\mathbf{a}_i - \mathbf{b}_i| \tag{2}$$

$$l2(\mathbf{a},\mathbf{b}) = \|\mathbf{b}-\mathbf{a}\|_2 = \sqrt{\sum_{i=1}^{n} (\mathbf{b}_i - \mathbf{a}_i)^2} \tag{3}$$

Furthermore, the back-end authenticates and authorizes the end-users. This is ensured by JSON Web Tokens (JWT), which are sent in the HTTP header of each request.

User Interfaces The website logic-mill.net features user registration, project presentation, and documentation in a Single Page web Application (SPA). It uses the *Vue JavaScript* framework[6] and consumes the *GraphQL* API provided by the backend.

4. Usage

The general idea behind the user interface is to enable a simple and easy plug-and-play interaction that simplifies the project setup. The machinery to use the document encoder model is already available, and the embeddings of large corpora are pre-computed and ready to be used. Users can do their projects in less time with fewer resources, since there is no need to download the raw data (time, storage), set up the machine learning pipeline (time, CPU/GPU, memory), encode the documents (time), and search through results (time, CPU/GPU, memory, internal storage).

User interfaces Upon registration on the website, users can access the system either through the web application on https://logic-mill.net/ (Fig. 2) or the Application Programming Interface (API).

The web application aims to help users familiarize themselves with the queries and data structure. It explains the different functionalities of the system and interactively shows the syntactically correct *GraphQL* queries. Thus, users can experiment, design, and adapt their queries.

The web app auto-generates these queries for various programming languages and tools. The release version of *Logic Mill* provides these examples in GraphQL for Curl, Python, R, and Go. However, any programming language with the ability to make HTTP requests and retrieve and process JSON responses can interact with the API endpoint.

API Functionality *Logic Mill* provides nine API endpoints for retrieval, pairwise similarity metric computation, and Nearest Neighbor search. Each functionality can be extended to multi-domain corpora (i.e., searching the most similar scientific publications for a patent), and they can involve available documents in the database as well as user-supplied documents or texts. The endpoints are summarized in Table 1.

Table 1
Overview of *Logic Mill*'s API functionality.

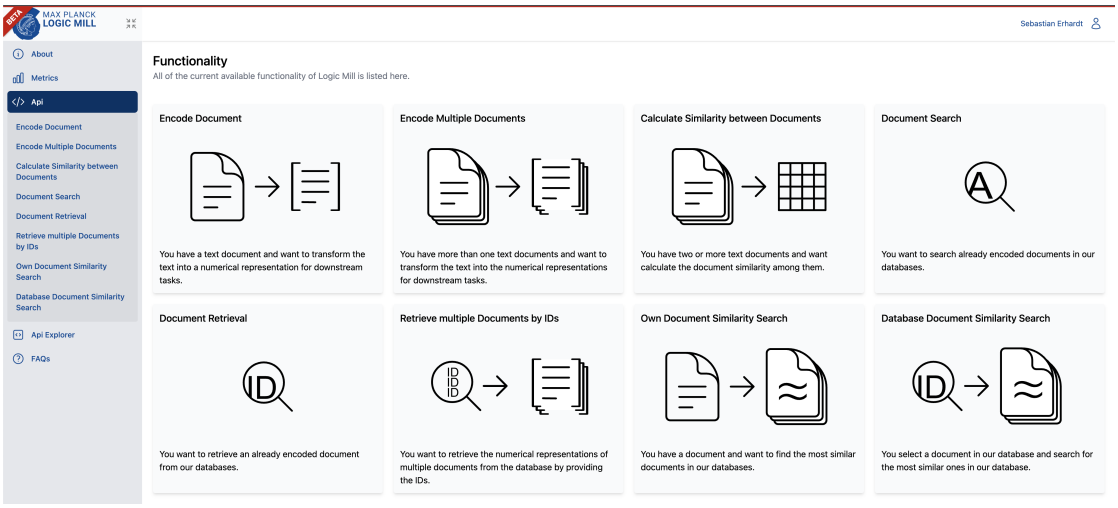
	Purely database	with own documents
Retrieval	Document, Documents searchDocuments	encodeDocument, encodeDocuments
Calculation	similarityCalculation	encodeDocumentAndSimilarityCalculation
NN Search	SimilaritySearch	embedDocumentAndSimilaritySearch

4.1. Retrieval

Document Retrieval Pre-computed embeddings for existing patent documents or scientific publications can be retrieved via the `Document` and `Documents` endpoints. Users provide the IDs and specify the database as well as the desired information such as title, IDs, or embedding, which are detailed in the online documentation. The returned information depends on the document type and data source, and copyright restrictions may apply.

[5] Gesellschaft für wissenschaftliche Datenverarbeiting mbH Göttingen. See https://www.gwdg.de/.
[6] See https://vuejs.org.

Figure 2: Screenshot of the Logic Mill Website - Overview



Document Search The `searchDocuments` endpoint allows keyword-based searches. *Logic Mill* retrieves documents matching the specified keywords and metadata from the available corpora. The returned information is consistent with that obtained through the `Document` and `Documents` endpoints.

Encode own documents The embeddings for users' curated documents can be generated and retrieved using the `encodeDocument` and `encodeDocuments` endpoints of our API. Users provide a title and an abstract, and the document encoder model returns the corresponding numerical representation.

4.2. Calculation

Users interested in pairwise similarities between documents have two options. They can retrieve embeddings for a set of documents one-by-one and compute similarity metrics themselves, or the rely on functionality provided by the system.

Calculate Document Similarities The `similarityCalculation` endpoint retrieves the similarity matrix for multiple documents in the database, enabling their comparison. Input consists of a list of source and target documents (identified by their identifiers and indices), along with the desired distance calculation metric (`cosine`, `l1`, `l2`).

Calculate Similarities with Own Documents To retrieve the similarities between a set of own curated documents and documents in the database, users use the `encodeDocumentAndSimilarityCalculation` endpoint. The user provides, for instance, title and abstract of the different documents, identifiers for later reference, and the type of distance calculation metric (`cosine`, `l1`, `l2`). The system encodes the documents with the help of the document encoder and uses provided metric to compute the distances among the encoded documents. A typical use case includes the representation as a similarity matrix.

4.3. Nearest Neighbor Search

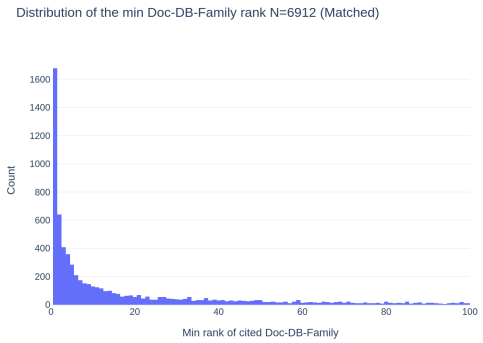
Database Document Similarity Search Given a known document, users can search for the approximate nearest neighbor within the same or other corpora. This is based on *cosine similarity* using the `SimilaritySearch` query. For example, users can request the five most similar scientific publications for a specific patent. The results include the title, similarity score, and ID of each document within its respective index.

Own Document Similarity Search Users can also provide their own documents and search for similar ones of *OpenAlex*, *DocDB*, that have already been encoded and stored in the database using the `embedDocumentAndSimilaritySearch` query.

5. Performance

To thoroughly evaluate the performance of our system, we conducted a series of experiments focusing on patent-to-patent and patent-to-publication citations. This evalu-

Figure 3: Distribution of the first ranked patent (DocDB family member) cited by the reference Patent



ation is analogous to the prior art search by patent offices. Prior art search is a crucial task in the patent examination process. It is designed to uncover existing patents, publications, and other relevant articles that might challenge the novelty of the patent application under prosecution. Our experiments simulate the prior art search process, testing the system's ability to identify relevant prior art accurately and ensuring its practical applicability and robustness. This approach not only validates the system's quality but also aligns it with industry-standard practices, thereby enhancing its reliability and utility for examiners and inventors who conduct comprehensive prior art searches to make informed decisions and protect their innovations.

5.1. Patents to Patents

The first task is to identify patent-to-patent citations for 10k randomly selected patents granted by the EPO whose family consists of at least 10 documents. The goal is to predict the patent citations to prior patents on a patent family level. We then search our DocDB index for 1,000 approximated nearest neighbors present at the time of the earliest filing date and retrieve the closest patent documents from the EPO, the USPTO, and the WIPO. The approximate nearest neighbor results are then aggregated on a patent family basis and re-ranked.

Of 10,000 test cases, we were able to identify at least one correct family citation within the first 1,000 nearest neighbors in 8,641 cases ($\approx$ 86%). In 6,912 cases, at least one family citation is among the 100 nearest neighbors ($\approx$ 69%). Indeed , most of the first family citations are within the first 20 results (cf. Figure 3).

Table 2 shows the Mean Reciprocal Rank (MRR) and the Mean Average Precision (MAP) for various cutoff values $k \in \{5, 10, 20, 50, 100\}$. For $k = 5$, the MRR is equal to 0.264 and the MAP equals 0.06.

Table 2
MRR and MAP for patent-to-patent citations

k	MRR	MAP
5	0.263874	0.062466
10	0.276368	0.065272
20	0.283207	0.070677
50	0.287384	0.078185
100	0.288638	0.082916

5.2. Patents to Publications

Our second task focuses on identifying relevant publications that a patent should cite. To establish a ground truth for these citations, we leverage the Reliance on Science dataset [1, 26], which links US patent IDs to their corresponding citations within the Open Alex database. Each patent-paper citation in this dataset is assigned a confidence score ranging from 1 to 10. For our analysis, we exclusively consider citations with the highest confidence score of 10. The search parameters employed for this task remain consistent with those used in the previous analysis.

There are instances where we are not able to obtain an embedding. A common reason is the absence of title and (English) abstract. Another reason is that IDs present in the Reliance on Science dataset have changed in more recent versions of OpenAlex.

Of initially 10,000 randomly selected samples, we found at least one cited publication among the top 100 results in 3,740 cases ($\approx$ 37%). As can be seen in Figure 4, the distribution is not as steep as in the patent-to-patent case. Table 3 presents the MRR and MAP for various $k \in \{5, 10, 20, 50, 100\}$.

To understand the omission of a significant share of citations, we manually examined anecdotal cases where matches were not found within our approximate nearest neighbor (ANN) results. Our analysis reveals that many of these citations were applicant-added rather than examiner-cited. These applicant-added citations often appear in the patent description, a context not considered during model training, nor encoded in our system. Additionally, a lack of semantic similarity between the cited publication and the focal patent's abstract makes associating the two challenging even for humans.

5.3. System Performance

The strength of the system is however not only the novel language model utilized, but the speed and ease-of-use. The system can identify the closest 100 to 10,000 documents based on an already encoded reference document within one second. Depending on the load and search parameters, results can be retrieved even faster, within 100 milliseconds. If encoding is required, the process

Figure 4: Distribution of the first ranked publication cited by the reference patent

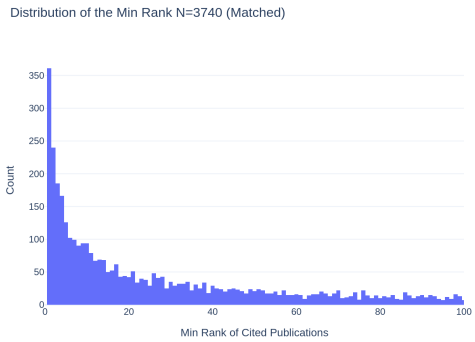


Table 3
MRR and MAP for patent-to-publication citations

k	MRR	MAP
5	0.162932	0.052162
10	0.179573	0.066367
20	0.190259	0.080288
50	0.197886	0.093623
100	0.200598	0.100609

takes one to two seconds.

6. Conclusion and Future Developments

Logic Mill is a novel modular software system that helps navigate knowledge embedded in scientific publications, patent documents, and other text corpora. The system is scalable and openly accessible. New documents are regularly ingested into the system.

Users can leverage the system, for example, in the following contexts:

- retrieve numerical representations of existing documents in Logic Mill's database
- generate numerical representations for their own documents
- calculate similarities between users' given documents, or documents in the database, or between users' given document and the one in the database
- search for similar documents present in the database given a query document that either exists in the database or users can provide the query document

Despite some present shortcomings, *Logic Mill* can assist many user groups. Researchers interested in studying the economics of innovation, science of science, and knowledge flows may be particularly interested in these capabilities. Its search capabilities may also be of interest to patent examiners and inventors seeking prior art related to their ongoing inventions. Researchers in many fields may want to use *Logic Mill* as a literature and citation recommender system.

As *Logic Mill* is actually agnostic about the underlying document encoder model, the system may improve with the advent of better models. For instance, the next generations of deep learning language models may provide larger context windows so that *Logic Mill* can be able to encode a larger portion of documents. This might extend to models that are specifically trained to identify similarities between patents and their cited publications.

The impact of the approximate nearest neighbor (ANN) search on our patent data also needs to be investigated[7].

6.0.1. Acknowledgements

For helpful comments, we thank Matt Marx and the participants at the 2022 Munich Summer Institute and the 2022 Summer School on Data and Algorithms for Science, Technology & Innovation Studies at KU Leuven. We also thank the 2023 I3 NBER workshop participants for their comments. We also thank the team at the computing and IT competence center of GWDG for their continuous support. Dietmar Harhoff acknowledges support from Deutsche Forschungsgemeinschaft (CRC 190).

[7] A up-to-date benchmarking with industry datasets can be found here: https://ann-benchmarks.com/

References

[1] M. Marx, A. Fuegi, Reliance on science: Worldwide front-page patent citations to scientific articles, Strategic Management Journal 41 (2020) 1572–1594. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smj.3145. doi:https://doi.org/10.1002/smj.3145.

[2] F. Poege, D. Harhoff, F. Gaessler, S. Baruffaldi, Science quality and the value of inventions, Science Advances 5 (2019) eaay7323. URL: https://www.science.org/doi/10.1126/sciadv.aay7323. doi:10.1126/sciadv.aay7323.

[3] A. B. Jaffe, G. de Rassenfosse, Patent citation data in social science research: Overview and best practices, Journal of the Association for Information Science and Technology 68 (2017) 1360–1374. URL: https://onlinelibrary.wiley.com/doi/10.1002/asi.23731. doi:10.1002/asi.23731.

[4] A. Rubin, E. Rubin, Systematic Bias in the Progress of Research, Journal of Political Economy 129 (2021) 2666 – 2719. URL: https://doi.org/10.1086/715021. doi:10.1086/715021.

[5] S. L. Woltmann, L. Alkærsig, Tracing university–industry knowledge transfer through a text mining approach, Scientometrics 117 (2018) 449–472. URL: http://link.springer.com/10.1007/s11192-018-2849-9. doi:10.1007/s11192-018-2849-9.

[6] B. Kelly, D. Papanikolaou, A. Seru, M. Taddy, Measuring Technological Innovation over the Long Run, American Economic Review: Insights 3 (2021) 303–320. URL: https://pubs.aeaweb.org/doi/10.1257/aeri.20190499. doi:10.1257/aeri.20190499.

[7] J. Beel, B. Gipp, S. Langer, C. Breitinger, Research-paper recommender systems: a literature survey, International Journal on Digital Libraries 17 (2016) 305–338. URL: https://doi.org/10.1007/s00799-015-0156-0. doi:10.1007/s00799-015-0156-0.

[8] K. Sparck Jones, A Statistical Interpretation of Term Specificity and its Application in Retrieval, Journal of Documentation 28 (1972) 11–21. URL: https://doi.org/10.1108/eb026526. doi:10.1108/eb026526, publisher: MCB UP Ltd.

[9] M. Ghosh, S. Erhardt, M. E. Rose, E. Buunk, D. Harhoff, PaECTER: Patent-level Representation Learning using Citation-informed Transformers, 2024. URL: https://arxiv.org/abs/2402.19411. doi:10.48550/ARXIV.2402.19411, version Number: 1.

[10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, in: Proceedings of the 2019 Conference of the North, Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186. URL: http://aclweb.org/anthology/N19-1423. doi:10.18653/v1/N19-1423.

[11] J. Priem, H. Piwowar, R. Orr, OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts, 2022. URL: https://arxiv.org/abs/2205.01833. doi:10.48550/ARXIV.2205.01833, version Number: 2.

[12] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space, 2013. URL: http://arxiv.org/abs/1301.3781. doi:10.48550/arXiv.1301.3781, arXiv:1301.3781 [cs].

[13] J. Pennington, R. Socher, C. Manning, GloVe: Global Vectors for Word Representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1532–1543. URL: https://aclanthology.org/D14-1162. doi:10.3115/v1/D14-1162.

[14] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching Word Vectors with Subword Information, 2017. URL: http://arxiv.org/abs/1607.04606. doi:10.48550/arXiv.1607.04606, arXiv:1607.04606 [cs].

[15] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (1997) 1735–1780. URL: https://direct.mit.edu/neco/article/9/8/1735-1780/6109. doi:10.1162/neco.1997.9.8.1735.

[16] O. Melamud, J. Goldberger, I. Dagan, context2vec: Learning Generic Context Embedding with Bidirectional LSTM, in: Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, Association for Computational Linguistics, Berlin, Germany, 2016, pp. 51–61. URL: https://aclanthology.org/K16-1006. doi:10.18653/v1/K16-1006.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention Is All You Need, in: NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017. URL: https://dl.acm.org/doi/10.5555/3295222.3295349. doi:10.5555/3295222.3295349.

[18] I. Beltagy, K. Lo, A. Cohan, SciBERT: A Pretrained Language Model for Scientific Text, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, Hong Kong, China, 2019, pp. 3613–3618. URL: https://www.aclweb.org/anthology/D19-1371. doi:10.18653/v1/D19-1371.

[19] G. Aslanyan, I. Wetherbee, Patents Phrase to Phrase Semantic Matching Dataset, 2022. URL: http://arxiv.org/abs/2208.01171, number: arXiv:2208.01171

arXiv:2208.01171 [cs].

[20] S. Althammer, M. Buckley, S. Hofstätter, A. Hanbury, Linguistically Informed Masking for Representation Learning in the Patent Domain, 2021. URL: http://arxiv.org/abs/2106.05768, number: arXiv:2106.05768 arXiv:2106.05768 [cs].

[21] A. Cohan, S. Feldman, I. Beltagy, D. Downey, D. Weld, SPECTER: Document-level Representation Learning using Citation-informed Transformers, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 2270–2282. URL: https://www.aclweb.org/anthology/2020.acl-main.207. doi:10.18653/v1/2020.acl-main.207.

[22] J.-S. Lee, J. Hsiang, Patent classification by fine-tuning BERT language model, World Patent Information 61 (2020) 101965. URL: https://linkinghub.elsevier.com/retrieve/pii/S0172219019300742. doi:10.1016/j.wpi.2020.101965.

[23] S. Fricke, Semantic Scholar, Journal of the Medical Library Association 106 (2018). URL: http://jmla.pitt.edu/ojs/jmla/article/view/280. doi:10.5195/jmla.2018.280.

[24] Y. A. Malkov, D. A. Yashunin, Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs, 2018. URL: http://arxiv.org/abs/1603.09320. doi:10.48550/arXiv.1603.09320, arXiv:1603.09320 [cs].

[25] M. Aumüller, E. Bernhardsson, A. Faithfull, ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms, Information Systems 87 (2020) 101374. URL: https://linkinghub.elsevier.com/retrieve/pii/S0306437918303685. doi:10.1016/j.is.2019.02.006.

[26] M. Marx, A. Fuegi, Reliance on science by inventors: Hybrid extraction of in-text patent-to-article citations, Journal of Economics & Management Strategy 31 (2022) 369–392. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jems.12455. doi:https://doi.org/10.1111/jems.12455.

A. Appendix

Logic Mill provides an API endpoint that uses *GraphQL*. The *GraphQL* query determines what should be executed and what information should be returned. Users need an API key to access the API.

Logic Mill web app provides a user-interface where dynamic *GraphQL* queries are generated for *cURL, Python, R* and *Go*. [8]

[8] Stata is another commonly used statistics tool, but it cannot retrieve these JSON responses. Recent versions of Stata can, however, embed Python code.

Once a query is constructed in *GraphQL*, it can be implemented and executed using any modern programming language.

Basic structure of a request A basic request written in *Python* is displayed in source code . [9] The variable called `query` defines the *GraphQL* query. User-specific `variables` and requested information. By providing the index and the ID (in this case an EPO patent with the number of `EP19164094B1`), the title of the document and the vector (the numerical representation) of the encoded text are requested. The return variables can be customized (lines 8-11) to include other information about the document. The object called `response` (line 19) is a dictionary that can be processed further. As one can see, the *Python code* directly includes the *GraphQL* query.

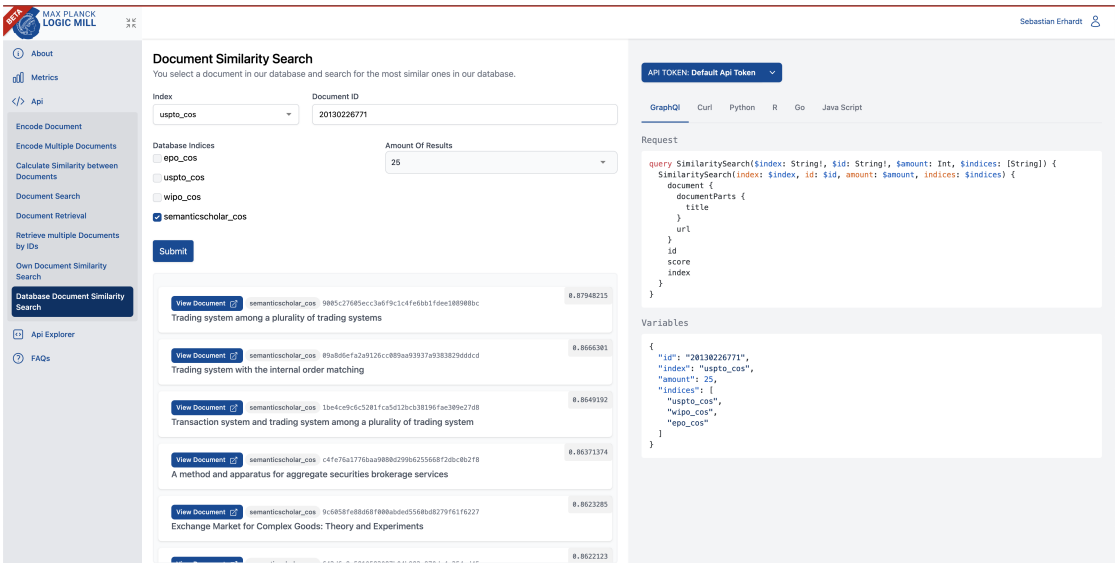
Example code showing the structure of any interaction with the web API in Python

```
import requests
import json

TOKEN = 'XXX'
ENDPOINT = 'https://lm/api/endpoint/url/here'
headers = {
  'content-type': 'application/json',
  'Authorization': 'Bearer ' + TOKEN,
}
query="""{
  Document(index: "docdb_cos", id: "EP19164094B1") {
    documentParts {
      title
    }
    vector
  }
}"""

r = requests.post(ENDPOINT,
    headers=headers,
    json={'query': query})

if r.status_code == 200:
    response = r.json()
    print(response)
else:
    print(f"Error executing\n{query}\non {url}")
```

Parameters While the previous example is the most straightforward and basic implementation, it is not always the most suitable in practice. In many cases, one has to make multiple requests to retrieve all the data. For example, it is possible to request the encoding for multiple documents in one request. However, retrieving more than 10,000 documents is impossible. To do this, the code needs to include a loop that executes a query with different parameters with each iteration. We will call this a parameterized query. The user provides two

[9] Our examples are written in Python using the `requests` package for http requests.

Figure 5: Logic Mill Website - Example Query



parameters to interact with the web API. These are the *GraphQL* query and the query parameters. Doing the loop with parameters will make the code more readable. The code example A shows the example with the query and the variables object. The code for looping is omitted, as is the base code for handling the request.

Example code showing with a query with parameters

```
# (same setup as above)
# Build GraphQL query
query="""
query Documents($index: String!, $keyword: String!) {
  Documents(index: $index, keyword: $keyword) {
    id
    documentParts {
      title
    }
    vector
  }
}
"""

# Build variables
variables = [
    {"keyword": "EP19164094B1", "index": "docdb_cos"},
    {"keyword": "20130226771", "index": "docdb_cos"}
]

# Send request
r = requests.post(ENDPOINT,
    headers=headers,
    json={
        'query': query,
        'variables': variables
    })

# Handle request
# (...)
```