DeepDive: Declarative Knowledge Base Construction

By Ce Zhang, Christopher Ré, Michael Cafarella, Christopher De Sa, Alex Ratner, Jaeho Shin, Feiran Wang, and Sen Wu

Abstract

The dark data extraction or knowledge base construction (KBC) problem is to populate a relational database with information from unstructured data sources, such as emails, webpages, and PDFs. KBC is a long-standing problem in industry and research that encompasses problems of data extraction, cleaning, and integration. We describe DeepDive, a system that combines database and machine learning ideas to help to develop KBC systems. The key idea in DeepDive is to frame traditional extract-transform-load (ETL) style data management problems as a single large statistical inference task that is declaratively defined by the user. DeepDive leverages the effectiveness and efficiency of statistical inference and machine learning for difficult extraction tasks, whereas not requiring users to directly write any probabilistic inference algorithms. Instead, domain experts interact with DeepDive by defining features or rules about the domain. DeepDive has been successfully applied to domains such as pharmacogenomics, paleobiology, and antihuman trafficking enforcement, achieving human-caliber quality at machine-caliber scale. We present the applications, abstractions, and techniques used in DeepDive to accelerate the construction of such dark data extraction systems.

1. INTRODUCTION

The goal of knowledge base construction (KBC) is to populate a structured relational database from unstructured input sources, such as text documents, PDFs, and diagrams. As the amount of available unstructured information has skyrocketed, this task has become a critical component in enabling a wide range of new analysis tasks. For example, analyses of protein-protein interactions for biological, clinical, and pharmacological applications²⁹; online human trafficking activities for law enforcement support; and paleological facts for macroscopic climate studies³⁶ are all predicated on leveraging data from large volumes of text documents. This data must be collected in a structured format in order to be used, however, and in most cases doing this extraction by hand is untenable, especially when domain expertise is required. Building an automated KBC system is thus often the key development step in enabling these analysis pipelines.

The process of populating a structured relational database from unstructured sources has also received renewed interest in the database community through high-profile start-up companies, established companies such as IBM's Watson,^{5, 15} and a variety of research efforts.^{9, 26, 31, 41, 46} At the same time, the natural language processing and machine learning communities are attacking similar problems.^{3, 12, 22}

Although different communities place differing emphasis on the extraction, cleaning, and integration phases, all seem to be converging toward a common set of techniques that includes a mix of data processing, machine learning, and engineers-in-the-loop.^a

Here, we discuss DeepDive, our open-source engine for constructing knowledge bases with human-caliber quality at machine-caliber scale (Figure 1). DeepDive takes the viewpoint that in information extraction, the problems of extraction, cleaning, and integration are not disjoint algorithmic problems, though the database community has treated them as such for several decades. Instead, these problems can be more effectively attacked jointly, and viewed as a single statistical inference problem that takes all available information into account to produce the best possible end result. We have found that one of the most harmful inefficiencies of traditional pipelined approaches is that developers struggle to understand how changes to the separate extraction, cleaning, or integration modules improve the overall system quality, leading them to incorrectly distribute their development

Figure 1. Knowledge base construction (KBC) is the process of populating a structured relational knowledge base from unstructured sources. DeepDive is a system aimed at facilitating the KBC process by allowing domain experts to integrate their domain knowledge without worrying about algorithms.

^a <http://deepdive.stanford.edu>.

The original version of this paper is entitled "Incremental Knowledge Base Construction Using DeepDive" and was published in *Proceedings of the VLDB Endowment*, 2015. This paper also contains content from other previously published work.^{16, 36, 39, 51}

efforts. For example, developers might decide to sink a large amount of time into improving the quality of some upstream component of their pipeline, only to find that it has a negligible effect on end system performance—essentially, running into an Amdahl’s law for quality. In contrast, by formulating the task as a single probabilistic inference problem, DeepDive allows the developer to effectively profile the end-to-end quality of his or her application. We argue that our approach leads to higher quality end-to-end models in less time, which is the ultimate goal of all information extraction systems.

Like other KBC systems, DeepDive uses a high-level declarative language to enable the user to describe application inputs, outputs, and model structure.^{9, 31, 33} DeepDive’s language is based on SQL, but also inherits Markov Logic Networks’ formal semantics to enable users to declaratively describe their KBC task as a type of probabilistic graphical model called a *factor graph*.^{11, 33}

DeepDive uses a standard execution model^{9, 31, 33} in which programs go through two main phases, grounding and inference. In the *grounding* phase, DeepDive evaluates a sequence of SQL queries to produce a *factor graph* that describes a set of random variables and how they are correlated. Essentially, every tuple in the database which represents a *candidate* extraction to be potentially included in the output knowledge base is included as a random variable (node) in this factor graph. In the *inference* phase, DeepDive then takes the factor graph from the grounding phase and performs statistical inference using standard techniques, for example, Gibbs sampling.^{47, 50} The output of inference is the marginal probability of every tuple in the output knowledge base. As with Google’s Knowledge Vault¹² and others,³⁴ DeepDive also produces marginal probabilities that are *calibrated*: if one examined all facts with probability 0.9, we would expect approximately 90% of these facts to be correct. To calibrate these probabilities, DeepDive estimates (i.e., learns) parameters of the statistical model from data. Inference is a subroutine of the learning procedure and is the critical loop. Inference and learning are computationally intense (hours on 1TB RAM/48-core machines).

In our experience, we have found that DeepDive can reliably obtain extremely high quality on a range of KBC tasks. In the past few years, DeepDive has been used to build dozens of high-quality KBC systems by a handful of technology companies, a number of law enforcement agencies via DARPA’s MEMEX program, and scientists in fields, such as paleobiology, drug repurposing, and genomics. Recently, we compared the quality of a DeepDive system’s extractions to those provided by human volunteers over the last 10 years for a paleobiology database, and we found that the DeepDive system had higher quality (both precision and recall) on many entities and relationships. Moreover, on all of the extracted entities and relationships, DeepDive had no worse quality.³⁶ Additionally, the winning entry of the 2014 TAC-KBC competition was built on DeepDive.¹

One key lesson learned was that in all cases, enabling developers to iterate quickly was critical to achieving such high quality. More broadly, we have seen that the process of developing KBC systems for real applications is fundamentally iterative: quality requirements change, new data

sources arrive, and new concepts are needed in the application. Thus, DeepDive’s architecture is designed around a set of techniques that not only make the execution of statistical inference and learning efficient, but also make the entire pipeline incremental in the face of changes both to the data and to the declarative specification.

This article aims at giving a broad overview of DeepDive. The rest of the article is organized as follows. Section 2 describes some example applications of DeepDive and outlines core technical challenges. Section 3 presents the system design and language for modeling KBC systems inside DeepDive. We discuss the different techniques in Section 4 and give pointers for readers who are interested in each technique.

2. APPLICATIONS AND CHALLENGES

KBC plays a critical role in many analysis tasks, both scientific and industrial, and is often *the* bottleneck to answering new and impactful macroscopic questions. In many scientific analyses, for example, one first needs to assemble a large, high-quality knowledge base of facts (typically from the literature) in order to understand macroscopic trends and patterns, for example, about the amount of carbon in the Earth’s atmosphere throughout time³⁶ or all the drugs that interact with a particular gene,²⁹ and some scientific disciplines have undertaken decade-long collection efforts to this end, for example, PaleoDB.org and PharmaGKB.org.

In parallel, KBC has attracted interest from industry^{15, 52} and many areas of academia outside of computer science.^{2, 3, 6, 14, 23, 25, 31, 34, 37, 41, 43, 48} To understand the common patterns in KBC systems, we are actively collaborating with scientists from a diverse set of domains, including geology,⁴⁹ paleontology,³⁶ pharmacology for drug repurposing, and others. We first describe one KBC application we built, called PaleoDeepDive, then present a brief description of other applications built with similar purposes and finally discuss the challenges inherent in building such systems.

2.1. PaleoDB and PaleoDeepDive

Paleontology is based on the description and biological classification of fossils, an enterprise that has been recorded in hundreds to thousands of scientific publications over the past four centuries. One central task that paleontologists have long been concerned with is the construction of a knowledge base about fossils from scientific publications. Existing knowledge bases compiled by human volunteers—for example, PaleoDB—have already greatly expanded the intellectual reach of paleontology and led to many fundamental new insights into macroevolutionary processes and the nature of biotic responses to global environmental change. However, the current process of using human volunteers is usually expensive and time-consuming. For example, PaleoDB, one of the largest such knowledge bases, took more than 300 professional paleontologists and 11 human years to build over the last two decades, resulting in PaleoDB.org. To get a sense of the impact of this database on this field, at the time of writing, this dataset has contributed to 205 publications, of which 17 have appeared in *Nature* or *Science*.

The potential impact of automating this labor-intensive extraction task and the difficulty of the task itself provided an ideal test bed for our KBC research. In particular, we constructed a prototype called PaleoDeepDive³⁶ that takes in PDF documents and extracts a set of paleontological entities and relations (see Figure 2). This prototype attacks challenges in optical character recognition, natural language processing, information extraction, and integration. Some statistics about the process are shown in Figure 3. As part of the validation of this system, we performed a double-blind experiment to assess the quality of PaleoDeepDive versus PaleoDB. We found that PaleoDeepDive achieved accuracy comparable to—and sometimes better than—that of PaleoDB (see Figure 3).³⁶ Moreover, PaleoDeepDive was able to process roughly 10x the number of documents, with per-document recall roughly 2.5x that of human annotators.

2.2. Beyond paleontology

The success of PaleoDeepDive motivates a series of other KBC applications in a diverse set of domains, including both natural and social sciences. Although these applications focus on very different types of KBs, they are usually built in a way similar to PaleoDeepDive. This similarity across applications has motivated us to build DeepDive as a unified framework to support these diverse applications.

Human trafficking. Human trafficking is an odious crime that uses physical, economic, or other means of coercion to obtain labor from human beings, who are often used in sex or factory work. Identifying victims of human trafficking is difficult for law enforcement using traditional means; however, like many other forms of commerce, sex work advertising is now online, where providers of sex services post ads containing price, location, contact information, physical characteristics, and other data. As part of the DARPA MEMEX project, we ran DeepDive on approximately 90M advertisements and 0.5M forum posts, creating two distinct structured tables that included extracted attributes about potentially trafficked workers, such as price, location, phone number, service types, age, and various other attributes that can be used to detect signs of potential trafficking or abuse. In many cases, DeepDive is able to extract these attributes with comparable or greater quality levels than human annotators; for example, on phone number extraction from service ads, DeepDive achieves 99.5% precision and 95.5% re-

call, whereas human annotators only obtain 93% precision and 92.5% recall. MEMEX has been covered on 60 min and other news sources, currently supports the operations of several law enforcement agencies nationwide, and has been used in at least one arrest and conviction.

Medical genetics. The body of literature in the life sciences has been growing at an accelerating speed, to the extent that it has been unrealistic for scientists to perform research solely based on reading and/or keyword search. Numerous manually curated structured knowledge bases are likewise unable to keep pace with exponential increases in the number of publications available online. For example, OMIM is an authoritative database of human genes and Mendelian genetic disorders that dates back to the 1960s, and so far contains about 6000 hereditary diseases or phenotypes, growing at a rate of roughly 50 records per month for many years. Conversely, almost 10,000 publications were deposited into PubMed Central per month last year. In collaboration with Prof. Gill Bejerano at Stanford, we are developing DeepDive applications to create knowledge bases in the field of medical genetics. Specifically, we use DeepDive to extract mentions of direct causal relationships between specific gene variants and clinical phenotypes from the literature that are presently being applied to clinical genetic diagnostics and reproductive counseling.^b

Pharmacogenomics. Understanding the interactions of chemicals in the body is a key to drug discovery. However, the majority of this data resides in the biomedical literature and cannot be easily accessed. The Pharmacogenomics Knowledge Base is a high quality database that aims to annotate the relationships between drugs, genes, diseases, genetic variation, and pathways in the literature. In collaboration with Emily Mallory and Prof. Russ Altman at Stanford, we used DeepDive to extract mentions of gene-gene interactions from the scientific literature,²⁹ and are currently developing DeepDive applications with extraction schemas that include relations between genes, diseases, and drugs in order to predict novel pharmacological relationships.^c

TAC-KBP. TAC-KBP is a NIST-sponsored research competition in which the task is to extract common properties

^b <http://www.cbsnews.com/news/new-search-engine-exposes-the-dark-web/>.

^c <https://www.pharmgkb.org/>.

Figure 2. Example relations extracted from text, tables, and diagrams in the paleontology literature by PaleoDeepDive.

Figure 3. Quality of KBC systems built with DeepDive. On many applications, KBC systems built with DeepDive achieve comparable (and sometimes better) quality than professional human volunteers, and lead to similar scientific insights on topics, such as biodiversity. This quality is achieved by iteratively integrating diverse sources of data—often quality scales with the amount of information we enter into the system.

of people and organizations (e.g., age, birthplace, spouses, and shareholders) from 1.3 million newswire and web documents—this task is also termed *slot filling*. In the 2014 evaluation, 31 US and international teams participated in the competition, including a Stanford team that submitted a solution based on DeepDive.¹ The DeepDive-based solution achieved the highest precision, recall, and F1 of all the submissions.

2.3. Challenges

In all the applications mentioned above, KBC systems built with DeepDive achieved high quality as illustrated in Figure 3. Achieving this high quality level requires that we deal with several challenging aspects of the KBC problem.

Unstructured data complexity. In its full generality, the KBC task encompasses several longstanding grand challenges of computer science, including machine reading and computer vision. Even for simple schemas, extraction of structured information from unstructured sources contains many challenging aspects. For example, consider extracting the relation *Causes* (Gene, Phenotype)—that is, assertions of a genetic mutation causing a certain phenotype (symptom)—from the scientific literature (see Section 2.2). Genes generally have standardized forms of expression (e.g., BRCA1); however, they are easily confused with acronyms for diseases they cause; signals from across the document must be used to resolve these false positives. Phenotypes are even more challenging, because they can be expressed in many synonymous forms (e.g., “headache,” “head pain,” and “pain in forehead”). And extracting pairs that participate in the *Caused* relation encompasses dealing with all the standard challenges of linguistic variation and complexity, as well as application-specific domain terminology.

This challenge becomes even more serious when information comes from different sources that potentially

Figure 4. One challenge of building high-quality KBC systems is exploiting diverse sources of information jointly to extract data accurately. In this example page of a *Paleontology* journal article, identifying the correct location of *Xenacanthus* requires integrating information from within tables, text, and external structured knowledge bases. This problem becomes even more challenging when many extractors are not 100% accurate, motivating the joint probabilistic inference engine inside DeepDive.

need to be considered together—that is, *jointly*—to make a correct extraction. In Figure 4, for example, to reach the extraction that the genus *Xenacanthus* appears in the location of the name *Obora*, the extraction system needs to consult extractions from text, tables, and external structured sources.

Scale. KBC systems need to be able to ingest massive numbers of documents, far outstripping the document counts of even well-funded human curation efforts. For example, Figure 5 illustrates the data flow of PaleoDeepDive. The input to PaleoDeepDive contains nearly 300K journal articles and books, whose total size exceeds 2TB. These raw inputs are then processed with tools such as OCR and linguistic parsing, which are computationally expensive and may take hundreds of thousands of machine hours.^d

Multimodal input. We have found that text is often not enough: often, the data that are interesting to scientists are located in the tables, figures, and images of articles. For example, in geology, more than 50% of the facts that we are interested in are buried in tables.¹⁶ For paleontology, the relationship between taxa, as known as taxonomy, is almost exclusively expressed in section headers.³⁶ For pharmacology, it is not uncommon for a simple diagram to contain a large number of metabolic pathways. Additionally, external sources of information (other knowledge bases) typically contain high-quality signals (e.g., Freebase and Macrostrat) that we would like to leverage and integrate. To build a high-quality KBC system, we need to deal with these diverse modalities of input.^e

^d <http://www.freebase.com/>.

^e <http://macrostrat.org/>.

Figure 5. Another challenge of building high-quality KBC systems is that one usually needs to deal with data at the scale of terabytes. These data are not only processed with traditional relational operations, but also operations involving machine learning and statistical inference. Thus, DeepDive consists of a set of techniques to increase the speed, scale, and incremental execution of inference tasks involving billions of correlated random variables.

3. KBC USING DEEPDIVE

We describe DeepDive, an end-to-end framework for building KBC systems with a declarative language.

3.1. Definitions for KBC systems

The *input* to a KBC system is a heterogeneous collection of unstructured, semistructured, and/or structured data, ranging from text documents to existing but incomplete KBs, and an *application schema* specifying the target relations to extract. The *output* of the system is a relational database containing relations extracted from the input according to the application schema. Creating the knowledge base involves extraction, cleaning, and integration.

EXAMPLE 3.1. Figure 6 illustrates a running example in which our goal is to construct a knowledge base with pairs of individuals who are married to each other. The input to the system is a collection of news articles and an incomplete set of married people; the output is a Knowledge base (KB) containing pairs of people that the input sources assert to be married. A KBC system extracts linguistic patterns, for example, “... and his wife ...” between a pair of mentions of individuals (e.g., Barack Obama and M. Obama); these patterns are then used as features in a classifier deciding whether this pair of mentions indicates that they are married (in the *HasSpouse*) relation.

We adopt standard terminology from KBC, for example, ACE. There are four types of objects that a KBC system seeks to extract from input documents, namely *entities*, *relations*, *mentions*, and *relation mentions*. An *entity* is a real-world person, place, or thing. For example, “Michelle_Obama_1” represents the actual entity for a person whose name is “Michelle Obama”; another individual with the same name

Figure 6. A KBC system takes unstructured documents as input and outputs a structured knowledge base. The runtimes are for the TAC-KBP competition system. To improve quality, the developer adds new rules and new data with error analysis conducted on the result of the current snapshot of the system. DeepDive provides a declarative language to specify each type of different rule and data, and techniques to incrementally execute this iterative process.

would have another number. A *relation* associates two (or more) entities, and represents the fact that there exists a relationship between the participating entities. For example, “Barack_Obama_1” and “Michelle_Obama_1” participate in the *HasSpouse* relation, which indicates that they are married. These real-world entities and relationships are described in text. A *mention* is a span of text in an input document that refers to an entity or relationship: “Michelle” may be a mention of the entity “Michelle_Obama_1.” A *relation mention* is a phrase that connects two mentions that participate in a relation, such as Barack Obama and M. Obama. The process of mapping mentions to entities is called *entity linking*.

3.2. The DeepDive framework^f

DeepDive is an end-to-end framework for building KBC systems. In this section, we walk through each phase. DeepDive supports both SQL and Datalog, but we use datalog syntax for this exposition. The rules we describe in this section are manually created by the user of DeepDive, and the process of creating these rules is application-specific. For simplicity of exposition, we focus on an example with text input in the rest of the section (Figure 7).^g

Candidate mapping and feature extraction. All data in DeepDive—preprocessed input, intermediate data, and final output—is stored in a relational database. The first phase populates the database using a set of SQL queries and user-defined functions (UDFs) that we call *feature extractors*. By default, DeepDive stores all documents in the database in one sentence per row with markup produced by standard

^f <http://www.itl.nist.gov/iad/mig/tests/ace/2000/>.

^g For more information, including examples, please see <http://deepdive.stanford.edu>. Note that our engine is built on Postgres and Greenplum for all SQL processing and UDFs. There is also a port to MySQL.

Figure 7. An example KBC system (see Section 3.2 for details).

<

NLP preprocessing tools, including HTML stripping, part-of-speech tagging, and linguistic parsing. After this loading step, DeepDive executes two types of queries: (1) *candidate mappings*, which are SQL queries that produce possible mentions, entities, and relations, and (2) *feature extractors*, which associate features to candidates, for example, “... and his wife ...” in Example 3.1.

EXAMPLE 3.2. *Candidate mappings are usually simple. Here, we create a relation mention for every pair of candidate persons in the same sentence (s):*

```
(R1) MarriedCandidate(m1,m2):-
    PersonCandidate(s,m1),PersonCandidate(s,m2).
```

Candidate mappings are simply SQL queries with UDFs that look like low-precision but high-recall extract-transform-load (ETL) scripts. Such rules must be high recall: if the union of candidate mappings misses a fact, DeepDive has no chance to extract it.

We also need to extract features, and we extend classical Markov logic¹¹ in two ways: (1) *user-defined functions (UDFs)* and (2) *weight tying*, which we illustrate by example.

EXAMPLE 3.3. *Suppose that phrase(m1, m2, sent) returns the phrase between two mentions in the sentence, for example, “and his wife” in the above example. The phrase between two mentions may indicate whether two people are married. We would write this as:*

```
(FE1) MarriedMentions(m1,m2):-
    MarriedCandidate(m1,m2),Mention(s,m1),
    Mention(s,m2),Sentence(s,sent)
    weight = phrase(m1,m2,sent).
```

One can think about this as a classifier: This rule says that whether the text indicates that the mentions m1 and m2 are married is influenced by the phrase between those mention pairs. The system will infer, based on training data, its confidence (by estimating the weight) that two mentions are indeed indicated to be married.

Technically, phrase returns an identifier that determines which weights should be used for a given relation mention

in a sentence. If phrase returns the same result for two relation mentions, they receive the *same* weight. We explain weight tying in more detail in Section 3.3. In general, phrase could be an arbitrary UDF that operates in a per-tuple fashion. This allows DeepDive to support common feature types ranging from “bag-of-words” to context-aware NLP features to feature sets incorporating domain-specific dictionaries and ontologies. In addition to specifying sets of classifiers, DeepDive inherits Markov Logic’s ability to specify rich correlations between entities via weighted rules. Such rules are particularly helpful for data cleaning and data integration.

Supervision. Just as in Markov Logic, DeepDive can use training data or evidence about any relation; in particular, each user relation is associated with an evidence relation with the same schema and an additional field that indicates whether the entry is true or false. Continuing our example, the evidence relation MarriedMentions_Ev could contain mention pairs with positive and negative labels. Operationally, two standard techniques generate training data: (1) hand-labeling and (2) *distant supervision*, which we illustrate here.

EXAMPLE 3.4. *Distant supervision^{19,30} is a popular technique to create evidence in KBC systems. The idea is to use an incomplete KB of married entity pairs to heuristically label (as True evidence) all relation mentions that link to a pair of married entities:*

```
(S1) MarriedMentions_Ev(m1,m2,true):-
    MarriedCandidates(m1,m2),EL(m1,e1),
    EL(m2,e2),Married(e1,e2).
```

Here, Married is an (incomplete) list of married real-world persons that we wish to extend. The relation EL is for “entity linking” that maps mentions to their candidate entities. At first blush, this rule seems incorrect. However, it generates noisy, imperfect examples of sentences that indicate two people are married. Machine learning techniques are able to exploit redundancy to cope with the noise and learn the relevant phrases (e.g., and his wife). Negative examples are generated by relations that are largely disjoint (e.g., siblings). Similar to DIPRE⁴ and Hearst patterns,¹⁸ distant supervision exploits the “duality”³⁴ between patterns and relation instances; furthermore, it allows us to integrate this idea into DeepDive’s unified probabilistic framework.

Learning and inference. In the learning and inference phase, DeepDive generates a factor graph, similar to Markov Logic, and uses techniques from Tuffy.³³ The inference and learning are done using standard techniques (Gibbs sampling) that we describe below after introducing the formal semantics.

Error analysis. DeepDive runs the above three phases in sequence, and at the end of the learning and inference, it obtains a marginal probability p for each candidate fact. To produce the final KB, the user selects facts that DeepDive predicts are true with probability above some user-selected threshold, for example, $p > 0.95$. Typically, the user needs to inspect errors and repeat the previous steps, a process that we call *error analysis*. Error analysis is the process of understanding the most common mistakes (incorrect extractions, overly specific features, candidate mistakes, etc.) and deciding how to correct them.³⁹ To facilitate error analysis, users write standard SQL queries.

3.3. Discussion of design choices

We have found the following key aspects of the DeepDive approach that we believe enable noncomputer scientists to build sophisticated KBC systems: (1) There is no reference in a DeepDive program to the underlying machine learning algorithms. Thus, DeepDive programs are declarative in a strong sense. Probabilistic semantics provide a way to debug the system independent of the algorithm it uses. (2) DeepDive allows users to write feature extraction code (UDFs) in familiar languages (Python, SQL, and Scala). (3) By using and producing relational databases, DeepDive fits into the familiar SQL stack, which allows standard tools to inspect and visualize the data. (4) The user constructs an end-to-end system and then refines the quality of the system in a pay-as-you-go way.²⁸ In contrast, traditional pipeline-based ETL scripts may lead to a user's time and effort being overspent on a specific extraction or integration step—without the ability to evaluate how important each step is for the quality of the end result. Anecdotally, pay-as-you-go leads to more informed decisions about how to improve quality.

The above design choices necessitated overcoming several technical challenges, two of which we briefly highlight below.

Joint statistical inference. In many systems, successive stages are simply pipelined together, propagating errors from one stage to the next, and complicating iterative development efforts. This can also have noticeable performance effects when the information from different sources are all *noisy*, and potentially need to be considered together—that is, *jointly*—to make a correct extraction. To join extractions with different confidence levels together, one needs a principled framework. The DeepDive approach to this challenge is based on a Bayesian probabilistic approach. DeepDive treats all these information sources as one joint probabilistic inference problem, with all predictions modeled as random variables within a factor graph model. This probabilistic framework ensures that all facts produced by DeepDive are associated with a marginal probability. These marginal probabilities are meaningful in DeepDive; that is, they represent the empirical accuracy that one should expect for the extracted

mentions, and provide a guideline to the developer for improving the KBC system built using DeepDive.

Incremental and efficient execution. Especially with the above design choices, performance is a major challenge. In our KBC systems using DeepDive, we may need to perform inference and learning on a large number of highly correlated random variables. For example, in PaleoDeepDive, we construct factor graphs that contain more than 300 million variables, each representing a potential mention to extract as final output. Therefore, one of our technical focus areas has been to speed up probabilistic inference.^{32, 33, 35, 50, 51} A second major technical focus has been the development of efficient *incremental* methods for grounding and inference, given the iterative nature of KBC application development. In Section 4, we briefly describe these techniques and provide pointers to readers who are interested in further details.

4. TECHNIQUES

A DeepDive program is a set of rules with weights specified using the language we described above. During inference, the values of all weights are assumed to be known, whereas in learning, one finds the set of weights that maximizes the probability of the evidence. The execution of a DeepDive program consists of two phases: (i) grounding and (ii) statistical inference and learning. In this section, we briefly describe the techniques we developed in each phase to make DeepDive performant and scalable.

4.1. Grounding^h

As shown in Figure 8, DeepDive explicitly constructs a factor graph for inference and learning using a set of SQL queries. A factor graph is a triple $(V, F, \hat{w})$ in which V is a set of nodes that correspond to Boolean random variables, F is a set of hyperedges (for $f \in F, f \subseteq V$), and $\hat{w}: F \times \{0, 1\}^V \rightarrow \mathbb{R}$ is a weight function. In DeepDive, V and F are explicitly created using a set of SQL queries, and this process is called *grounding*.

^h There is a justification for probabilistic reasoning, as Cox's theorem asserts (roughly) that if one uses numbers as degrees of belief, then one must either use probabilistic reasoning or risk contradictions in one's reasoning system; that is, a probabilistic framework is the only sound system for reasoning in this manner. We refer the reader to Jaynes et al.²¹

Figure 8. Schematic illustration of grounding. Each tuple corresponds to a Boolean random variable and node in the factor graph. We create one factor for every set of groundings of an inference rule.

EXAMPLE 4.1. Take the database instances and rules in Figure 8 as an example: each tuple in relation R , S , and Q is a random variable, and V contains all random variables. The inference rules $F1$ and $F2$ ground factors with the same name in the factor graph, as illustrated in Figure 8. Both $F1$ and $F2$ are implemented as SQL statements in DeepDive.

Incremental grounding. Because DeepDive is based on SQL, we are able to take advantage of decades of work on incremental view maintenance. The input to this phase is the same as the input to the grounding phase: a set of SQL queries and the user schema. The output of this phase is how the output of grounding changes, that is, a set of modified variables ΔV and their factors ΔF . Since V and F are simply views over the database, any view maintenance techniques can be applied to incremental grounding. DeepDive uses the DRED algorithm,¹⁷ which handles both additions and deletions. Recall that in DRED, for each relation R_i in the user's schema, we create a *delta relation*, R_i^δ , with the same schema as R_i and an additional column count. For each tuple t , $t.count$ represents the number of derivations of t in R_i . On an update, DeepDive updates delta relations in two steps. First, for tuples in R_i^δ , DeepDive directly updates the corresponding counts. Second, an SQL query called a *delta rule* is executed, which processes these counts to generate modified variables ΔV and factors ΔF . We found that the overhead of DRED is modest and the gains may be substantial, so DeepDive always runs DRED—except on initial load.

4.2. Statistical inference and learning¹

The main task that DeepDive conducts on factor graphs is statistical inference, that is, determining for a given node what the marginal probability is that this node takes the value 1, that it is a correct output tuple that should be included in the final knowledge base. In general, computing these marginal probabilities is $\#P$ -hard.⁴⁵ Like many other systems, DeepDive uses Gibbs sampling⁴⁰ to estimate the marginal probability of every tuple in the database.

Efficiency and scalability. There are two components to scaling statistical algorithms: *statistical efficiency*, roughly how many steps an algorithm takes to converge, and *hardware efficiency*, how efficient each of those steps is. We introduced this terminology and studied this extensively in a recent paper.⁵¹

DimmWitted, the statistical inference and learning engine in DeepDive,⁵¹ is built upon our research on how to design a high-performance statistical inference and learning engine on a single machine.^{27, 32, 50, 51} DimmWitted models Gibbs sampling as a “column-to-row access” operation: each row corresponds to one factor, each column to one variable, and the nonzero elements in the data matrix correspond to edges in the factor graph. To process one variable, DimmWitted fetches one column of the matrix to get the set of factors, and other columns to get the set of variables that connect to the same factor. On standard benchmarks, DimmWitted was

3.7× faster than GraphLab's implementation without any application-specific optimization. Compared with traditional work, the main novelty of DimmWitted is that it considers *both* hardware efficiency and statistical efficiency for executing an inference and learning task.

- **Hardware efficiency.** DeepDive takes into consideration the architecture of modern nonuniform memory access (NUMA) machines. A NUMA machine usually contains multiple nodes (sockets), where each socket contains multiple CPU cores. To achieve higher hardware efficiency, one wants to decrease the communication across different NUMA nodes.
- **Statistical efficiency.** Pushing hardware efficiency to the extreme might decrease statistical efficiency, because the lack of communication between nodes might decrease the rate of convergence of a statistical inference and learning algorithm. DeepDive takes advantage of the theoretical results of model averaging⁵³ and our own results about lock-free execution.^{27, 32}

On the whole corpus of *Paleobiology*, the factor graph contains more than 0.2 billion random variables and 0.3 billion factors. On this factor graph, DeepDive is able to run Gibbs sampling on a machine with four sockets (10 cores per socket), and we find that we can generate 1000 samples for all 0.2 billion random variables in 28 min. This is more than 4× faster than a non-NUMA-aware implementation.

Incremental inference. Due to our choice of incremental grounding, the input to DeepDive's inference phase is a factor graph along with a set of changed variables and factors. The goal is to compute the output probabilities computed by the system. Our approach is to frame the incremental maintenance problem as approximate inference. Previous work in the database community has looked at how machine learning data products change in response to both new labels²⁴ and new data.^{7, 8} In KBC, both the program and data change on each iteration. Our proposed approach can cope with both types of change simultaneously.

The technical question is which approximate inference algorithms to use in KBC applications. We choose to study two popular classes of approximate inference techniques: *sampling-based materialization* (inspired by sampling-based probabilistic databases such as MCDB²⁰) and *variational-based materialization* (inspired by techniques for approximating graphical models⁴⁴). Applying these techniques to incremental maintenance for KBC is novel, and it is not theoretically clear how the techniques compare. Thus, we conducted an experimental evaluation of these two approaches on a diverse set of DeepDive programs. We found these two approaches to be sensitive to changes along three largely orthogonal axes: the size of the factor graph, the sparsity of correlations, and the anticipated number of future changes. The performance varies by up to two orders of magnitude in different points of the space. Our study of the tradeoff space highlights that neither materialization strategy dominates the other. To automatically choose the materialization strategy, we developed a simple rule-based optimizer.⁴²

¹ For example, for the grounding procedure illustrated in Figure 8, the delta rule for $F1$ is $q(x): \neg R(x, y)$.

5. RELATED WORK

KBC has been an area of intense studies over the last decade.^{2, 3, 6, 14, 23, 25, 31, 37, 41, 43, 48, 52} Within this space, there are a number of approaches.

5.1. Rule-based systems

The earliest KBC systems used pattern matching to extract relationships from text. The most well-known example is the “Hearst Pattern” proposed by Hearst¹⁸ in 1992. In her seminal work, Hearst observed that a large number of hyponyms can be discovered by simple patterns, for example, “X such as Y.” Hearst’s technique has formed the basis of many further techniques that attempt to extract high-quality patterns from text. Rule-based (pattern matching-based) KBC systems, such as IBM’s SystemT,^{25, 26} have been built to aid developers in constructing high-quality patterns. These systems provide the user with a (declarative) interface to specify a set of rules and patterns to derive relationships. These systems have achieved state-of-the-art quality on tasks, such as parsing.²⁶

5.2. Statistical approaches

One limitation of rule-based systems is that the developer needs to ensure that all rules provided to the system are high-precision rules. For the last decade, probabilistic (or machine learning) approaches have been proposed to allow the system to select from a range of a priori features automatically. In these approaches, the extracted tuple is associated with a marginal probability that it is true. DeepDive, Google’s knowledge graph, and IBM’s Watson are built on this approach. Within this space, there are three styles of systems based on classification,^{2, 3, 6, 14, 48} maximum a posteriori,^{23, 31, 43} and probabilistic graphical models.^{11, 37, 52} Our work on DeepDive is based on graphical models.

6. CURRENT DIRECTIONS

6.1. Data programming

In a standard DeepDive KBC application (e.g., as in Section 3.2), the weights of the factor graph that models the extraction task are learned using either hand-labeled training data or distant supervision. However, in many applications, assembling hand-labeled training data is prohibitively expensive (e.g., when domain expertise is required), and distant supervision can be insufficient or time consuming to implement perfectly. For example, users may come up with many potential distant supervision rules that overlap, conflict, and are of varying unknown quality, and deciding which rules to include and how to resolve their overlaps could take many development cycles. In a new approach called *data programming*,³⁸ we allow users to specify arbitrary *labeling functions*, which subsume distant supervision rules and allow users to programatically generate training data with increased flexibility. We then learn the relative accuracies of these labeling functions and denoise their labels using automated techniques, resulting in improved performance on the KBC applications outlined.

6.2. Lightweight extraction

In some cases, users may have simple extraction tasks which need to be implemented rapidly, or may wish to first iterate on a simpler initial version of a more complex extraction task.

For example, a user might have a complex extraction task involving multiple entity and relation types, connected by a variety of inference rules, over a large web-scale dataset; but they may want to start by iterating on just a single relationship over a subset of the data. For these cases, we are developing a lightweight, Jupyter notebook-based extraction system called Snorkel, intended for quick iterative development of simple extraction models using data programming.¹³ We envision Snorkel as a companion and complement to DeepDive.^j

6.3. Asynchronous inference

One method for speeding up the inference and learning stages of DeepDive is to execute them asynchronously. In recent work, we observed that asynchrony can introduce bias in Gibbs sampling, and outline some sufficient conditions under which the bias is negligible.¹⁰ Further theoretical and applied work in this direction will allow for faster execution of complex DeepDive models asynchronously.

Acknowledgments

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) XDATA program under no. FA8750-12-2-0335 and DEFT program under no. FA8750-13-2-0039, DARPA’s MEMEX program and SIMPLEX program, the National Science Foundation (NSF) CAREER Award under no. IIS-1353606, the Office of Naval Research (ONR) under awards nos. N000141210041 and N000141310129, the National Institutes of Health Grant U54EB020405 awarded by the National Institute of Biomedical Imaging and Bioengineering (NIBIB) through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative, the Sloan Research Fellowship, the Moore Foundation, American Family Insurance, Google, and Toshiba. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, AFRL, NSF, ONR, NIH, or the US government.

^j snorkel.stanford.edu.

References

1. Angeli, G. et al. Stanford’s 2014 slot filling systems. *TAC KBP* (2014).
2. Banko, M. et al. Open information extraction from the Web. In *IJCAI* (2007).
3. Betteridge, J., Carlson, A., Hong, S.A., Hruschka, E.R., Jr, Law, E.L., Mitchell, T.M., Wang, S.H. Toward never ending language learning. In *AAAI Spring Symposium* (2009).
4. Brin, S. Extracting patterns and relations from the world wide web. In *WebDB* (1999).
5. Brown, E. et al. Tools and methods for building Watson. *IBM Research Report* (2013).
6. Carlson, A. et al. Toward an architecture for never-ending language learning. In *AAAI* (2010).
7. Chen, F., Doan, A., Yang, J., Ramakrishnan, R. Efficient information extraction over evolving text data. In *ICDE* (2008).
8. Chen, F. et al. Optimizing statistical information extraction programs over evolving text. In *ICDE* (2012).
9. Chen, Y., Wang, D.Z. Knowledge expansion over probabilistic knowledge bases. In *SIGMOD* (2014).
10. De Sa, C., Olukotun, K., Ré, C. Ensuring rapid mixing and low bias for asynchronous gibbs sampling. *arXiv preprint arXiv:1602.07415* (2016).
11. Domingos, P., Lowd, D. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, 2009.
12. Dong, X.L. et al. From data fusion to knowledge fusion. In *Vldb* (2014).
13. Ehrenberg, H.R., Shin, J., Ratner, A.J., Fries, J.A., Ré, C. Data programming with DDLite: Putting humans in a different part of the loop. In *HILDA’16 SIGMOD* (2016), 13.
14. Etzioni, O. et al. Web-scale information extraction in KnowItAll: Preliminary results. In *WWW* (2004).
15. Ferrucci, D. et al. Building Watson: An overview of the DeepQA project. *AI Magazine* (2010).

16. Govindaraju, V. et al. Understanding tables in context using standard NLP toolkits. In *ACL* (2013).
17. Gupta, A., Mumick, I.S., Subrahmanian, V.S. Maintaining views incrementally. *SIGMOD Rec.* (1993).
18. Hearst, M.A. Automatic acquisition of hyponyms from large text corpora. In *COLING* (1992).
19. Hoffmann, R. et al. Knowledge-based weak supervision for information extraction of overlapping relations. In *ACL* (2011).
20. Jampani, R. et al. MCDB: A Monte Carlo approach to managing uncertain data. In *SIGMOD* (2008).
21. Jaynes, E.T. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.
22. Jiang, S. et al. Learning to refine an automatically extracted knowledge base using Markov logic. In *ICDM* (2012).
23. Kasneci, G. et al. The YAGO-NAGA approach to knowledge discovery. *SIGMOD Rec.* (2009).
24. Koc, M.L., Ré, C. Incrementally maintaining classification using an RDBMS. *PVLDB* (2011).
25. Krishnamurthy, R. et al. SystemT: A system for declarative information extraction. *SIGMOD Rec.* (2009).
26. Li, Y., Reiss, F.R., Chiticariu, L. SystemT: A declarative information extraction system. In *HLT* (2011).
27. Liu, J. and et al. An asynchronous parallel stochastic coordinate descent algorithm. *ICML* (2014).
28. Madhavan, J. et al. Web-scale data integration: You can only afford to pay as you go. In *CIDR* (2007).
29. Mallory, E.K. et al. Large-scale extraction of gene interactions from full text literature using deepdive. *Bioinformatics* (2015).
30. Mintz, M. et al. Distant supervision for relation extraction without labeled data. In *ACL* (2009).
31. Nakashole, N. et al. Scalable knowledge harvesting with high precision and high recall. In *WSDM* (2011).
32. Niu, F. et al. Hogwild! A lock-free approach to parallelizing stochastic gradient descent. In *NIPS* (2011).
33. Niu, F. et al. Tuffy: Scaling up statistical inference in Markov logic networks using an RDBMS. *PVLDB* (2011).
34. Niu, F. et al. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *Int. J. Semantic Web Inf. Syst.* (2012).
35. Niu, F. et al. Scaling inference for Markov logic via dual decomposition. In *ICDM* (2012).
36. Peters, S.E. et al. A machine reading system for assembling synthetic Paleontological databases. *PLoS One* (2014).
37. Poon, H., Domingos, P. Joint inference in information extraction. In *AAAI* (2007).
38. Ratner, A., De Sa, C., Wu, S., Selsam, D., Ré, C. Data programming: Creating large training sets, quickly. *arXiv preprint arXiv:1605.07723* (2016).
39. Ré, C. et al. Feature engineering for knowledge base construction. *IEEE Data Eng. Bull.* (2014).
40. Robert, C.P., Casella, G. *Monte Carlo Statistical Methods*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
41. Shen, W. et al. Declarative information extraction using datalog with embedded extraction predicates. In *VLDDB* (2007).
42. Shin, J. et al. Incremental knowledge base construction using deepdive. *PVLDB* (2015).
43. Suchanek, F.M. et al. SOFIE: A self-organizing framework for information extraction. In *WWW* (2009).
44. Wainwright, M., Jordan, M. Log-determinant relaxation for approximate inference in discrete Markov random fields. *Trans. Sig. Proc.* (2006).
45. Wainwright, M.J., Jordan, M.I. Graphical models, exponential families, and variational inference. *FTML* (2008).
46. Weikum, G., Theobald, M. From information to knowledge: Harvesting entities and relationships from web sources. In *PODS* (2010).
47. Wick, M. et al. Scalable probabilistic databases with factor graphs and MCMC. *PVLDB* (2010).
48. Yates, A. et al. TextRunner: Open information extraction on the Web. In *NAACL* (2007).
49. Zhang, C. et al. GeoDeepDive: Statistical inference using familiar data-processing languages. In *SIGMOD* (2013).
50. Zhang, C., Ré, C. Towards high-throughput Gibbs sampling at scale: A study across storage managers. In *SIGMOD* (2013).
51. Zhang, C., Ré, C. DimmWitted: A study of main-memory statistical analytics. *PVLDB* (2014).
52. Zhu, J. et al. StatSnowball: A statistical approach to extracting entity relationships. In *WWW* (2009).
53. Zinkevich, M. et al. Parallelized stochastic gradient descent. In *NIPS* (2010), 2595–2603.

Michael Cafarella and JaeHo Shin
 ([michael.cafarella, jae.ho.shin]
 @lattice.io), Lattice Data, Inc., Palo
 Alto, CA.

**Christopher De Sa, Alex Ratner,
 Christopher Ré, Feiran Wang,**

and Sen Wu ([chrismre, cdesa, ajratner,
 feiran, senwu]@cs.stanford.edu),
 Computer Science Department, Stanford
 University, Stanford, CA.

Ce Zhang (ce.zhang@inf.ethz.ch), ETH
 Zurich, Zurich, Switzerland.

© 2017 ACM 0001-0782/17/5 \$15.00

World-Renowned Journals from ACM

ACM publishes over 50 magazines and journals that cover an array of established as well as emerging areas of the computing field. IT professionals worldwide depend on ACM's publications to keep them abreast of the latest technological developments and industry news in a timely, comprehensive manner of the highest quality and integrity. For a complete listing of ACM's leading magazines & journals, including our renowned Transaction Series, please visit the ACM publications homepage: www.acm.org/pubs.

ACM Transactions on Interactive Intelligent Systems

ACM Transactions on Interactive Intelligent Systems (TIIS). This quarterly journal publishes papers on research encompassing the design, realization, or evaluation of interactive systems incorporating some form of machine intelligence.

ACM Transactions on Computation Theory

ACM Transactions on Computation Theory (ToCT). This quarterly peer-reviewed journal has an emphasis on computational complexity, foundations of cryptography and other computation-based topics in theoretical computer science.

PLEASE CONTACT ACM MEMBER
 SERVICES TO PLACE AN ORDER
 Phone: 1.800.342.6626 (U.S. and Canada)
 +1.212.626.0500 (Global)
 Fax: +1.212.944.1318
 (Hours: 8:30am–4:30pm, Eastern Time)
 Email: acmhelp@acm.org
 Mail: ACM Member Services
 General Post Office
 PO Box 30777
 New York, NY 10087-0777 USA

Association for
 Computing Machinery

Advancing Computing as a Science & Profession

www.acm.org/pubs