GYANI: An Indexing Infrastructure for Knowledge-Centric Tasks

Dhruv Gupta
Max Planck Institute for Informatics
Saarbrücken Graduate School of Computer Science
Saarland Informatics Campus, Germany
dhgupta@mpi-inf.mpg.de

Klaus Berberich
Max Planck Institute for Informatics
Saarland Informatics Campus, Germany
htw saar, Saarbrücken, Germany
kberberi@mpi-inf.mpg.de

ABSTRACT

In this work, we describe GYANI (*gyan* stands for *knowledge* in Hindi), an indexing infrastructure for search and analysis of large semantically annotated document collections. To facilitate the search for sentences or text regions for many knowledge-centric tasks such as information extraction, question answering, and relationship extraction, it is required that one can query large annotated document collections interactively. However, currently such an indexing infrastructure that scales to millions of documents and provides fast query execution times does not exist. To alleviate this problem, we describe how we can effectively index layers of annotations (e.g., part-of-speech, named entities, temporal expressions, and numerical values) that can be attached to sequences of words. Furthermore, we describe a query language that provides the ability to express regular expressions between word sequences and semantic annotations to ease search for sentences and text regions for enabling knowledge acquisition at scale. We build our infrastructure on a state-of-the-art distributed extensible record store. We extensively evaluate GYANI over two large news archives and the entire Wikipedia amounting to more than fifteen million documents. We observe that using GYANI we can achieve significant speed ups of more than 95× in information extraction, 53× on extracting answer candidates for questions, and 12× on relationship extraction task.

ACM Reference Format:
Dhruv Gupta and Klaus Berberich. 2018. GYANI: An Indexing Infrastructure for Knowledge-Centric Tasks. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271745

1 INTRODUCTION

Knowledge-centric tasks such as information extraction rely on meaningful text regions extracted from the analysis of large semantically annotated document collections. A semantically meaningful text region connects, for instance, a named entity to a literal via a paraphrase of a knowledge graph predicate (e.g., alan turing published papers during the 30s) within a context window of few sentences. Currently there exists no indexing infrastructure that allows for interactive querying of such text regions on large annotated document collections. To enable such complex knowledge-centric tasks at scale, we are faced with three key challenges. The first challenge is to provide a more expressive query language to enable knowledge acquisition at scale. The second challenge is to identify a data model that respects the salient relationships between sequences of words and semantic annotations. The third challenge is to identify key indexes as building blocks using which we can then assemble the required search results quickly. By building such a core infrastructure and an expressive language for querying such text regions at scale we can provide more training data for improving the effectiveness of complex machine learning algorithms that are developed to serve these knowledge-centric tasks.

A flexible and more expressive query language for us is one that goes beyond simple Boolean operators. What is thus needed is a counterpart to GREP for analysis of large semantically annotated document collections. GREP is a powerful Unix utility that allows for regular expression based search over text documents. It is an indispensable tool when trying to find and manipulate lines of text matching a particular regular expression. However, when it comes to searching millions of annotated documents, a counterpart to GREP is missing. This is challenging, as natural language text, unlike field delimited files (e.g., TSV) is not structured. However, with the help of modern natural language processing tools we can impose a lexico-syntactic structure over text. Such tools now allow us to annotate large document collections with various kinds of semantic annotations, such as part-of-speech (e.g., NN), temporal expressions, (e.g., last year), and named entities (e.g., Alan Turing). The annotations give deeper semantics to the terms as well as provide canonical linguistic structure to text. This lexico-syntactic structure thus offers us an opportunity to implement a counterpart to GREP over annotated document collections.

Queries composed of regular expressions over word sequences and annotations offer us an opportunity for knowledge acquisition at scale. With such a query language we can simplify many knowledge-centric tasks. For instance, a template to identify n-ary relations about disasters can be expressed as: ⟨(NUMBER) were killed in (LOCATION) on (DATE)⟩. Below we discuss existing solutions for knowledge-centric tasks and outline how these approaches can benefit from our infrastructure.

Information Extraction requires one to acquire facts that hold between named entities from unstructured text [17, 25]. For this task, extraction templates are employed, e.g., Hearst patterns [17]. A template for identifying scientists in a document collection, can be written as: ⟨scientists such as NNP_1 … NNP_n⟩. To execute this example, documents that contain the terms ⟨scientists such as⟩ are retrieved and the sentences are annotated for part-of-speech tags and subsequently filtered to produce relations. With GYANI this can be achieved interactively by issuing the following query: ⟨scientists such as (PERSON)∗⟩.

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271745

Question Answering. Knowledge Graphs encode relationships or facts in the form of ⟨SUBJECT, PREDICATE, OBJECT⟩ triples. Natural language questions such as: ⟨which countries joined nato and when?⟩, need to be transformed into a structured query using the SPARQL language [14, 28]. However, knowledge graphs are rarely complete (e.g., with respect to temporal information) and thus we are forced to spot the answers for such questions in large annotated document collections. By providing the facility to formulate a question as a template that expresses the relationship between a named entity and temporal expression we can provide a large extracted set of sentences as an input to machine learning algorithms that can further reason about the correct time interval for this question [16]. For example, with GYANI we can extract training data with following query: ⟨(LOCATION) joined nato (DATE)⟩.

Fact Spotting. The inverse task of information extraction is to find textual evidence in support of the facts in a knowledge graph [23]. To deal with the linguistic variations in which many of the canonical relations are phrased, paraphrase dictionaries are employed [25]. To account for the many different ways a named entity can be mentioned, its surface forms are added to increase the recall of the pattern-matching procedure. The procedure to perform this task can be greatly simplified by combining regular expressions, annotations, and word sequences. This has important consequences when we are trying to spot out-of-knowledge-graph entities, for which we may have to manually specify many surface forms for a yet to be canonicalized named entity. For example, consider the query: ⟨[united states | us | usa] to criticize [russia | russland]⟩.

Semantic Search. Semantic annotations are an important building block for many linguistic and information retrieval tasks as they lend themselves for conveying deeper semantics to the terms in text. Thereby, allowing the user to convey her information need in a more structured manner [7, 14]. However, current inverted indexes offer only limited capabilities to search and analyze semantically annotated text. Using prior art many documents that might qualify to satisfy the user's information need are lost due to this semantic gap. For instance, a user issuing the query ⟨nineties decade⟩, will miss documents that also contain the time interval 1990 – 1999 or the query ⟨paris hilton⟩ may retrieve documents matching the location and hotel when the intent was related to the celebrity. We propose to bridge the semantic gap by providing operators to attach meanings to terms. For example, to retrieve sentences in document collections detailing relationships between the person, paris hilton and the time period, nineties decade: ⟨(paris hilton)⊕(PERSON) (WORD)∗ (nineties decade)⊕([1990,1999])⟩.

Implementing a GREP-like interface over large annotated document collections poses many challenges. At the data modeling level, we need to identify a data model in which we can model a text document along with a multitude of annotations while preserving the sequential order of words. At the index implementation level, we need to work through all possible choices for indexing units in the design space such that the index structures provide fast query execution times (i.e., order of millisecond query execution times for complex and lengthy queries). To address these challenges, we describe GYANI, an infrastructure that indexes semantically annotated text using a novel data model and provides a highly expressive language for queries involving regular expressions over word sequences and annotations. Furthermore, we show how we model the possible combinations of word sequences and annotations in a multi-layered data model to process complex and verbose GREP-like queries for knowledge-centric tasks quickly and at scale.

Contributions and Outline. As a first contribution, we describe a novel data model to represent text with various layers of semantic annotations (Section 2.1). Second, we propose a novel query language involving regular expressions between words and annotations (Section 2.3). Third, we present an efficient implementation of our data model that provides fast query execution times (Section 2.4). Finally, we construct a comprehensive testbed of knowledge-centric tasks to show the efficiency of GYANI at the tasks of information extraction, relation extraction, question answering, fact spotting, and semantic search (Section 3).

2 GYANI

Gyan in Hindi means *knowledge* and *gyani* refers to a person who possesses knowledge. Our proposed infrastructure is named GYANI to personify an index over text that contains knowledge by virtue of semantic annotations adorned on text. We next describe the data model that consists of layers of annotations that can be attached to sequences of words, thereby allowing different semantic interpretations of natural language. We then describe the query language to perform regular expression based search in semantically annotated text. Lastly, we discuss the design space of the data structures used for indexing annotated documents in GYANI.

2.1 Data Model

Consider a document collection, $\mathrm{D} = \{\mathrm{d}_1, \mathrm{d}_2, \ldots, \mathrm{d}_N\}$. Each document in the collection $\mathrm{d} \in \mathrm{D}$ consists of layers of sequences containing words or semantic annotations. Each layer of sequences consists of elements drawn from a particular vocabulary with their positional span. Concretely, each element ℓ in a layer $\mathcal{L}$ is a relation defined over the Cartesian space described by its layer alphabet $\Sigma_{\mathcal{L}}$ and an interval of natural numbers $\mathbb{N}$ indicating their positions in the sequence. Formally, the definition of an element is,

$$\ell \subset \mathbb{N} \times \mathbb{N} \times \Sigma_{\mathcal{L}}. \tag{1}$$

The word layer in each document consists of words drawn from vocabulary $\Sigma_{\mathcal{V}}$ with unit length positional spans. Formally,

$$\mathrm{d}_{\mathcal{V}} = \langle w_{[1,1]}, \ldots, w_{[|\mathrm{d}|,|\mathrm{d}|]} \rangle, \tag{2}$$

where $|\mathrm{d}|$ denotes the number of words in the document and the interval in subscript records the unit length spans as *positions* of the word in the sequence.

Natural language word sequences present in the documents can be preprocessed with various kinds of semantic annotators, e.g., part-of-speech, temporal expressions, and named entities. Each type of annotation thus signifies an interpretation (semantics) of the terms by inspecting the signals derived from its surrounding context. For instance, in the sentence, ⟨peace was brokered between them during 1903⟩, the term 1903 will be tagged as a cardinal number by part-of-speech tagger and as a date by a temporal tagger. We treat each sequence of annotation derived over a sequence of words as an annotation layer. An annotation layer $\mathrm{d}_{\mathcal{L}}$ over a document is denoted by,

$$\mathrm{d}_{\mathcal{L}} = \langle \ell_{[\mathrm{i},\mathrm{j}]}, \ldots, \ell_{[\mathrm{k},\mathrm{l}]} \rangle, \tag{3}$$

where, each annotation ℓ *spans* the positions described by the interval in subscript ($[i, j]$ with $i \leqslant j$). In short-hand notation, we refer to a sequence of words as, $w_{\langle i:j \rangle} = \langle w_{[i,i]}, \ldots, w_{[j,j]} \rangle$, and a sequence of annotations as, $\ell_{\langle i:l \rangle} = \langle \ell_{[i,j]}, \ldots, \ell_{[k,l]} \rangle$, where $i \leqslant j$, $k \leqslant l$, and $i < k$. A text region consists of word sequences $w_{\langle i:j \rangle}$ in a document $d \in D$ that satisfies conditions imposed by the query involving word sequences and the semantic annotations. In other words, a document is considered a match if it contains at least one text region that meets the constraints specified in the query involving word sequences and semantic annotations.

Each document is further accompanied by metadata such as its unique identifier (id) and its timestamp (ts), $d_{\mathcal{M}} = \langle id, ts \rangle$. The pairs of all valid $\langle id, ts \rangle$ pairs in the document collection (D) are denoted by $\Sigma_{\mathcal{M}}$.

2.2 Semantic Annotations

Natural language processing (NLP) tools allow us to markup various kinds of annotations in text. We specifically focus on four fundamental types of semantic annotations that are commonly provided by off-the-shelf NLP toolkits: part-of-speech, temporal expressions, numerical values, and named entities. We describe these four different types of semantic annotations in the following paragraphs and justify their importance.

Part-of-Speech (PoS) Tags are assigned to a word based on common linguistic characteristics derived from their surrounding terms [18]. Thus, a word's PoS tag can quickly help to describe the context in which it occurs [18]. Examples of PoS tags are: nouns (NN), verbs (VB), and quantities (CD). The PoS annotation layer is denoted by $d_{\mathcal{P}}$. PoS tags are significant as they form the basis for tasks in computational linguistic. For instance, PoS annotations such as nouns can be used to generate text summaries or they can be employed for complex natural language processing tasks such as named entity recognition and named entity disambiguation [18].

Temporal Expressions convey mentions of time in text. Natural language descriptions of time are often convoluted, as time can be explicit as a concrete date (e.g., July 4, 1776), it can be implicit emphasizing that it is commonsense knowledge and refers to already known facts (e.g., independence day of the usa), or it can be relative with respect to dates mentioned in the narrative of the text (e.g., yesterday). Temporal annotators are able to detect these implicit, explicit, and relative temporal expressions. Moreover, the detected expressions can be resolved to definite time intervals by using metadata such as publication dates. Formally, each temporal expression that is annotated is represented as an interval ($t = [b, e]$) with a begin and end time point. We denote the temporal annotation layer by $d_{\mathcal{T}}$. Temporal expressions are significant in computational linguistics as they signify the presence of events.

Numerical Values are resolved values of words tagged as cardinal numbers (CD) that are not temporal expressions. Examples of such annotations are percentage values, monetary values, and other numerical figures. Just like temporal expressions, these numerical values can be explicit (e.g., 95%) or implicit (e.g., a dozen). Numerical values are useful in quantifying tragic events e.g., identifying casualties of a natural disaster or for quantifying financial events e.g., profit and loss for a company. The layer containing these annotations is denoted by $d_{\mathcal{N}}$.

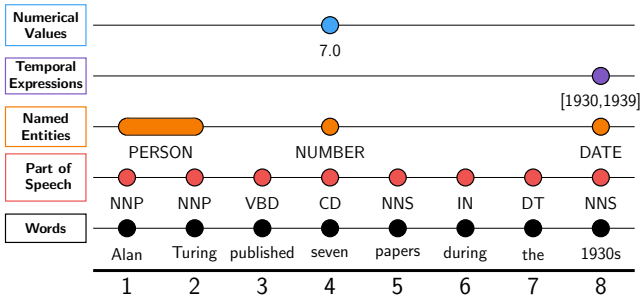

Figure 1: Data model showing the text & annotation layers.

Named Entities are mentions of persons, organization, locations etc. in text. These annotations are classified based on context surrounding their mentions. For instance, in the sentence, ⟨alan turing was a scientist who lived in britain⟩, the mentions alan turing and britain are classified as person and location, respectively. We denote the named entity annotation layer by $d_{\mathcal{E}}$. Named entities can be further disambiguated to their canonical entries (URI) in knowledge graphs and accommodated in our data model as a layer of URIs. Entity annotations allow for advanced text analytics, e.g., the ability to perform entity summarization by retrieving all sentences in documents that contain mentions of that entity.

These four kinds of annotations offer an immense opportunity for various text analytics applications that can be developed for linguists, scholars in humanities, and journalists. Figure 1 illustrates how the different annotation layers for part-of-speech tags, named entities, temporal expressions, and numerical values are overlayed over a sequence of words according to our data model.

2.3 Query Language

We now describe the language to express queries involving regular expressions over word sequences and annotations. The complete grammar containing production rules for query generation is given in Figure 2. In the following paragraphs we discuss the semantics of the operators and how to form queries using this grammar.

To formalize the semantics of the operators, we use the function GYANI(Q) to represent the mapping between query Q and a set of documents $\{d_1, d_2, \ldots, d_k\}$ that satisfy the conditions of the query. A document is deemed a match if contains a text region matching the query. A text region is considered a match to a query if and only if the sequence of elements $\ell_{\langle i:l \rangle}$ in the query occurs as a contiguous sequence in the appropriate layer of the document $d_{\mathcal{L}}$. Formally,

$$\ell_{\langle i:l \rangle} = \langle \ell_{[i,j]} \ldots \ell_{[k,l]} \rangle \sqsubset d_{\mathcal{L}}. \tag{4}$$

For instance, ⟨alan turing⟩ ⊏ ⟨computer scientist alan turing⟩, while ⟨scientist turing⟩ ⊏̸ ⟨computer scientist alan turing⟩.

Boolean Operators in our query language are: AND ($\boxed{\wedge}$), OR ($\boxed{\vee}$), and NEGATION ($\boxed{\neg}$). The binary operators AND and OR act on two sequences of words such that the resulting documents must contain both or either of the sequences of words respectively. The NEGATION operator, on the other hand, is a unary operator and acts on a sequence of words such that the resulting documents do not contain that sequence of words. For instance, the query: [⟨alan turing⟩ $\boxed{\vee}$ ⟨enigma machine⟩] $\boxed{\wedge}$ ⟨wins war⟩ selects those documents that contain either of the sequences of words ⟨alan turing⟩ or ⟨enigma machine⟩ with the phrase ⟨wins war⟩.

Stack Operator. With the help of annotations, different interpretations can be stacked on top of sequences of words in the text layer. This is done with the help of the STACK ($\oplus$) operator. Thus, a particular interpretation can be attached to a sequence of words, for example: $\langle$last year$\rangle\oplus$[1918,1918], refers to those temporal expressions that resolve to the year 1918. The STACK operator is a unary operator, such that it results in only those documents that contain the sequences of words with that particular annotation attached to them. Formally, the semantics can be specified as,

$$\text{GYANI}\left(w_{\langle i:j\rangle} \oplus \ell_{[i,j]}\right) = \left\{ d \in D \,\middle|\, w_{\langle i:j\rangle} \sqsubset d_{\mathcal{V}} \wedge \ell_{[i,j]} \sqsubset d_{\mathcal{L}} \right\}.$$

For example, the query: $\langle$paris hilton$\rangle\oplus$(PERSON) retrieves those documents that contain the sequence of words paris hilton annotated as a person by the named entity annotator.

Regular Expression Operators. We consider four basic regular expressions in our query language at word or annotation level. These are (based on [6]), STAR ($*$) that allows greater than zero repetition; PLUS ($+$) that allows greater than one repetition; QUES ($?$) that either matches a single word or annotation or nothing; and DOT ($.$) that acts as a placeholder for any word or annotation. We also provide the UNION ($|$) operator to group together results for two different word sequences. The semantics of UNION operator is equivalent to that of the OR operator. We are particularly interested in combinations of those regular expressions that join two word or annotation sequences to match and retrieve text regions. The regular expression operators that we provide are: DOT STAR ($\boxed{.*}$), DOT PLUS ($\boxed{.+}$), DOT QUES ($\boxed{.?}$), and DOT ($\boxed{.}$) operator. We show the semantics of the DOT PLUS operator below:

$$\text{GYANI}\left(w_{\langle i:j\rangle} \boxed{.+} w_{\langle k:l\rangle}\right) = \left\{ d \in D \,\middle|\, \begin{array}{r} (w_{\langle i:j\rangle} \sqsubset d_{\mathcal{V}}) \wedge \\ (w_{\langle k:l\rangle} \sqsubset d_{\mathcal{V}}) \wedge \\ (k - j \geqslant 2) \end{array} \right\}. \quad (5)$$

Semantics for the other regular expression operators can be obtained similarly by adjusting the gaps between the two word or annotation sequences that are being joined (i.e., varying the distance $(k - j)$ in Equation 5). Concretely, for the DOT STAR $\boxed{.*}$ operator we have $(k - j \geqslant 1)$; for the DOT QUES $\boxed{.?}$ operator $(k - j \in [1, 2])$; and for the DOT $\boxed{.}$ operator $(k - j = 2)$. The operators $\boxed{.*}$ and $\boxed{.+}$ can be greedy in matching multiple sentences in a document that satisfy the positional constraints. In order, to keep only the shortest possible match, these can be turned lazy by using QUES as a flag. That is, the operators $\boxed{.*?}$ and $\boxed{.+?}$ match only the shortest positional difference.

Projection Operators. The regular expression operators, $\boxed{.*}$ and $\boxed{.+}$ will yield documents that contain text regions spanning multiple sentences. In many knowledge-centric applications it is desirable that the text regions lie within a context window of few sentences or be restricted to a single sentence. To support this operation, we propose the k—PROJECT operator $\boxed{\pi_k}$ where k specifies the number of sentences the positional intervals may span.

We further specifically instantiate operators that project the regular expression match for words or annotations within a sentence boundary. These operators consist of PHRASE STAR ($\boxed{\ell*}$), PHRASE PLUS ($\boxed{\ell+}$), PHRASE QUES ($\boxed{\ell?}$), and PHRASE DOT ($\boxed{\ell}$) operators. The PHRASE PROJECTION operators are obtained by binding the regular expression to a specific annotation type. The overriding constraint

OPERATOR $\rightarrow$	$\wedge \mid \vee \mid \neg \mid \oplus$
REG. EXPR. $\rightarrow$	$* \mid + \mid ? \mid . \mid \mid$
QUERY $\rightarrow$	$\mid \langle$SEQUENCE$\rangle$ OPERATOR $\langle$SEQUENCE$\rangle$ QUERY*
	$\mid \langle$SEQUENCE$\rangle$ REG. EXPR. $\langle$SEQUENCE$\rangle$ QUERY*
	$\mid \langle$SEQUENCE$\rangle$ REG. EXPR. QUERY*
	$\mid$ REG. EXPR. $\langle$SEQUENCE$\rangle$ QUERY*
	$\mid$ [QUERY] $\mid \langle$ SEQUENCE $\rangle \mid \varepsilon$
SEQUENCE $\rightarrow$	$\Sigma^+_{\mathcal{V}} \mid \Sigma^+_{\mathcal{P}} \mid \Sigma^+_{\mathcal{E}} \mid \Sigma^+_{\mathcal{T}} \mid \Sigma^+_{\mathcal{N}}$

Figure 2: The query language.

we impose on these operators is that the resulting positions lie within sentence boundaries in addition to the positional constraints of the regular expression. The semantics of one such instance (other instances can be derived in a similar manner) is stated below:

$$\text{GYANI}\left(w_{\langle i:j\rangle} \boxed{\ell*}\right) = \left\{ d \in D \,\middle|\, \begin{array}{r} (w_{\langle i:j\rangle} \sqsubset d_{\mathcal{V}}) \wedge (l_{[p,q]} \sqsubset d_{\mathcal{L}}) \wedge \\ (j < p) \wedge ([i, q] \subseteq [m, n]) \end{array} \right\}.$$

where, the interval $[m, n]$ encompasses the sentence boundary containing the word sequence and the annotation in the document.

We can further combine regular expressions and the projection operators to establish the following equivalence:

$$w_l \boxed{\ell+} w_r \equiv \boxed{\pi_1}\left(w_l \boxed{.*?} \ell \boxed{.*?} w_r\right). \quad (6)$$

2.4 Index Design

Next, we discuss the design and implementation aspects concerning the data model. While considering the implementation for GYANI we considered three key aspects: scalability, reliability, and compatibility. Prior work [26, 30] highlights the utility of using combinations of inverted indexes and augmented indexes (e.g., next word, phrase, or direct indexes) can provide in answering phrase queries. In particular, we base our index design on a combination of inverted and direct indexes. Since, these indexes are present in existing infrastructure [26], a complete overhaul of the indexes is not required thereby assuring compatibility of our implementation. Additionally, these indexes can be sharded (distributed) across a network of commodity hardware, making them extremely scalable and reliable.

Design Space. We now describe the design space to decide what are the appropriate indexing units for our data model. The basic queries in our language consist of word sequences, $w_{\langle i:j\rangle}$. The word sequences can be stacked with annotations, $w_{\langle i:j\rangle} \oplus \ell_{[i,j]}$, to specify semantics. Moreover, for queries involving regular expressions, we need to maintain positional constraints. Figure 3 summarizes the key choices. We now contemplate four different choices for designing our indexes to support the query language.

NAÏVE DESIGN. A naïve design choice is to implement GREP as-is. That is, given a query traverse the entire document collection to retrieve the required documents. This can also be achieved by a direct index, that stores the layers of annotations and word sequence against the document metadata.

INVERTED INDEX DESIGN. The naïve design can be improved by using an inverted index over elements to narrow down the search space. The inverted index structure will thus store individual elements with singleton positional offsets, to indicate unit intervals.

As shown in Figure 3, this design choice corresponds to indexing units being unigrams $w_{[i,i]}$ or annotations $\ell_{[i,i]}$ spanning unit intervals. The types of queries that can be answered using an inverted index built on these units are, Boolean and wildcard combinations of unigrams or single annotations. However, there are three shortcomings of adopting this design choice. First, this approach is *lossy*, as the positional information conveyed by intervals is lost. That is, a named entity with positional interval $[i, j]$ is not equivalent to $\{i\}, \{i+1\}, \ldots, \{j\}$. For instance, the annotation $(\text{PERSON})_{[1,2]}$ conveys that the words at positions $w_{[1,1]}$ and $w_{[2,2]}$ is one person as opposed to $(\text{PERSON})_{[1,1]}$ & $(\text{PERSON})_{[2,2]}$ indicative of two different persons. Second, retrieving word sequences incurs a high computational cost as the number of calls to the index is proportional to the size of the sequence. Third, word sequences stacked with annotations can not be answered in this design space. These issues can be mitigated to some degree by combining the inverted index with the direct index. This combined design mimics the choice made by Cafarella and Etzioni [9] for their neighbor index. The neighbor index modeled an inverted index over unigrams and then wraps the annotation layer for the document containing the unigram as an additional payload to the posting list.

K-FRAGMENT DESIGN. Going beyond single elements as indexing units, we can decide on their size with respect to our annotation choices. Since, the annotations for PoS tags, named entities, temporal expressions, and numerical values incrementally build on each other, they share positional information based on the word sequences which they annotate. For example, in Figure 1, the word sequence alan turing is annotated with PoS tags $\{(\text{NNP})(\text{NNP})\}$ and named entity tag (PERSON). Thus, we can treat the layers of annotation that build on the PoS tag as a fragment or as an indivisible unit to index. We refer to this indivisible unit of variable word sequence attached to its $k-1$ annotation layers as a *k-fragment*. In this design space, semantic queries can be answered, if and only if the variable -length word sequence and all the attached annotations for it are known to the user. While, maintaining complete fragments in an inverted index is cheap, it poses little utility since the length of the word sequences accompanying a particular annotation is variable. Therefore, both these design choices are ill suited for implementing our data model. We thus look at the design space between these two extremes, which is discussed next.

GYANI DESIGN. We have already considered the extremes of indexing word sequences as single words to variable length fragments. A better design choice, is thus to consider word sequences as combinations of fixed size n-grams. The N-GRAM INDEX, consists of n-grams pointing to a list of postings that contain the metadata $(\langle id, ts\rangle)$ and a list S of positional spans $(s = [i, j])$ of the document in which it occurs. Specifically, we construct unigram $(n = 1)$, bigram $(n = 2)$ and trigram $(n = 3)$ indexes to retrieve word sequences. In addition, we construct ANNOTATION INDEXES to maintain the locations of semantic annotations in the document collection. To index the variable-length word sequences with their annotations, we consider *2-fragments* that are pairwise combinations of the word sequences with one of the attached annotations. These, binary fragments are the basic indexing units for the 2-FRAGMENT INDEXES. To locate sentence boundaries and maintaining positional constraints we index all layers in a DIRECT INDEX.

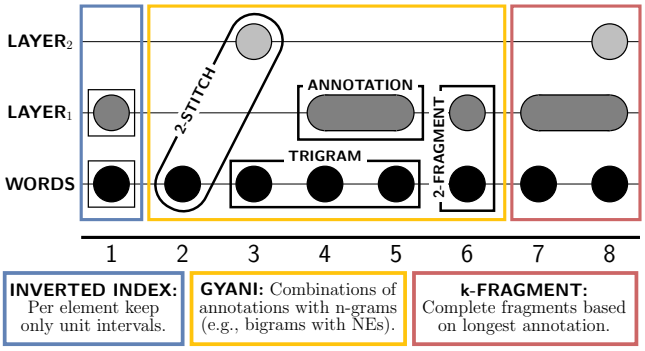

Figure 3: Design space.

Knowledge-centric tasks such as information extraction rely on extraction templates leveraging regular expressions between annotations and word sequences. Processing regular expression queries involving word sequences and annotations can be extremely expensive if only ANNOTATION INDEXES are utilized, as the posting lists for each annotation can span the entire document collections (e.g., $\langle(\text{LOCATION})\text{ city of }(\text{LOCATION})\rangle$). This is a problem similar to indexing stopwords in document collections. In order to execute such complex queries, we additionally index combinations of elements across layers that are shifted. Indexing units that arise from combination of word sequences and ordered co-occurring annotations from $k-1$ different layers are termed as as *k-stitches*. For example by combining a word sequence and named entity we can create a 2-stitch: $\langle\text{city of }(\text{LOCATION})\rangle$. Thus, we additionally create 2-STITCH INDEXES that record pair-wise ordered co-occurrences of unigrams, bigrams, and trigrams with annotations within sentence boundaries. These ordered co-occurring combinations of word sequences and annotations convey relations similar to those obtained by dependency parsing. A relation in a dependency parse tree connects within a sentence two words with a particular relationship which is a subset of the combinations modeled by our 2-stitch indexes.

We further need to store information regarding sentence boundaries in order to restrain the text regions obtained from the indexes. Sentence boundaries can be obtained by retrieving them from the direct index (where they are stored as separate annotations). Or they can be added as an additional payload to the N-GRAM INDEXES. For the latter case, we make the design choice to store sentence numbers that further allows us to easily compute relaxations to context windows of more than one sentence.

2.5 Query Processing

We now discuss how the indexes in GYANI are combined to retrieve documents for queries expressed in our language.

Retrieving Word & Annotated Sequences. We first illustrate how the basic tokens in our query language are retrieved. Word sequences are retrieved from an N-GRAM INDEX, which requires normalizing the requested word sequence into a list of n-grams and subsequently merging their positions. This method of retrieving any arbitrary length word sequence is shown in Algorithm 1. The retrieval of annotated word sequences using the STACK operator requires directly querying the 2-FRAGMENT INDEX, and no additional processing is required.

Algorithm 1: Processing word sequences.

Input : $Q = \langle w_1 \dots w_k \rangle$
Output: Posting List, L, containing document metadata $\langle id, ts \rangle$ containing Q and its positions $s = [i, j]$.
Function NGramQuery(Q)
  N ← generate a list of n-grams from Q
  L ← retrieve posting list for N[0] from n-GRAM INDEX
  L′ ← ∅ // temporary variable
  for $(i \leftarrow 1; i < (k - n + 1); i++)$ **do**
    L′ ← retrieve posting list for N[i]
    L ← merge positions for postings in L & L′ s.t.each position for a posting in L is before the position for the same posting in L′
  return L

Processing Regular Expressions. We now consider how to process queries involving regular expressions operators. In particular, we consider the operator $\boxed{.+}$ in Algorithm 2. The operators $\boxed{.*}$, $\boxed{.?}$, and $\boxed{\cdot}$ can be implemented similarly. The $\boxed{.+}$ operator is a binary operator that consumes as its left and right operand two posting lists. The output list contains postings indicating each occurrence of the element in the left operand is succeeded by the element in the right operand. This join operation is specified with the help of the operator $\bowtie_{\boxed{.+}}$. The $\bowtie_{\boxed{.+}}$ operator takes as input the positions of the elements in the left S_l and right operand S_r, and the gap constraint Δ. The gap constraint Δ indicates the permissible interval size between the elements of the left and right operand. For the operator $\boxed{.+}$, the gap constraint Δ is equal to 2. For the operators $\boxed{.*}$, $\boxed{.?}$, and $\boxed{\cdot}$ the constraints are, $\Delta \geqslant 1$, $\Delta \in [1, 2]$, and $\Delta = 2$.

Processing Projection Operators. We now consider how to process queries involving PHRASE PROJECTION operators that project the regular expression match to within a sentence. Processing these operators can be done using combination of different indexes.

First, we can process PHRASE PROJECTION operators using a combination of DIRECT and N-GRAM indexes. Here, the DIRECT index is used to identify sentence boundaries and to identify the positions of the annotations in its respective layer of a document. Since, these regular expressions with annotations or words return results at the sentence level, we instantiate each operator to expand the suffix ($\boxed{\blacktriangleright}$); or the prefix ($\boxed{\blacktriangleleft}$); or both suffix and prefix ($\boxed{\blacklozenge}$) of the attached word sequence. For instance, for the query $\langle$war started on$\rangle$(DATE), involves processing the word sequence $\langle$war started on$\rangle$ and selecting those sentences in which it occurs with a date as a suffix. While, the query (WORD)+ $\langle$war started on$\rangle$ (DATE) involves expanding both a prefix and suffix expansion. The processing for the suffix expansion of $\boxed{\ell+}$ operator is shown in Algorithm 3. The operators $\boxed{\ell*}$, $\boxed{\ell?}$, and $\boxed{\ell}$ and the associated type of expansion are implemented in a similar manner.

Second, we can implement the query processing for PHRASE PROJECTION operators using 2-STITCH and N-GRAM indexes. The processing of these operators is done by first looking up the ordered co-occurrences of the word sequences and annotations from the 2-STITCH indexes and then merging them based on overlaps in the text regions they share within the same sentence. For example, for the query ⟨ (ORGANIZATION) acquired the start-up for (MONEY) ⟩, we can lookup the 2-stitches ⟨ (ORGANIZATION) acquired the start-up for⟩ and ⟨ acquired the start-up for (MONEY) ⟩ and merge them based on overlapping text regions.

Algorithm 2: Processing the $\boxed{.+}$ operator.

Input : Posting lists L_l and L_r corresponding to the left and right operands of $\boxed{.+}$ operator, respectively.
Output: Posting List, $L \leftarrow L_l \boxed{.+} L_r$.
Function $\boxed{.+}(L_l, L_r)$
  R ← find all common metadata for postings in L_l & L_r
  L ← ∅, S ← ∅
  foreach $\langle id, ts \rangle \in R$ **do**
    S ← $\bowtie_{\boxed{.+}}$(*positionsfor* $\langle id, ts \rangle$ *in* L_l,*positionsfor* $\langle id, ts \rangle$ *in* L_r,2)
    L ← L.append(new ⟨ $\langle id, ts \rangle$, S ⟩)
  return L
Function $\bowtie_{\boxed{.+}}(S_l, S_r, \Delta)$
  S ← ∅
  if *the last interval in* S_r *lies before the first interval in* S_l **then**
    return S
  if *first interval in* S_r *is before the first interval in* S_l **then**
    S_r ← remove intervals from the front of S_r until the first interval in it is after the first interval in S_l
  for $(i \leftarrow 0; i < |S_l|; i++)$ **do**
    for $(j \leftarrow 0; j < |S_r|; j++)$ **do**
      if $(S_l[i]$ *is before* $S_r[j]) \wedge (S_r[j].end - S_r[j].begin \geqslant \Delta)$ **then**
        S ← S.append($[S_l[i]$.begin, $S_r[j]$.end$]$)
  return S

Algorithm 3: Processing the $\boxed{\blacktriangleright+}$ operator.

Input : Posting list L.
Output: Expanded Posting List, L, using suffix expansion.
Function $\boxed{\blacktriangleright+}$(L)
  S ← ∅ // holds the annotation layer
  B ← ∅ // holds the sentence boundaries
  N ← ∅ // holds the new positions after expansion
  count ← 0 // holds the annotation count
  foreach *posting* P *in list* L **do**
    S ← retrieve the annotation layer containing ℓ for the metadata $\langle id, ts \rangle$ using the DIRECT INDEX
    B ← retrieve the sentence boundaries for the metadata $\langle id, ts \rangle$ using the DIRECT INDEX
    count ← 0, N ← ∅
    foreach *position interval* $[x, y]$ *in posting* P **do**
      count ← count the number of annotations that are equal to ℓ between the positional interval spanning y and the end of the sentence using S and B
      if count > 1 **then**
        N ← N.append($[x, y]$)
        N ← N.append(position of the annotations found between y and the end of sentence)
    replace positions of P with expanded positions N
  return L

Third and finally, we can use a combination of ANNOTATION, N-GRAM, and DIRECT indexes for processing PHRASE PROJECTION operators. To do so, we can compute a regular expression join between the word sequence obtained from the N-GRAM index and annotation obtained from the ANNOTATION index. After the join, we can restrict the text region to within one sentence using the $\boxed{\pi_1}$ operator. The $\boxed{\pi_k}$ operator restricts the positional spans of a posting list that are input to the operator to lie within a span of k sentences. The implementation for this operator is given in Algorithm 4. For instance, the following query $\boxed{\pi_1}$((LOCATION) $\boxed{.*?}\langle$declared war on$\rangle$), yields documents where the resulting text regions after the $\boxed{.*?}$ operation lie within a sentence of the retrieved documents.

3 EVALUATION

We now describe the evaluation setup of the experiments that includes a description of he document collections. We then show how the testbeds for the knowledge-centric tasks were constructed. Finally, we discuss the results obtained for the experiments.

Table 1: Document collection statistics. The table shows the sizes of annotated collections as well as annotation statistics.

COLLECTION	SIZE (GB)	$n_{\text{documents}}$	n_{words}	$n_{\text{sentences}}$	$n_{\text{part-of-speech}}$	$n_{\text{named entity}}$	n_{time}	n_{numbers}
NEW YORK TIMES	49.7	1,855,623	1,058,949,098	54,024,146	1,058,949,098	107,745,696	15,411,681	21,720,437
WIKIPEDIA	156.0	5,327,767	2,807,776,276	192,925,710	2,807,776,276	444,301,507	97,064,344	82,591,612
GIGAWORD	193.6	9,870,655	3,988,683,648	181,386,746	3,988,683,648	517,420,195	72,247,124	102,299,554

Algorithm 4: Processing $\boxed{\pi_k}$ operator.

```
Input  : Posting List L and sentence window k.
Output : Posting List L' such that each position lies within a k sentence
         window.
Function ⌈π_k⌉(L, k)
    foreach posting P in list L do
        S ← ∅                                   // modified positions for P
        B ← retrieve the sentence boundaries for the metadata ⟨id, ts⟩
        using the DIRECT INDEX
        if k > 1 then
            B ← coalesce(B, k)
        foreach position interval [x, y] in posting P do
            foreach position interval [m, n] in B do
                if [m, n].contains([x, y]) then
                    S ← S.append([x, y])
        replace positions of P with modified positions S
    return L
Function coalesce(B, k)
    S ← ∅                                       // coalesced positions
    b, e ← −1                                   // begin and end for intervals
    for (i ← 0; i < |B|.size − (k − 1); i++) do
        b ← B[i].begin
        e ← B[i + k − 1].end
        S ← S.append([b, e])
    return S
```

3.1 Evaluation Setup

Document Collections. We considered three large document collections to index with GYANI. Statistics regarding the collections are summarized in Table 1. The first document collection is the New York Times, which comprises of news articles published over a twenty year (1987-2007) time period [2]. The second document collection is the English Gigaword collected from seven distinct English news publishers over a sixteen year (1995-2010) time period [1]. The third and final document collection is the entire English Wikipedia [4] (we use the snapshot available on March 13th, 2017). An important aspect of all the aforementioned document collections is that, they are written in well poised grammar and language, so that the automated annotations obtained via NLP tools are of high quality.

Annotating Document Collections. Each document in the collections was annotated with four different types of semantic annotations. We utilized the Stanford Core NLP [22] toolkit to annotate the documents with part-of-speech, named entities, temporal expressions, and numerical quantities. The processing for the documents was done as follows. First, each document's text content is created by concatenating the headline, the article body, and other auxiliary keyword or classification terms provided as metadata into one long document string. Second, the publication date for the news article is obtained; for Wikipedia pages we used the document creation time. Third, the document string is fed into the annotation pipeline, which performs sentence boundary detection, tokenization, and tags each token with the aforementioned types of annotations. Fourth, for each token we analyze the type of annotation performed and subsequently create the layer elements. For the parts of speech tags we keep the tag as the annotation element and unit interval spans as positions. The Stanford Core NLP named entity annotator provides ten different classes of entities. We divide these classes as per our requirement. For the named entity layer, we consider all the class tags as annotation elements: PERSON, ORGANIZATION, LOCATION, DATE, TIME, DURATION, MONEY, PERCENT, NUMBER, and ORDINAL. For the temporal expressions, we consider for resolution the classes: DATE, TIME, and DURATION. For the numerical quantities we consider for resolution the classes: MONEY, PERCENT, NUMBER, and ORDINAL. Finally, the document string, along with the strings containing the annotation layers as per our data model are stored together with the metadata information containing the timestamp (determined using publication date or creation time) and its identifier (determined using the hash of the publisher supplied identifier string or the title string).

Table 2: Index sizes in Gigabytes (GB).

INDEX TYPE	NYT	WIKIPEDIA	GIGAWORD
DIRECT	18.80	44.80	52.40
N-GRAM	45.90	126.30	154.40
ANNOTATION	2.39	7.65	9.33
2-FRAGMENT	6.30	23.10	24.16
2-STITCH	141.00	473.00	542.40

Implementation Details. The implementation of the entire infrastructure was done in Java. All document processing and indexing was done in a distributed manner over a cluster of twenty machines running the Cloudera CDH 5.90 distribution of Hadoop. All machines in the cluster were equipped with Intel Xeon CPUs with up to 24 cores and a clock speed of up to 3.50 GHz, up to 128 GB of primary memory, and up to eight 4 TB hard disks as secondary storage. We utilized the 1.2.0 CDH 5.9.0 version of HBase for implementing our indexes.

GYANI Indexes. We instantiated the index types discussed in Section 2.4 for each of the document collections. Posting lists are compressed using the PForDelta compression technique [5]. We summarize the index sizes for the various types in Table 2.

3.2 Knowledge-Centric Tasks

We next describe the structure of the queries for the five knowledge-centric tasks used in our evaluation.

Information Extraction (IE) Task. To construct information extraction templates, we utilize the paraphrases of relations [25] present in the Yago knowledge graph. The information extraction templates are constructed as follows. First, for each of the Yago relations the domain of the subject and the range of the object is identified. For example, for the predicate wasBornIn the domain of the subject is PERSON and the range of the object is LOCATION. Second, for the given relation, we look up how the relation is expressed in text using a paraphrase dictionary [25]. For example, the paraphrases for wasBornIn are grew up, returned to, and raised in. Finally, we combine the subject, paraphrase of the relation, and the object to form the information extraction template. For instance, a template for the relation wasBornIn is: (PERSON) ⟨raised in⟩ (LOCATION).

Table 3: Testbed statistics.

TASK	n_{query}	μ_{word}	$\mu_{annotation}$
TASK-IE	56,261	2.50	2.22
TASK-RE-NEWS	116,157	83.77	0.00
TASK-QA-NEWS	828,208	37.68	1.60
TASK-FS-NEWS	1,618,377	126.61	0.83
TASK-RE-WIKI	861,235	67.89	0.00
TASK-QA-WIKI	18,151,907	19.47	1.48
TASK-FS-WIKI	26,164,545	81.32	0.69
TASK-SQ	4,589	6.15	2.53

Relation Extraction (RE) Task. The aim of the relation extraction task to identify the textual patterns of the predicate given its subject and object arguments. From the paraphrase dictionary [25], we also have concrete instances of subject-object pairs identified in the New York Times and Wikipedia. However, most of the named entities can be expressed in myriad surface forms. In order to capture the different surface forms for a given named entity we turn to the redirectedFrom relation in the Yago knowledge graph. To create queries for this task we proceed as follows. First, we distill the unique instances of the subject-object pairs identified by [25] in both news and encyclopedic sources. Second, we look up surface forms for the named entities contained in the subject and object arguments from Yago's redirectedFrom relation. Third, we combine the named entity and its various surface forms as UNION wildcard clause, e.g., [kennedy | jfk]. Finally, we construct the relation extraction template by combining the subject and objects arguments with $\boxed{\ell*}$ operator. For example, [john f. kennedy | jfk | john fitzgerald kennedy] (WORD)* [ronald reagan | 40th president of the united states | ronnie reagan].

Question Answering (QA) Task. The aim of the question answering task is to retrieve sentences as candidate answers in response to a query with annotation wildcards. For this task we consider the textual patterns of the predicates that occur between the subject-object instances, also available from the dataset in [25]. To create the queries for this task we carry out the following steps. First, we consider only the subject's named entity and its surface forms from the subject-object pairs. Second, we combine the textual pattern detected as a predicate for the given subject-object pair [25]. Finally, we replace the object with its appropriate range type with a $\boxed{\ell}$ annotation wildcard operator. For example, [microsoft | office corporation] to work closely with (ORGANIZATION).

Fact Spotting (FS) Task. The aim of the fact spotting task is to retrieve sentences that are evidences of facts from a knowledge graph. The process of creating queries for this task is similar to that of question answering task except that we keep the named entity and its surface forms for the object argument. An example of a query in this task is [microsoft | office corporation] to crush [netscape | devedge].

Semantic Search (SQ) Task. The aim of the semantic search task is to demonstrate the ability to express queries containing word sequences overloaded with a semantic meaning. To create queries for this task, we turn to a compendium of important events compiled by the New York Times called "On this Day" events [3]. Each event in this compendium consists of a date and an accompanying textual description. The steps involved in creating the queries are as follows. First, each of the event descriptions are run through an annotation process similar to the one applied to the document collections. Second, we combine annotations and word sequences from the named entity, time, and numerical quantity layers to form stacked phrases. Finally, we combine them with the Boolean $\boxed{\wedge}$ operator to form the semantic query. For example, (mohandas k. gandhi)$\oplus$(PERSON) $\boxed{\wedge}$ (india)$\oplus$(LOCATION).

In Table 3 we summarize the entire testbed statistics. As can be noticed from query length in Table 3, the queries in our testbed are quite complex, lengthy, and verbose. The dataset is publicly available at the following website:

http://resources.mpi-inf.mpg.de/dhgupta/data/cikm2018/.

3.3 Experimental Results

We evaluate GYANI for efficiency by measuring end-to-end query execution times. For each of the task we sampled 100 queries from the appropriate testbed for evaluation. Each sample is executed three times and the average execution time is reported. We execute each task under two settings: warm and cold caches. In the warm cache setting, each query is executed once to bring the relevant posting lists into the main memory of the HBase cluster and then executed three more times to measure its execution time. In the cold cache settings, the sample of queries is executed three times by shuffling the order of query execution between rounds. The time measured consists of retrieving the posting lists from HBase and further performing the necessary operations dictated by the tasks, which may further involve accessing the direct index. In order to minimize interruptions due to garbage collection we utilize the concurrent garbage-first garbage collector (G1GC). Experiments are run on two servers capable of handling high I/O bound jobs as our front-end and the Hadoop cluster acting as our back-end storage. Each server consists of up to two Intel Xeon processors with up to 96 cores, clocked at 2.66 GHz and up to 1.48 TB of primary memory.

Baselines. The baselines we evaluate are aligned with respect to the design choices explored in Section 2.4. We first measure the time to scan the entire document collection without any indexes. This NAÏVE DESIGN thus establishes a worst case upper bound to execute a single query by finding the pattern in the entire document collection. This simple design thus imitates GREP in an embarrassingly parallel manner. The first baseline implements the INVERTED INDEX DESIGN and is denoted by TEXTI. This baseline considers only the word sequences that can be obtained by combining N-GRAM indexes and the DIRECT index. With TEXTI we test how efficiently an infrastructure can retrieve candidate documents relying only on text. To a certain extent, TEXTI simulates the "neighbor index" [9], where we are forced to access the DIRECT index to match the context around the words during query processing. To execute the queries with N-GRAM indexes and DIRECT index, we identify the sentences (using the DIRECT index) containing the text only arguments of the query and apply the AND operator between the obtained posting lists to obtain the final result. The second baseline considers N-GRAM indexes, ANNOTATION indexes, and DIRECT index to evaluate regular expression queries. We call this baseline ANNI. The ANNI baseline considers posting lists for annotations when evaluating regular expression queries for the knowledge-centric tasks. It additionally resorts to the DIRECT index for identifying sentence boundaries when restraining the results to within one sentence. We evaluate our infrastructure GYANI that leverages the complete set of indexes

Table 4: GREP baseline times in seconds.

NEW YORK TIMES	WIKIPEDIA	GIGAWORD
111.00	322.00	396.00

proposed. With GYANI, however, we leverage the sentence identifiers stored within the N-GRAM indexes to restrict the regular expression matches to within one sentence. In order to execute the same set of queries for all the three infrastructures, we choose those queries where the predicate does not contain any annotation wildcards and the subject and object regular expressions are either (PERSON), (ORGANIZATION), or (LOCATION). To execute the queries against TEXTI we replace (PERSON), (ORGANIZATION), or (LOCATION) operator with (WORD)+. For the semantic query task, only the TEXTI baseline is applicable where only the N-GRAM indexes are used to execute the word-only versions of the semantic queries.

Results. We now discuss the results for TEXTI, ANNI, and GYANI at the five different tasks over three document collections.

Grepping the Entire Document Collection. We first report the results for the baseline GREP that involves scanning (i.e., matching the .* operator) the entire document collection on our Hadoop cluster. This is akin to running GREP as an embarrassingly parallel task per query over a large document collection. We report the time taken to scan the three different document collections for a single query in Table 4. The time required for scanning the document collections is proportional to the collection's size. The minimum amount of time was required for the New York Times which is the smallest collection amongst the three. While, Gigaword required the most time as it was the largest amongst the three collections.

End-to-end Query Execution Times. We now discuss the results obtained for query execution times over the three different document collections. The results are reported in Table 5 are in seconds. Note that our system and the baselines shown in Table 5 retrieve equivalent sets of text regions as results. All values marked with $\vartriangle$ and $\blacktriangle$ indicate statistically significant results ($p \leqslant 0.05$) with respect to TEXTI and ANNI respectively. The significance was measured using the paired t-test. For the information extraction task we can see that our proposed infrastructure GYANI drastically brings down execution times from several seconds (several minutes in case of Gigaword) to within milliseconds per query. The drastic decrease in execution time can be attributed to the observation that GYANI relies on the 2-STITCH indexes and does not resort to ANNOTATION indexes (which ANNI does) and DIRECT index (which both ANNI and TEXTI do). For the question answering task, our proposed approach again delivers results within milliseconds as compared to the other two baselines. The gains again can be attributed to the same observation as with the IE task. For the relationship extraction and fact spotting task the performance of GYANI is better or at par with ANNI. However, compared to TEXTI the query execution costs are brought down from several minutes to few seconds. The gain that GYANI attains over the other baselines is due to the fact that it does not resort to the DIRECT index for identifying sentence boundaries (it uses the sentence numbers available within the N-GRAM indexes) when evaluating the regular expressions between the query arguments. For the semantic query task, we can see that by directly leveraging the 2-fragment indexes GYANI identifies the result more quickly than the TEXTI baseline, which uses only N-GRAM indexes, as it can not disambiguate their semantics.

Table 5: Query execution times in seconds.

	TASK	TEXTI (COLD) (S)	ANNI (COLD) (S)	GYANI (COLD) (S)
NEW YORK TIMES	IE	8.38 ± 20.61	12.32 ± 12.41	$^{\vartriangle\blacktriangle}$ 0.01 ± 0.02
	QA	9.68 ± 18.11	9.18 ± 0.82	$^{\vartriangle\blacktriangle}$ 0.15 ± 0.16
	FS	7.10 ± 34.49	0.29 ± 0.57	$^{\vartriangle}$ 0.29 ± 0.58
	RE	41.92 ± 122.89	$^{\vartriangle}$ 2.75 ± 9.98	$^{\vartriangle}$ 2.41 ± 8.30
	SQ	1.22 ± 3.96	—	0.69 ± 2.98
	TASK	TEXTI (WARM) (S)	ANNI (WARM) (S)	GYANI (WARM) (S)
NEW YORK TIMES	IE	3.53 ± 10.97	11.55 ± 11.15	$^{\vartriangle\blacktriangle}$ 0.01 ± 0.01
	QA	4.81 ± 9.70	9.13 ± 0.42	$^{\vartriangle\blacktriangle}$ 0.09 ± 0.15
	FS	4.39 ± 21.79	0.30 ± 0.55	$^{\vartriangle}$ 0.29 ± 0.51
	RE	29.60 ± 111.56	$^{\vartriangle}$ 2.73 ± 9.90	$^{\vartriangle}$ 2.42 ± 8.25
	SQ	0.96 ± 2.98	—	0.86 ± 3.61
	TASK	TEXTI (COLD) (S)	ANNI (COLD) (S)	GYANI (COLD) (S)
WIKIPEDIA	IE	17.73 ± 35.35	51.52 ± 33.08	$^{\vartriangle\blacktriangle}$ 0.11 ± 0.25
	QA	21.10 ± 73.18	28.30 ± 19.33	$^{\vartriangle\blacktriangle}$ 0.21 ± 0.63
	FS	5.76 ± 38.32	0.46 ± 0.96	0.46 ± 0.95
	RE	105.31 ± 298.58	$^{\vartriangle}$ 2.50 ± 7.00	$^{\vartriangle}$ 2.16 ± 5.81
	SQ	2.64 ± 4.50	—	2.50 ± 6.61
	TASK	TEXTI (WARM) (S)	ANNI (WARM) (S)	GYANI (WARM) (S)
WIKIPEDIA	IE	7.18 ± 14.88	43.69 ± 19.37	$^{\vartriangle\blacktriangle}$ 0.06 ± 0.16
	QA	8.25 ± 34.22	25.92 ± 1.55	$^{\vartriangle\blacktriangle}$ 0.14 ± 0.55
	FS	2.49 ± 14.33	0.49 ± 0.99	0.46 ± 0.93
	RE	39.73 ± 113.50	$^{\vartriangle}$ 2.36 ± 6.42	$^{\vartriangle}$ 2.15 ± 5.81
	SQ	2.58 ± 4.31	—	$^{\vartriangle}$ 1.43 ± 2.70
	TASK	TEXTI (COLD) (S)	ANNI (COLD) (S)	GYANI (COLD) (S)
GIGAWORD	IE	36.69 ± 88.43	65.10 ± 51.88	$^{\vartriangle\blacktriangle}$ 0.07 ± 0.10
	QA	57.78 ± 109.89	43.93 ± 2.99	$^{\vartriangle\blacktriangle}$ 0.39 ± 0.54
	FS	52.41 ± 212.42	$^{\vartriangle}$ 1.31 ± 2.58	$^{\vartriangle\blacktriangle}$ 1.25 ± 2.47
	RE	316.52 ± 1048.42	$^{\vartriangle}$19.33 ± 83.11	$^{\vartriangle}$ 15.69 ± 61.45
	SQ	5.25 ± 7.82	—	$^{\vartriangle}$ 3.65 ± 5.96
	TASK	TEXTI (WARM) (S)	ANNI (WARM) (S)	GYANI (WARM) (S)
GIGAWORD	IE	12.44 ± 33.78	61.88 ± 33.04	$^{\vartriangle\blacktriangle}$ 0.13 ± 0.96
	QA	19.15 ± 38.10	43.11 ± 2.43	$^{\vartriangle\blacktriangle}$ 0.31 ± 0.50
	FS	32.75 ± 172.51	1.27 ± 2.51	1.26 ± 2.46
	RE	256.10 ± 1047.16	$^{\vartriangle}$ 18.11 ± 75.08	$^{\vartriangle}$ 15.67 ± 61.13
	SQ	5.21 ± 7.23	—	$^{\vartriangle}$ 3.48 ± 5.56

Summary. Summing up our experimental results across tasks and document collections, we observe that the indexes that constitute GYANI consume at most 5.62× the space required by the uncompressed semantically annotated document collections. SQ and FS are the tasks that profit least with speed ups in response time of 1.12× and 5.41× respectively. For IE, QA, and RE, as more complex tasks, GYANI achieves impressive speed ups of at least 95.69×, 53.44×, and 12.23×, respectively. We designed GYANI as a versatile tool supporting various knowledge-centric tasks. We point out that, should only specific tasks need to be supported, a subset of indexes would suffice.

4 RELATED WORK

Searching Semi-Structured Text. The earliest attempts in information extraction from semi-structured text documents relied on region algebras [11, 12, 27]. The PAT system [27] supported expressions that could match SGML tags to match regions of text for information extraction. Their approach also allowed the user to query for regions of text with the help of the *region expressions*. Clarke et al. [11] proposed a data model that relied on maintaining *generalized concordance lists* to index positional spans for SGML tags. A clear contrast between these early works and our work is in accommodating semantic annotations in text.

Lalmas [20] provides an overview of work on XML retrieval including expressive query languages such as XPath and XQuery. While documents with semantic annotations could be represented as XML, this would entail a blowup in space and formulating regular expression queries for knowledge-centric tasks in the aforementioned languages is all but intuitive.

Miller and Myers [24] were the first to realize the unavailability of popular Unix delimited-text manipulators such as GREP for semi-structured documents. Their data model represented text regions by Cartesian coordinates. The authors then leveraged R*-trees to intersect and compute proximity between rectangles. Cho and Rajagopalan [10] focused on how to allow queries to contain regular expressions at character level. Their proposal was the concept of a k-gram index that indexed selective n-grams for efficient regular expression based search. However, both these approaches do not provide any scope for handling semantically annotated text.

Searching Annotated Text. Ferrucci and Lally [14] presented *Unstructured Information Management Architecture* (UIMA), a comprehensive and integrated suite of annotators and text analytics pipeline. UIMA supported modeling implicit annotations present in text as "common analysis structure" that allowed overlaying of annotations to cover common portions of text. Cafarella and Etzioni [9] proposed the "neighbor index" that provided Boolean queries involving phrases, annotations, and functions over annotations. Both the UIMA framework and "neighbor index" aimed at providing the functionality of Boolean queries (not regular expressions) over annotated text.

Li and Rafiei [21] described how to execute queries involving wildcards, part-of-speech tags, and words. Their implementation relied on commercial search engines for retrieving text snippets. Bast and Buchold [7] proposed an index architecture that incorporates both knowledge-graph relations associated with entities and the contextual text containing that entity. This thus allows for search over a combined index of knowledge graph and unstructured text. Massung et al. [31] investigated how to index aggregated feature vector representations of text along with documents for a unified framework for analysis. A recent survey on information extraction over text and knowledge graphs [8] lacks any mention of an implementation that allows for structured search involving word sequences, annotations, and regular expressions.

The computational linguistics community also looked into query languages for annotated corpora (including additional annotations such as dependencies) [15, 19]. Scalability, though, has not been a focus in those works and the considered corpora were at least an order of magnitude smaller than the ones we consider in this work.

Searching Text Using RDBMS. Solutions to enable structured search over semantically annotated text can also addressed using conventional database technologies [13, 32]. However, none of these approaches supports wildcard operators. By adopting the RDBMS approach, Zhou et al. [32] described a data model that encodes words, annotations, the confidence of the accompanying annotations, and its positional span. Queries over this data model are mapped to SQL queries for execution. Cornacchia et al. [13] considered the problem of implementing IR systems using array databases with efficient storage schemes for sparse arrays.

5 CONCLUSIONS

In this work, we described GYANI, an infrastructure for supporting sophisticated knowledge-centric tasks at scale. We first proposed a novel data model that accommodates word sequences and layers of semantic annotations associated with them. We then proposed a novel language that allows the user to express queries consisting of regular expressions over word sequences and annotations. To allow for fast query execution times, we further described the appropriate indexes to support our query language in a complex design space. Finally, our experimental results over five knowledge-centric tasks show the ability of GYANI to efficiently support search and analysis of large semantically annotated document collections for knowledge acquisition at scale.

REFERENCES

[1] English Gigaword Fifth Edition. (https://catalog.ldc.upenn.edu/LDC2011T07).
[2] New York Times (NYT) Corpus. (https://catalog.ldc.upenn.edu/LDC2008T19).
[3] NYT: On This Day. (https://learning.blogs.nytimes.com/on-this-day/).
[4] Wikipedia: The Free Encyclopedia. (https://www.wikipedia.org/).
[5] JavaFastPFOR. (https://github.com/lemire/JavaFastPFOR).
[6] GNU Grep 3.0. (https://www.gnu.org/software/grep/manual/grep.html).
[7] H. Bast and B. Buchhold. An index for efficient semantic full-text search. In *CIKM'13*.
[8] H. Bast et al. Semantic Search on Text and Knowledge Bases. *Foundations and Trends in Information Retrieval* 10, 2-3 (2016), 119–271.
[9] M. J. Cafarella and O. Etzioni. A search engine for natural language applications. In *WWW'05*.
[10] J. Cho and S. Rajagopalan. A Fast Regular Expression Indexing Engine. In *ICDE'02*.
[11] C.L. A. Clarke et al. An Algebra for Structured Text Search and a Framework for its Implementation. *Comput. J.* 38, 1 (1995), 43–56.
[12] M. P Consens and T. Milo. Algebras for Querying Text Regions - Expressive Power and Optimization. *J. Comput. Syst. Sci.* 57, 3 (1998), 272–288.
[13] R. Cornacchia et al. Flexible and efficient IR using array databases. *VLDB J.* 17, 1 (2008), 151–168.
[14] D. A. Ferrucci and A. Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Nat. Lang. Eng.* 10, 3-4 (Sept. 2004), 327–348.
[15] E. Frick et al. Evaluating Query Languages for a Corpus Processing System. In *LREC'12*.
[16] D. Gupta and K. Berberich. Identifying Time Intervals for Knowledge Graph Facts. In *WWW'18*.
[17] M. A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *COLING'92*.
[18] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2nd ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.
[19] T. Krause et al. graphANNIS: A Fast Query Engine for Deeply Annotated Linguistic Corpora. Corpus Linguistic Software Tools 31, 1 (2016), 1–25.
[20] M. Lalmas. XML Retrieval. Morgan & Claypool Publishers (2009).
[21] H. Li. Data extraction from text using wild card queries. *Masters Abstracts International* (2006).
[22] C. D. Manning et al. The Stanford CoreNLP Natural Language Processing Toolkit. In *ACL'14*.
[23] S. Metzger et al. S3K: seeking statement-supporting top-K witnesses. In *CIKM'11*.
[24] R. C Miller and Brad A Myers. Lightweight Structured Text Processing. *USENIX Annual Technical Conference, General Track* (1999).
[25] N. Nakashole et al. PATTY: A Taxonomy of Relational Patterns with Semantic Types. In *EMNLP-CoNLL'12*.
[26] K. Panev and K. Berberich. Phrase Queries with Inverted + Direct Indexes. In *WISE'14*.
[27] A. Salminen and F. W. Tompa. PAT expressions: an algebra for text search. *Acta Linguistica Hungarica* (1994).
[28] D. Savenkov and E. Agichtein. When a Knowledge Base Is Not Enough: Question Answering over Knowledge Bases with External Text Data. In *SIGIR'16*.
[29] F. M. Suchanek et al. YAGO: A Large Ontology from Wikipedia and WordNet. *Web Semant.* 6, 3 (Sept. 2008), 203–217.
[30] H. E. Williams et al. Fast Phrase Querying with Combined Indexes. *ACM Trans. Inf. Syst.* 22, 4 (Oct. 2004), 573–594.
[31] C. Zhai and S. Massung. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. Association for Computing Machinery and Morgan & Claypool, New York, NY, USA.
[32] M. Zhou et al. Data-oriented content query system - searching for data into text on the web. *In WSDM'10*.