Incremental Knowledge Base Construction Using DeepDive

Jaeho Shin[†] Sen Wu[†] Feiran Wang[†] Christopher De Sa[†] Ce Zhang[†‡] Christopher Ré[†]
[†]Stanford University
[‡]University of Wisconsin-Madison
{jaeho, senwu, feiran, cdesa, czhang, chrismre}@cs.stanford.edu

ABSTRACT

Populating a database with unstructured information is a long-standing problem in industry and research that encompasses problems of extraction, cleaning, and integration. Recent names used for this problem include dealing with dark data and knowledge base construction (KBC). In this work, we describe DeepDive, a system that combines database and machine learning ideas to help develop KBC systems, and we present techniques to make the KBC process more efficient. We observe that the KBC process is iterative, and we develop techniques to incrementally produce inference results for KBC systems. We propose two methods for incremental inference, based respectively on sampling and variational techniques. We also study the tradeoff space of these methods and develop a simple rule-based optimizer. DeepDive includes all of these contributions, and we evaluate DeepDive on five KBC systems, showing that it can speed up KBC inference tasks by up to two orders of magnitude with negligible impact on quality.

1. INTRODUCTION

The process of populating a structured relational database from unstructured sources has received renewed interest in the database community through high-profile start-up companies (e.g., Tamr and Trifacta), established companies like IBM's Watson [7, 16], and a variety of research efforts [11, 25, 28, 36, 40]. At the same time, communities such as natural language processing and machine learning are attacking similar problems under the name *knowledge base construction* (KBC) [5, 14, 23]. While different communities place differing emphasis on the extraction, cleaning, and integration phases, all communities seem to be converging toward a common set of techniques that include a mix of data processing, machine learning, and engineers-in-the-loop.

The ultimate goal of KBC is to obtain high-quality structured data from unstructured information. These databases are richly structured with tens of different entity types in complex relationships. Typically, quality is assessed using two complementary measures: precision (how often a claimed tuple is correct) and recall (of the possible tuples to extract, how many are actually extracted). These systems can ingest massive numbers of documents–far outstripping the document counts of even well-funded human curation efforts. Industrially, KBC systems are constructed by skilled engineers in a months-long (or longer) process–not a one-shot algorithmic task. Arguably, the most important question in such systems is how to best use skilled engineers' time to rapidly improve data quality. In its full generality, this question spans a number of areas in computer science, including programming languages, systems, and HCI. We focus on a narrower question, with the axiom that *the more rapid the programmer moves through the KBC construction loop, the more quickly she obtains high-quality data.*

This paper presents DeepDive, our open-source engine for knowledge base construction.[1] DeepDive's language and execution model are similar to other KBC systems: DeepDive uses a high-level declarative language [11, 28, 30]. From a database perspective, DeepDive's language is based on SQL. From a machine learning perspective, DeepDive's language is based on Markov Logic [13, 30]: DeepDive's language inherits Markov Logic Networks' (MLN's) formal semantics.[2] Moreover, it uses a standard execution model for such systems [11, 28, 30] in which programs go through two main phases: *grounding*, in which one evaluates a sequence of SQL queries to produce a data structure called a *factor graph* that describes a set of random variables and how they are correlated. Essentially, every tuple in the database or result of a query is a random variable (node) in this factor graph. The *inference* phase takes the factor graph from grounding and performs statistical inference using standard techniques, e.g., Gibbs sampling [42, 44]. The output of inference is the marginal probability of every tuple in the database. As with Google's Knowledge Vault [14] and others [31], DeepDive also produces marginal probabilities that are *calibrated*: if one examined all facts with probability 0.9, we would expect that approximately 90% of these facts would 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).

Proceedings of the VLDB Endowment, Vol. 8, No. 11

[1] `http://deepdive.stanford.edu`
[2] DeepDive has some technical differences from Markov Logic that we have found useful in building applications. We discuss these differences in Section 2.3.

In our experience with DeepDive, we found that KBC is an iterative process. 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 law enforcement agencies via DARPA's MEMEX, and scientists in fields such as paleobiology, drug repurposing, and genomics. Recently, we compared a DeepDive system's extractions to the quality of extractions provided by human volunteers over the last ten 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 [32]. Additionally, the winning entry of the 2014 TAC-KBC competition was built on DeepDive [3]. In all cases, we have seen the process of developing KBC systems is iterative: quality requirements change, new data sources arrive, and new concepts are needed in the application. This led us to develop techniques to make the entire pipeline incremental in the face of changes both to the data and to the DeepDive program. Our primary technical contributions are to make the grounding and inference phases more incremental.[3]

Incremental Grounding. Grounding and feature extraction are performed by a series of SQL queries. To make this phase incremental, we adapt the algorithm of Gupta, Mumick, and Subrahmanian [18]. In particular, DeepDive allows one to specify "delta rules" that describe how the output will change as a result of changes to the input. Although straightforward, this optimization has not been applied systematically in such systems and can yield up to 360× speedup in KBC systems.

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 data and rules. The goal is to compute the output probabilities computed by the system. Our approach is to frame the incremental maintenance problem as one of approximate inference. Previous work in the database community has looked at how machine learning data products change in response to both to new labels [24] and to new data [9,10]. 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 MCDB [21]) and *variational-based materialization* (inspired by techniques for approximating graphical models [38]). 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 are 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 develop a simple rule-based optimizer.

[3]As incremental learning uses standard techniques, we cover it only in the full version of this paper.

Experimental Evaluation Highlights. We used DeepDive programs developed by our group and DeepDive users to understand whether the improvements we describe can speed up the iterative development process of DeepDive programs. To understand the extent to which DeepDive's techniques improve development time, we took a sequence of six snapshots of a KBC system and ran them with our incremental techniques and completely from scratch. In these snapshots, our incremental techniques are 22× faster. The results for each snapshot differ at most by 1% for high-quality facts (90%+ accuracy); fewer than 4% of facts differ by more than 0.05 in probability between approaches. Thus, essentially the same facts were given to the developer throughout execution using the two techniques, but the incremental techniques delivered them more quickly.

Outline. The rest of the paper is organized as follows. Section 2 contains an in-depth analysis of the KBC development process, and the presentation of our language for modeling KBC systems. We discuss the different techniques for incremental maintenance in Section 3. We also present the results of the exploration of the tradeoff space and the description of our optimizer. Our experimental evaluation is presented in Section 4.

Related Work

Knowledge Base Construction (KBC) KBC has been an area of intense study over the last decade, moving from pattern matching [19] and rule-based systems [25] to systems that use machine learning for KBC [5, 8, 14, 15, 28]. Many groups have studied how to improve the quality of specific components of KBC systems [27, 43]. We build on this line of work. We formalized the development process and built DeepDive to ease and accelerate the KBC process, which we hope is of interest to many of these systems as well. DeepDive has many common features to Chen and Wang [11], Google's Knowledge Vault [14], and a forerunner of DeepDive, Tuffy [30]. We focus on the incremental evaluation from feature extraction to inference.

Declarative Information Extraction The database community has proposed declarative languages for information extraction, a task with similar goals to knowledge base construction, by extending relational operations [17, 25, 36], or rule-based approaches [28]. These approaches can take advantage of classic view maintenance techniques to make the execution incremental, but they do not study how to incrementally maintain the result of statistical inference and learning, which is the focus of our work.

Incremental Maintenance of Statistical Inference and Learning Related work has focused on incremental inference for specific classes of graphs (tree-structured [12] or low-degree [1] graphical models). We deal instead with the class of factor graphs that arise from the KBC process, which is much more general than the ones examined in previous approaches. Nath and Domingos [29] studied how to extend belief propagation on factor graphs with new evidence,

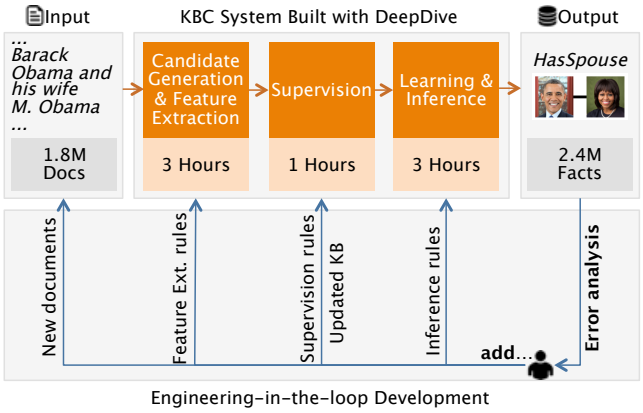


Figure 1: A KBC system takes as input unstructured documents and outputs a structured knowledge base. The runtimes are for the TAC-KBP competition system (News). To improve quality, the developer adds new rules and new data.

but without any modification to the structure of the graph. Wick and McCallum [41] proposed a "query-aware MCMC" method. They designed a proposal scheme so that query variables tend to be sampled more frequently than other variables. We frame our problem as approximate inference, which allows us to handle changes to the program and the data in a single approach.

2. KBC USING DEEPDIVE

We describe DeepDive, an end-to-end framework for building KBC systems with a declarative language. We first recall standard definitions, and then introduce the essentials of the framework by example, compare our framework with Markov Logic, and describe DeepDive's formal semantics.

2.1 Definitions for KBC Systems

The *input* to a KBC system is a heterogeneous collection of unstructured, semi-structured, and structured data, ranging from text documents to existing but incomplete KBs. The *output* of the system is a relational database containing facts extracted from the input and put into the appropriate schema. Creating the knowledge base may involve extraction, cleaning, and integration.

EXAMPLE 2.1. *Figure 1 illustrates our running example: a knowledge base with pairs of individuals that are married to each other. The input to the system is a collection of news articles and an incomplete set of married persons; the output is a KB containing pairs of person that are married. A KBC system extracts linguistic patterns, e.g., "... and his wife ..." between a pair of mentions of individuals (e.g., "Barack Obama" and "M. Obama"). Roughly, 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, e.g., ACE.[4] 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 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, M. Obama)". The process of mapping mentions to entities is called *entity linking*.

2.2 The DeepDive Framework

DeepDive is an end-to-end framework for building KBC systems, as shown in Figure 1.[5] We walk through each phase. DeepDive supports both SQL and datalog, but we use datalog syntax for 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.

Candidate Generation and Feature Extraction. All data in DeepDive 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 NLP pre-processing 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* that associate features to candidates, e.g., "... and his wife ..." in Example 2.1.

EXAMPLE 2.2. *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 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 Logic in two ways: (1) user-defined functions and (2) weight tying, which we illustrate by example.

EXAMPLE 2.3. *Suppose that* phrase(m1, m2, sent) *returns the phrase between two mentions in the sentence, e.g., "and his wife" in the above example. The phrase between two*

[4] http://www.itl.nist.gov/iad/mig/tests/ace/2000/

[5] 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.

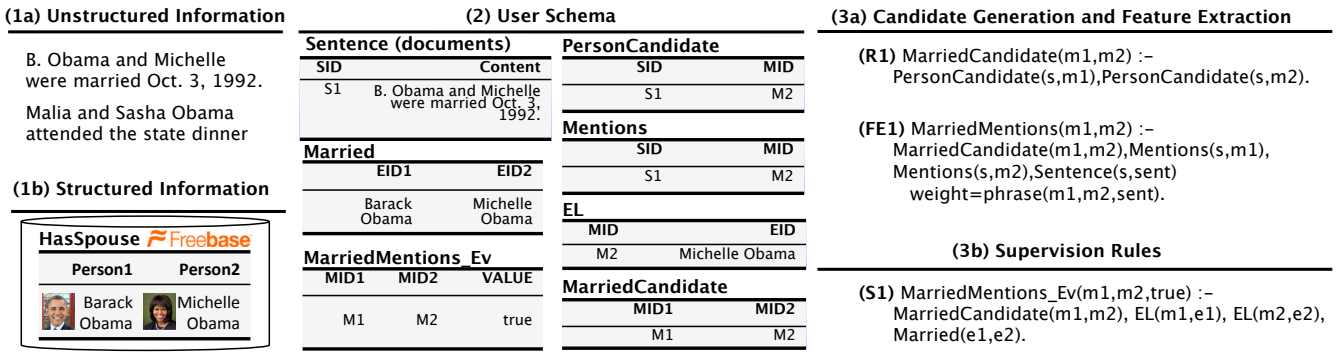


Figure 2: An example KBC system. See Section 2.2 for details.

mentions may indicate whether two people are married. We would write this as:

$$\textit{(FE1)}\ \texttt{MarriedMentions}(\mathrm{m1},\mathrm{m2}) :- \texttt{MarriedCandidate}(\mathrm{m1},\mathrm{m2}), \texttt{Mention}(\mathrm{s},\mathrm{m1}), \texttt{Mention}(\mathrm{s},\mathrm{m2}), \texttt{Sentence}(\mathrm{s},\mathrm{sent}) \quad \mathrm{weight} = \mathsf{phrase}(\mathrm{m1},\mathrm{m2},\mathrm{sent}).$$

One can think about this like 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 2.3. In general, phrase could be an arbitrary UDF that operates in a per-tuple fashion. This allows DeepDive to support common examples of features such as "bag-of-words" to context-aware NLP features to highly 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 below.

Example 2.4. Distant supervision *[20, 27] 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:*

$$\textit{(S1)}\ \texttt{MarriedMentions\_Ev}(\mathrm{m1},\mathrm{m2},\mathrm{true}) :- \texttt{MarriedCandidates}(\mathrm{m1},\mathrm{m2}), \texttt{EL}(\mathrm{m1},\mathrm{e1}), \texttt{EL}(\mathrm{m2},\mathrm{e2}), \texttt{Married}(\mathrm{e1},\mathrm{e2}).$$

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 DIPRE [6] and Hearst patterns [19], distant supervision exploits the "duality" [6] 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 [30]. 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 often selects facts in which we are highly confident, e.g., $\mathrm{p} > 0.95$. Typically, the user needs to inspect errors and repeat, a process that we call *error analysis*. Error analysis is the process of understanding the most common mistakes (incorrect extractions, too-specific features, candidate mistakes, etc.) and deciding how to correct them [34]. To facilitate error analysis, users write standard SQL queries.

2.3 Discussion of Design Choices

We have found three related aspects of the DeepDive approach that we believe enable non-computer scientists to write DeepDive programs: (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 independently of any algorithm. (2) DeepDive allows users to write feature extraction code in familiar languages (Python, SQL, and Scala). (3) DeepDive fits into the familiar SQL stack, which allows standard tools to inspect and visualize the data. A second key property is that the user constructs an end-to-end system and then refines the quality of the system in a pay-as-you-go way [26]. In contrast, traditional pipeline-based ETL scripts may lead to time and effort spent on extraction and integration–without the ability to evaluate how important each step is for end-to-end application quality. Anecdotally, pay-as-you-go leads to more informed decisions about how to improve quality.

Comparison with Markov Logic. Our language is based on Markov Logic [13, 30], and our current language inherits Markov Logic's formal semantics. However, there are three differences in how we implement DeepDive's language:

Weight Tying. As shown in rule FE1, DeepDive allows factors to share weights across rules, which is used in every DeepDive system. As we will see declaring a classifier is a one-liner in DeepDive: $\texttt{Class}(x) :\text{-}\, R(x, f)$ with $\texttt{weight} = w(f)$ declares a classifier for objects (bindings of x); $R(x, f)$ indicates that object x has features f. In standard MLNs, this would require one rule for each feature.[6] In MLNs, every rule introduces a single weight, and the correlation structure and weight structure are coupled. DeepDive decouples them, which makes writing some applications easier.

User-defined Functions. As shown in rule FE1, DeepDive allows the user to use user-defined functions (phrase in FE1) to specify feature extraction rules. This allows DeepDive to handle common feature extraction idioms using regular expressions, Python scripts, etc. This brings more of the KBC pipeline into DeepDive, which allows DeepDive to find optimization opportunities for a larger fraction of this pipeline.

Implication Semantics. In the next section, we introduce a function g that counts the number of groundings in different ways. g is an example of *transformation groups* [22, Ch. 12], a technique from the Bayesian inference literature to model different noise distributions. Experimentally, we show that different semantics (choices of g) affect the quality of KBC applications (up to 10% in F1 score) compared with the default semantics of MLNs. After some notation, we give an example to illustrate how g alters the semantics.

2.4 Semantics of a DeepDive Program

A DeepDive program is a set of rules with weights. During inference, the values of all weights w are assumed to be known, while, in learning, one finds the set of weights that maximizes the probability of the evidence. As shown in Figure 3, a DeepDive program defines a standard structure called a *factor graph* [39]. First, we directly define the probability distribution for rules that involve weights, as it may help clarify our motivation. Then, we describe the corresponding factor graph on which inference takes place.

Each possible tuple in the user schema–both IDB and EDB predicates–defines a Boolean random variable (r.v.). Let $\mathbb{V}$ be the set of these r.v.'s. Some of the r.v.'s are fixed

[6]Our system Tuffy introduced this feature to MLNs, but its semantics had not been described in the literature.

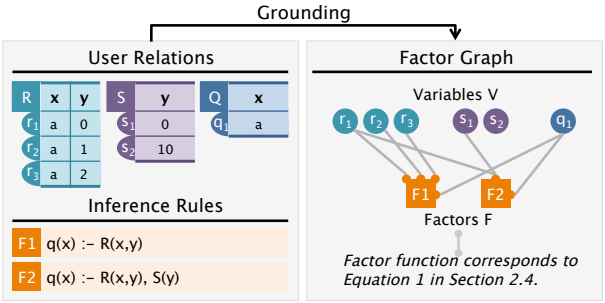


Figure 3: Schematic illustration of grounding. Each tuple corresponds to a Boolean random variable and node in the factor graph. We create one factor for every set of groundings.

Semantics	$g(n)$
Linear	n
Ratio	$\log(1+n)$
Logical	$\mathbb{1}_{\{n>0\}}$

Figure 4: Semantics for g in Equation 1.

to a specific value, e.g., as specified in a supervision rule or by training data. Thus, $\mathbb{V}$ has two parts: a set $\mathcal{E}$ of *evidence* variables (those fixed to a specific values) and a set $\mathcal{Q}$ of *query* variables whose value the system will infer. The class of evidence variables is further split into *positive evidence* and *negative evidence*. We denote the set of positive evidence variables as $\mathcal{P}$, and the set of negative evidence variables as $\mathcal{N}$. An assignment to each of the query variables yields a *possible world* I that must contain all positive evidence variables, i.e., $I \supseteq \mathcal{P}$, and must not contain any negatives, i.e., $I \cap \mathcal{N} = \emptyset$.

Boolean Rules We first present the semantics of *Boolean* inference rules. For ease of exposition only, we assume that there is a single domain $\mathbb{D}$. A rule γ is a pair (q, w) such that q is a Boolean query and w is a real number. An example is as follows:

$$q() :\text{-}\, R(x, y), S(y) \quad \texttt{weight} = w.$$

We denote the body predicates of q as $\texttt{body}(\bar{z})$ where $\bar{z}$ are all variables in the body of $q()$, e.g., $\bar{z} = (x, y)$ in the example above. Given a rule $\gamma = (q, w)$ and a possible world I, we define the *sign of* γ *on* I as $\mathsf{sign}(\gamma, I) = 1$ if $q() \in I$ and -1 otherwise.

Given $\bar{c} \in \mathbb{D}^{|\bar{z}|}$, a *grounding* of q w.r.t. $\bar{c}$ is a substitution $\texttt{body}(\bar{z}/\bar{c})$, where the variables in $\bar{z}$ are replaced with the values in $\bar{c}$. For example, for q above with $\bar{c} = (a, b)$ then $\texttt{body}(\bar{z}/(a, b))$ yields the grounding $R(a, b), S(b)$, which is a conjunction of facts. The *support* $n(\gamma, I)$ of a rule γ in a possible world I is the number of groundings $\bar{c}$ for which $\texttt{body}(\bar{z}/\bar{c})$ is satisfied in I:

$$n(\gamma, I) = |\{\bar{c} \in \mathbb{D}^{|\bar{z}|} \ : \ I \models \texttt{body}(\bar{z}/\bar{c}) \ \}|$$

The *weight of* γ *in* I is the product of three terms:

$$w(\gamma, I) = w \ \mathsf{sign}(\gamma, I) \ g(n(\gamma, I)), \qquad (1)$$

where g is a real-valued function defined on the natural numbers. For intuition, if $w(\gamma, I) > 0$, it adds a weight that indicates that the world is more likely. If $w(\gamma, I) < 0$, it indicates that the world is less likely. As motivated above, we introduce g to support multiple semantics. Figure 4 shows choices for g that are supported by DeepDive, which we compare in an example below.

Let Γ be a set of Boolean rules, the weight of Γ on a possible world I is defined as

$$W(\Gamma, I) = \sum_{\gamma \in \Gamma} w(\gamma, I).$$

This function allow us to define a probability distribution over the set J of possible worlds:

$$\Pr[I] = Z^{-1} \exp(W(\Gamma, I)) \text{ where } Z = \sum_{I \in J} \exp(W(\Gamma, I)), \quad (2)$$

and Z is called the *partition function*. This framework is able to compactly specify much more sophisticated distributions than traditional probabilistic databases [37].

Example 2.5. *We illustrate the semantics by example. From the Web, we could extract a set of relation mentions that supports* "Barack Obama is born in Hawaii" *and another set of relation mentions that support* "Barack Obama is born in Kenya." *These relation mentions provide conflicting information, and one common approach is to "vote." We abstract this as up or down votes about a fact* $q()$.

$$q() :\text{-} \ \mathsf{Up}(x) \qquad \mathsf{weight} = 1.$$
$$q() :\text{-} \ \mathsf{Down}(x) \qquad \mathsf{weight} = -1.$$

We can think of this as a having a single random variable $q()$ *in which the size of* Up *(resp.* Down*) is an evidence relation that indicates the number of "Up" (resp. "Down") votes. There are only two possible worlds: one in which* $q() \in I$ *(is true) and not. Let* $|\mathsf{Up}|$ *and* $|\mathsf{Down}|$ *be the sizes of* Up *and* Down*. Following Equation 1 and 2, we have*

$$\Pr[q()] = \frac{e^W}{e^{-W} + e^W}$$

where

$$W = g(|\mathsf{Up}|) - g(|\mathsf{Down}|).$$

Consider the case when $|\mathsf{Up}| = 10^6$ *and* $|\mathsf{Down}| = 10^6 - 100$*. In some scenarios, this small number of differing votes could be due to random noise in the data collection processes. One would expect a probability for* $q()$ *close to* 0.5*. In the linear semantics* $g(n) = n$*, the probability of* q *is* $(1 + e^{-200})^{-1} \approx 1 - e^{-200}$*, which is extremely close to* 1*. In contrast, if we set* $g(n) = \log(1 + n)$*, then* $\Pr[q()] \approx 0.5$*. Intuitively, the probability depends on their* ratio *of these votes. The* logical *semantics* $g(n) = \mathbb{1}_{n>0}$ *gives exactly* $\Pr[q()] = 0.5$*. However, it would do the same if* $|\mathsf{Down}| = 1$*. Thus, logical semantics may ignore the strength of the voting information. At a high level, ratio semantics can learn weights from examples with different raw counts but similar ratios. In contrast, linear is appropriate when the raw counts themselves are meaningful.*

No semantic subsumes the other, and each is appropriate in some application. We have found that in many cases the ratio semantics is more suitable for the application that the user wants to model. We show in the full version that these semantics also affect efficiency empirically and theoretically–even for the above simple example. Intuitively, sampling converges faster in the logical or ratio semantics because the distribution is less sharply peaked, which means that the sampler is less likely to get stuck in local minima.

Extension to General Rules. Consider a general inference rule $\gamma = (q, w)$, written as:

$$q(\bar{y}) :\text{-} \ \mathsf{body}(\bar{z}) \quad \mathsf{weight} = w(\bar{x}).$$

where $\bar{x} \subseteq \bar{z}$ and $\bar{y} \subseteq \bar{z}$. This extension allows *weight tying*. Given $\bar{b} \in \mathbb{D}^{|\bar{x} \cup \bar{y}|}$ where $\bar{b}_x$ (resp. $\bar{b}_y$) are the values of $\bar{b}$ in $\bar{x}$ (resp. $\bar{y}$), we expand γ to a set Γ of Boolean rules by substituting $\bar{x} \cup \bar{y}$ with values from $\mathbb{D}$ in all possible ways.

$$\Gamma = \{(q_{\bar{b}_y}, w_{\bar{b}_x}) \mid q_{\bar{b}_y}() :\text{-} \ \mathsf{body}(\bar{z}/\bar{b}) \text{ and } w_{\bar{b}_x} = w(\bar{x}/\bar{b}_x)\}$$

where each $q_{\bar{b}_y}()$ is a *fresh* symbol for distinct values of $\bar{b}_t$, and $w_{\bar{b}_x}$ is a real number. Rules created this way may have free variables in their bodies, e.g., $q(x) :\text{-} \ R(x, y, z)$ with $w(y)$ create $|\mathbb{D}|^2$ different rules of the form $q_a() :\text{-} \ R(a, b, z)$, one for each $(a, b) \in \mathbb{D}^2$, and rules created with the same value of b share the same weight. Tying weights allows one to create models succinctly.

Example 2.6. *We use the following as an example:*

$$\mathsf{Class}(x) :\text{-} \ R(x, f) \quad \mathsf{weight} = w(f).$$

This declares a binary classifier as follows. Each binding for x *is an object to classify as in* Class *or not. The relation* R *associates each object to its features. E.g.,* $R(a, f)$ *indicates that object* a *has a feature* f*.* $\mathsf{weight} = w(f)$ *indicates that weights are functions of feature* f*; thus, the same weights are tied across values for* a*. This rule declares a logistic regression classifier.*

It is straightforward formal extension to let weights be functions of the return values of UDFs as we do in DeepDive.

2.5 Inference on Factor Graphs

As in Figure 3, DeepDive explicitly constructs a factor graph for inference and learning using a set of SQL queries. Recall that 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 $\hat{w} : F \times \{0, 1\}^V \to \mathbb{R}$ is a weight function. We can identify possible worlds with assignments since each node corresponds to a tuple; moreover, in DeepDive, each hyperedge f corresponds to the set of groundings for a rule γ. In DeepDive, V and F are explicitly created using a set of SQL queries. These data structures are then passed to the sampler, which runs outside the database, to estimate the marginal probability of each node or tuple in the database. Each tuple is then reloaded into the database with its marginal probability.

Example 2.7. *Take the database instances and rules in Figure 3 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 3. Both* F1 *and* F2 *are implemented as SQL in DeepDive.*

To define the semantics, we use Equation 1 to define $\hat{w}(f, I) = w(\gamma, I)$, in which γ is the rule corresponding to

f. As before, we define $\hat{W}(F, I) = \sum_{f \in F} \hat{w}(f, I)$, and then the probability of a possible world is the following function:

$$\Pr[I] = Z^{-1} \exp\left\{\hat{W}(F, I)\right\} \text{ where } Z = \sum_{I \in J} \exp\{\hat{W}(F, I)\}$$

The main task that DeepDive conducts on factor graphs is statistical inference, i.e., for a given node, what is the marginal probability that this node takes the value 1? Since a node takes value 1 when a tuple is in the output, this process computes the marginal probability values returned to users. In general, computing these marginal probabilities is ♯P-hard [39]. Like many other systems, DeepDive uses Gibbs sampling [35] to estimate the marginal probability of every tuple in the database.

3. INCREMENTAL KBC

To help the KBC system developer be more efficient, we developed techniques to incrementally perform the grounding and inference step of KBC execution.

Problem Setting. Our approach to incrementally maintaining a KBC system runs in two phases. **(1) Incremental Grounding.** The goal of the incremental grounding phase is to evaluate an update of the DeepDive program to produce the "delta" of the modified factor graph, i.e., the modified variables ΔV and factors ΔF. This phase consists of relational operations, and we apply classic incremental view maintenance techniques. **(2) Incremental Inference.** The goal of incremental inference is given $(\Delta V, \Delta F)$ run statistical inference on the changed factor graph.

3.1 Standard Techniques: Delta Rules

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, i.e., 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 DRED algorithm [18] that 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, $t.\texttt{count}$ 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, a SQL query called a *"delta rule"*[7] is executed which processes these counts to generate modified variables ΔV and factors ΔF. We found that the overhead DRED is modest and the gains may be substantial, and so DeepDive always runs DRED–except on initial load.

3.2 Novel Techniques for Incremental Maintenance of Inference

We present three techniques for the incremental inference phase on factor graphs: given the set of modified variables ΔV and modified factors ΔF produced in the incremental grounding phase, our goal is to compute the new distribution. We split the problem into two phases. In the *materialization phase*, we are given access to the entire DeepDive program, and we attempt to store information about the original distribution, denoted $\Pr^{(0)}$. Each approach will store different information to use in the next phase, called the *inference phase*. The input to the inference phase is the materialized data from the preceding phase and the changes made to the factor graph, the modified variables ΔV and factors ΔF. Our goal is to perform inference with respect to the changed distribution, denoted $\Pr^{(\Delta)}$. For each approach, we study its space and time costs for materialization and the time cost for inference. We also analyze the empirical tradeoff between the approaches in Section 3.2.4.

3.2.1 Strawman: Complete Materialization

The strawman approach, complete materialization, is computationally expensive and often infeasible. We use it to set a baseline for other approaches.

Materialization Phase We explicitly store the value of the probability Pr[I] for every possible world I. This approach has perfect fidelity, but storing all possible worlds takes an exponential amount of space and time in the number of variables in the original factor graph. Thus, the strawman approach is often infeasible on even moderate-sized graphs.[8]

Inference Phase We use Gibbs sampling: even if the distribution has changed to $\Pr^{(\Delta)}$, we only need access to the new factors in $\Delta\Pi_{\mathcal{F}}$ and to Pr[I] to perform the Gibbs update. The speed improvement arises from the fact that we do not need to access all factors from the original graph and perform a computation with them, since we can look them up in Pr[I].

3.2.2 Sampling Approach

The sampling approach is a standard technique to improve over the strawman approach by storing a set of possible worlds sampled from the original distribution instead of storing all possible worlds. However, as the updated distribution $\Pr^{(\Delta)}$ is different from the distribution used to draw the stored samples, we cannot reuse them directly. We use a (standard) Metropolis-Hastings scheme to ensure convergence to the updated distribution.

Materialization Phase In the materialization phase, we store a set of possible worlds drawn from the original distribution. For each variable, we store the set of samples as a *tuple bundle*, as in MCDB [21]. A single sample for one random variable only requires 1 bit of storage. Therefore, the sampling approach can be efficient in terms of materialization space. In the KBC systems we evaluated, 100 samples require less than 5% of the space of the original factor graph.

Inference Phase We use the samples to generate *proposals* and adapt them to estimate the up-to-date distribution. This idea of using samples from similar distributions as proposals is standard in statistics, e.g., importance sampling, rejection sampling, and different variants of Metropolis-Hastings methods. After investigating these approaches, in DeepDive, we use the *independent Metropolis-Hastings* approach [2,

[7] For example, for the grounding procedure illustrated in Figure 3, the delta rule for F1 is $q^\delta(x) :- R^\delta(x, y)$.

[8] Compared with running inference from scratch, the strawman approach does not materialize any factors. Therefore, it is necessary for strawman to enumerate each possible world and save their probability because we do not know *a priori* which possible world will be used in the inference phase.

Algorithm 1 Variational Approach (Materialization)

Input: Factor graph $\mathsf{FG} = (V, \mathsf{F})$, regularization parameter λ, number of samples N for approximation.
Output: An approximated factor graph $\mathsf{FG}' = (V, \mathsf{F}')$

1: $I_1, ..., I_N \leftarrow \mathsf{N}$ samples drawn from FG.
2: $\mathsf{NZ} \leftarrow \{(v_i, v_j)$: v_i and v_j are in some factor in $\mathsf{FG}\}$.
3: $M \leftarrow$ covariance matrix estimated using $I_1, ..., I_N$, such that M_{ij} is the covariance between variable i and variable j. Set $M_{ij} = 0$ if $(v_i, v_j) \notin \mathsf{NZ}$.
4: Solve the following optimization problem using gradient descent [4], and let the result be $\hat{X}$

$$\begin{aligned} \arg\max_X \quad & \log\det X \\ \text{s.t.}, \quad & X_{kk} = M_{kk} + 1/3, \\ & |X_{kj} - M_{kj}| \leqslant \lambda \\ & X_{kj} = 0 \text{ if } (v_k, v_j) \notin \mathsf{NZ} \end{aligned}$$

5: **for all** i, j s.t. $\hat{X}_{ij} \neq 0$ **do**
6: Add in F' a factor from (v_i, v_j) with weight $\hat{X}_{ij}$.
7: **end for**
8: **return** $\mathsf{FG}' = (V, \mathsf{F}')$.

35], which generates proposal samples and accepts these samples with an *acceptance test*. We choose this method only because the acceptance test can be evaluated using the sample, ΔV, and $\Delta \mathsf{F}$–without the entire factor graph. Thus, we may fetch many fewer factors than in the original graph, but we still converge to the correct answer.

The fraction of accepted samples is called the *acceptance rate*, and it is a key parameter in the efficiency of this approach. The approach may exhaust the stored samples, in which case the method resorts to another evaluation method or generates fresh samples.

3.2.3 Variational Approach

The intuition behind our variational approach is as follows: rather than storing the exact original distribution, we store a factor graph with *fewer factors* that approximates the original distribution. On the smaller graph, running inference and learning is often faster.

Materialization Phase The key idea of the variational approach is to approximate the distribution using simpler or sparser correlations. To learn a sparser model, we use Algorithm 1 which is a log-determinant relaxation [38] with a ℓ_1 penalty term [4]. We want to understand its strengths and limitations on KBC problems, which is novel. This approach uses standard techniques for learning that are already implemented in DeepDive [45].

The input is the original factor graph and two parameters: the number of samples N to use for approximating the covariance matrix, and the regularization parameter λ, which controls the sparsity of the approximation. The output is a new factor graph that has only binary potentials. The intuition for this procedure comes from graphical model structure learning: an entry (i, j) is present in the inverse covariance matrix only if variables i and j are connected in the factor graph. Given these inputs, the algorithm first draws a set of N possible worlds by running Gibbs sampling on the original factor graph. It then estimates the covariance matrix based on these samples (Lines 1-3). Using the estimated covariance matrix, our algorithms solves the optimization problem in Line 4 to estimate the inverse covariance matrix $\hat{X}$. Then, the algorithm creates one factor for each pair of variables such that the corresponding entry in $\hat{X}$ is non-zero, using the value in $\hat{X}$ as the new weight (Line 5-7). These are all the factors of the approximated factor graph (Line 8).

Inference Phase Given an update to the factor graph (e.g., new variables or new factors), we simply apply this update to the approximated graph, and run inference and learning directly on the resulting factor graph. As shown in Figure 5(c), the execution time of the variational approach is roughly linear in the sparsity of the approximated factor graph. Indeed, the execution time of running statistical inference using Gibbs sampling is dominated by the time needed to fetch the factors for each random variable, which is an expensive operation requiring random access. Therefore, as the approximated graph becomes sparser, the number of factors decreases and so does the running time.

Parameter Tuning We are among the first to use these methods in KBC applications, and there is little literature about tuning λ. Intuitively, the smaller λ is, the better the approximation is–but the less sparse the approximation is. To understand the impact of λ on quality, we show in Figure 6 the quality F1 score of a DeepDive program called News (see Section 4) as we vary the regularization parameter. As long as the regularization parameter λ is small (e.g., less than 0.1), the quality does not change significantly. In all of our applications we observe that there is a relatively large "safe" region from which to choose λ. In fact, for all five systems in Section 4, even if we set λ at 0.1 or 0.01, the impact on quality is minimal (within 1%), while the impact on speed is significant (up to an order of magnitude). Based on Figure 6, DeepDive supports a simple search protocol to set λ. We start with a small λ, e.g., 0.001, and increase it by $10\times$ until the KL-divergence is larger than a user-specified threshold, specified as a parameter in DeepDive.

3.2.4 Tradeoffs

We studied the tradeoff between different approaches and summarize the empirical results of our study in Figure 5. The performance of different approaches may differ by more than two orders of magnitude, and neither of them dominates the other. We use a synthetic factor graph with pairwise factors[9] and control the following axes:

(1) Number of variables in the factor graph. In our experiments, we set the number of variables to values in {2, 10, 17, 100, 1000, 10000}.

(2) Amount of change. How much the distribution changes affects efficiency, which manifests itself in the acceptance rate: the smaller the acceptance rate is, the more difference there will be in the distribution. We set the acceptance rate to values in {1.0, 0.5, 0.1, 0.01}.

(3) Sparsity of correlations. This is the ratio between the number of non-zero weights and the total weight. We set the sparsity to values in $\{0.1, 0.2, 0.3, 0.4, 0.5, 1.0\}$ by selecting uniformly at random a subset of factors and set their weight to zero.

We now discuss the results of our exploration of the tradeoff space, presented in Figure 5(a-c).

[9]In Figure 5, the numbers are reported for a factor graph whose factor weights are sampled at random from $[-0.5, 0.5]$. We also experimented with different intervals ($[-0.1, 0.1]$, $[-1, 1]$, $[-10, 10]$), but these had no impact on the tradeoff

		Strawman	Sampling	Variational
Mat. Phase	Space	2^{n_a}	$S_I n_a/\varrho$	n_a^2
	Cost	$2^{n_a} S_M \times C(n_a, f)$	$S_I C(n_a, f)/\varrho$	$n_a^2 + S_M C(n_a, f)$
Inference Phase	Cost	$S_I \times C(n_a+n_f, 1+f')$	$S_I n_a/\varrho + S_I C(n_f, f')/\varrho$	$S_I \times C(n_a+n_f, n_a^2+f')$
Sensitivity	Size of the Graph	**High**	Low	Mid
	Amount of Change	Low	**High**	Low
	Sparsity of Correlation	Low	Low	**High**

n_a: # original vars f: # original factors ϱ: acceptance rate
n_f: # modified vars f': # modified factors S_I: # samples for inference
$C(\#v, \#f)$: Cost of Gibbs with $\#v$ vars, and $\#f$ factors S_M: # samples for materialization

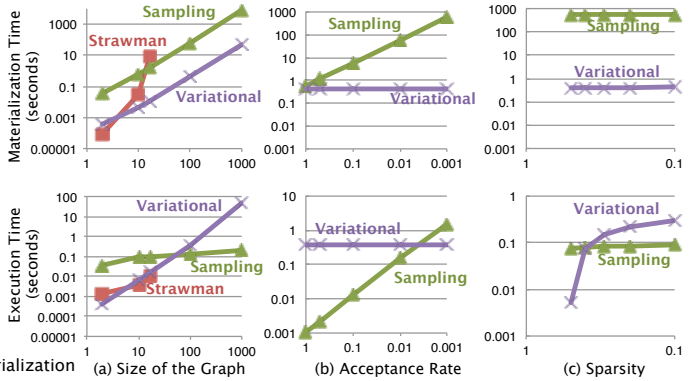


Figure 5: A Summary of the tradeoffs. Left: An analytical cost model for different approaches; Right: Empirical examples that illustrate the tradeoff space. All converge to <0.1% loss, and thus, have comparable quality.

Size of the Factor Graph Since the materialization cost of the strawman is exponential in the size of the factor graph, we observe that, for graphs with more than 20 variables, the strawman is significantly slower than either the sampling approach or the variational approach. Factor graphs arising from KBC systems usually contain a much larger number of variables; therefore, from now on, we focus on the tradeoff between sampling and variational approaches.

Amount of Change As shown in Figure 5(b), when the acceptance rate is high, the sampling approach could outperform the variational one by more than two orders of magnitude. When the acceptance rate is high, the sampling approach requires no computation and so is much faster than Gibbs sampling. In contrast, when the acceptance rate is low, e.g., 0.1%, the variational approach could be more than 5× faster than the sampling approach. An acceptance rate lower than 0.1% occurs for KBC operations when one updates the training data, adds many new features, or concept drift happens during the development of KBC systems.

Sparsity of Correlations As shown in Figure 5(c), when the original factor graph is sparse, the variational approach can be 11× faster than the sampling approach. This is because the approximate factor graph contains less than 10% of the factors than the original graph, and it is therefore much faster to run inference on the approximate graph. On the other hand, if the original factor graph is too dense, the variational approach could be more than 7× slower than the sampling one, as it is essentially performing inference on a factor graph with a size similar to that of the original graph.

Discussion: Theoretical Guarantees We discuss the theoretical guarantee that each materialization strategy provides. Each materialization method inherits the guarantee of that inference technique. The strawman approach retains the same guarantees as Gibbs sampling; For the sampling approach use standard Metropolis-Hasting scheme. Given enough time, this approach will converge to the true distribution. For the variational approach, the guarantees are more subtle and we point the reader to the consistency of structure estimation of Gaussian Markov random field [33] and log-determinate relaxation [38]. These results are theoretically incomparable, motivating our empirical study.

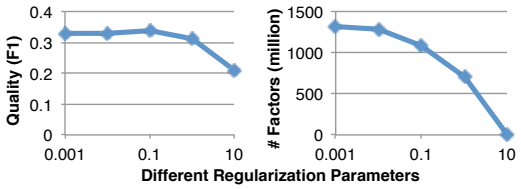


Figure 6: Quality and number of factors of the News corpus with different regularization parameters for the variational approach.

3.3 Choosing Between Different Approaches

From the study of the tradeoff space, neither the sampling approach nor the variational approach dominates the other, and their relative performance depends on how they are being used in KBC. We propose to materialize the factor graph using both the sampling approach *and* the variational approach, and defer the decision to the inference phase when we can observe the workload.

Materialization Phase Both approaches need samples from the original factor graph, and this is the dominant cost during materialization. A key question is *"How many samples should we collect?"* We experimented with several heuristic methods to estimate the number of samples that are needed, which requires understanding how likely future changes are, statistical considerations, etc. These approaches were difficult for users to understand, so DeepDive takes a best-effort approach: it generates as many samples as possible when idle or within a user-specified time interval.

Inference Phase Based on the tradeoffs analysis, we developed a *rule-based optimizer* with the following set of rules:

- If an update does not change the structure of the graph, choose the sampling approach.
- If an update modifies the evidence, choose the variational approach.
- If an update introduces new features, choose the sampling approach.
- Finally, if we run out of samples, use the variational approach.

This simple set of rules is used in our experiments.

System	# Docs	# Rels	# Rules	# vars	# factors
Adversarial	5M	1	10	0.1B	0.4B
News	1.8M	34	22	0.2B	1.2B
Genomics	0.2M	3	15	0.02B	0.1B
Pharma.	0.6M	9	24	0.2B	1.2B
Paleontology	0.3M	8	29	0.3B	0.4B

Figure 7: Statistics of KBC systems we used in experiments. The # vars and # factors are for factor graphs that contain all rules.

Rule	Description
A1	Calculate marginal probability for variables or variable pairs.
FE1	Shallow NLP features (e.g., word sequence)
FE2	Deeper NLP features (e.g., dependency path)
I1	Inference rules (e.g., symmetrical HasSpouse).
S1	Positive examples
S2	Negative examples

Figure 8: The set of rules in News. See Section 4.1

4. EXPERIMENTS

We conducted an experimental evaluation of DeepDive for incremental maintenance of KBC systems.

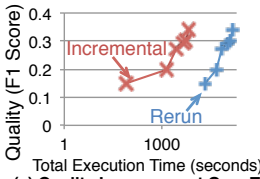
4.1 Experimental Settings

To evaluate DeepDive, we used DeepDive programs developed by our users over the last three years from paleontologists, geologists, biologists, a defense contractor, and a KBC competition. These are high-quality KBC systems: two of our KBC systems for natural sciences achieved quality comparable to (and sometimes better than) human experts, as assessed by double-blind experiments, and our KBC system for a KBC competition is the top system among all 45 submissions from 18 teams as assessed by professional annotators. To simulate the development process, we took snapshots of DeepDive programs at the end of every development iteration, and we use this dataset of snapshots in the experiments to understand our hypothesis that incremental techniques can be used to improve development speed.

Datasets and Workloads To study the efficiency of DeepDive, we selected five KBC systems, namely (1) News, (2) Genomics, (3) Adversarial, (4) Pharmacogenomics, and (5) Paleontology. Their names refers to the specific domains on which they focus. Figure 7 illustrates the statistics of these KBC systems and of their input datasets. We group all rules in each system into six rule templates with four workload categories. We focus on the News system below.

The News system builds a knowledge base between persons, locations, and organizations, and contains 34 different relations, e.g., HasSpouse or MemberOf. The input to the KBC system is a corpus that contains 1.8 million news articles and Web pages. We use four types of rules in News in our experiments, as shown in Figure 8, error analysis (rule A1), candidate generation and feature extraction (FE1, FE2), supervision (S1, S2), and inference (I1), corresponding to the steps where these rules are used.

Other applications are different in terms of the quality of the text. We choose these systems as they span a large range in the spectrum of quality: Adversarial contains advertisements collected from websites where each document may have only 1-2 sentences with grammatical errors; in contrast, Paleontology contains well-curated journal articles with precise, unambiguous writing and simple relationships.



(a) Quality Improvement Over Time

	Adv	News	Gen	Pha	Pale
Linear	**0.72**	0.32	0.47	0.52	0.74
Logical	**0.72**	**0.34**	**0.53**	0.56	0.80
Ratio	**0.72**	**0.34**	**0.53**	**0.57**	**0.81**

(b) Quality (F1) of Different Semantics

Figure 10: (a) Quality improvement over time; (b) Quality for different semantics.

Genomics and Pharma have precise texts, but the goal is to extract relationships that are more linguistically ambiguous compared to the Paleontology text. News has slightly degraded writing and ambiguous relationships, e.g., "member of." Rules with the same prefix, e.g., FE1 and FE2, belong to the same category, e.g., feature extraction.

DeepDive Details DeepDive is implemented in Scala and C++, and we use Greenplum to handle all SQL. All feature extractors are written in Python. The statistical inference and learning and the incremental maintenance component are all written in C++. All experiments are run on a machine with four CPUs (each CPU is a 12-core 2.40 GHz Xeon E5-4657L), 1 TB RAM, and 12×1TB hard drives and running Ubuntu 12.04. For these experiments, we compiled DeepDive with Scala 2.11.2, g++-4.9.0 with -O3 optimization, and Python 2.7.3. In Genomics and Adversarial, Python 3.4.0 is used for feature extractors.

4.2 End-to-end Performance and Quality

We built a modified version of DeepDive called Rerun, which given an update on the KBC system, runs the DeepDive program from scratch. DeepDive, which uses all techniques, is called Incremental. The results of our evaluation show that DeepDive is able to speed up the development of high-quality KBC systems through incremental maintenance with little impact on quality. We set the number of samples to collect during execution to {10, 100, 1000} and the number of samples to collect during materialization to {1000, 2000}. We report results for (1000, 2000), as results for other combinations of parameters are similar.

Quality Over Time We first compare Rerun and Incremental in terms of the wait time that developers experience to improve the quality of a KBC system. We focus on News because it is a well-known benchmark competition. We run all six rules sequentially for both Rerun and Incremental, and after executing each rule, we report the quality of the system measured by the F1 score and the *cumulative* execution time. Materialization in the Incremental system is performed only once. Figure 10(a) shows the results. Using Incremental takes significantly less time than Rerun to achieve the same quality. To achieve an F1 score of 0.36 (a competition-winning score), Incremental is 22× faster than Rerun. Indeed, each run of Rerun takes ≈ 6 hours, while a run of Incremental takes at most 30 minutes.

We further compare the facts extracted by Incremental and Rerun and find that these two systems not only have similar end-to-end quality, but are also similar enough to support common debugging tasks. We examine the facts with high-confidence in Rerun (> 0.9 probability), 99% of

Rule	Adversarial			News			Genomics			Pharma.			Paleontology		
	Rerun	Inc.	×	Rerun	Inc.	×	Rerun	Inc.	×	Rerun	Inc.	×	Rerun	Inc.	×
A1	1.0	0.03	33×	2.2	0.02	112×	0.3	0.01	30×	3.6	0.11	33×	2.8	0.3	10×
FE1	1.1	0.2	7×	2.7	0.3	10×	0.4	0.07	6×	3.8	0.3	12×	3.0	0.4	7×
FE2	1.2	0.2	6×	3.0	0.3	10×	0.4	0.07	6×	4.2	0.3	12×	3.3	0.4	8×
I1	1.3	0.2	6×	3.6	0.3	10×	0.5	0.09	6×	4.4	1.4	3×	3.8	0.5	8×
S1	1.3	0.2	6×	3.6	0.4	8×	0.6	0.1	6×	4.7	1.7	3×	4.0	0.5	7×
S2	1.3	0.3	5×	3.6	0.5	7×	0.7	0.1	7×	4.8	2.3	3×	4.1	0.6	7×

Figure 9: End-to-end efficiency of incremental inference and learning. All execution times are in hours. The column × refers to the speedup of INCREMENTAL (Inc.) over RERUN.

them also appear in INCREMENTAL, and vice versa. High confidence extractions are used by the developer to debug precision issues. Among all facts, we find that at most 4% of them have a probability that differs by more than 0.05. The similarity between snapshots suggests, our incremental maintenance techniques can be used for debugging.

Efficiency of Evaluating Updates We now compare RERUN and INCREMENTAL in terms of their speed in evaluating a given update to the KBC system. To better understand the impact of our technical contribution, we divide the total execution time into parts: (1) the time used for feature extraction and grounding; and (2) the time used for statistical inference and learning. We implemented classical incremental materialization techniques for feature extraction and grounding, which achieves up to a 360× speedup for rule FE1 in News. We get this speedup for free using standard RDBMS techniques, a key design decision in DeepDive.

Figure 9 shows the execution time of statistical inference and learning for each update on different systems. We see from Figure 9 that INCREMENTAL achieves a 7× to 112× speedup for News across all categories of rules. The analysis rule A1 achieves the highest speedup – this is not surprising because, after applying A1, we do not need to rerun statistical learning, and the updated distribution does not change compared with the original distribution, so the sampling approach has a 100% acceptance rate. The execution of rules for feature extraction (FE1, FE2), supervision (S1, S2), and inference (I1) has a 10× speedup. For these rules, the speedup over RERUN is to be attributed to the fact that the materialized graph contains only 10% of the factors in the full original graph. Below, we show that both the sampling approach and variational approach contribute to the speed-up. Compared with A1, the speedup is smaller because these rules produce a factor graph whose distribution changes more than A1. Because the difference in distribution is larger, the benefit of incremental evaluation is lower.

The execution of other KBC applications showed similar speedups, but there are also several interesting data points. For Pharmacogenomics, rule I1 speeds-up only 3×. This is caused by the fact that I1 introduces many new factors, and the new factor graph is 1.4× larger than the original one. In this case, DeepDive needs to evaluate those new factors, which is expensive. For Paleontology, we see that the analysis rule A1 gets a 10× speed-up because as illustrated in the corpus statistics (Figure 7), the Paleontology factor graph has fewer factors for each variable than other systems. Therefore, executing inference on the whole factor graph is cheaper.

Materialization Time One factor that we need to consider is the materialization time for INCREMENTAL. INCREMENTAL took 12 hours to complete the materialization (2000 samples), for each of the five systems. Most of this time is spent in getting 2× more samples than for a single run of RERUN. We argue that paying this cost is worthwhile given that it is a one-time cost and the materialization can be used for many successive updates, amortizing the one-time cost.

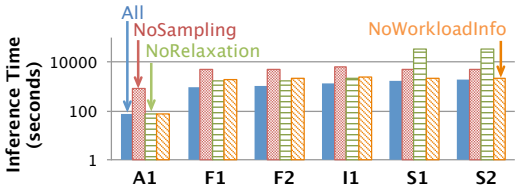


Figure 11: Study of the tradeoff space on News.

4.3 Lesion Studies

We conducted lesion studies to verify the effect of the tradeoff space on the performance of DeepDive. In each lesion study, we disable a component of DeepDive, and leave all other components untouched. We report the execution time for statistical inference and learning.

We evaluate the impact of each materialization strategy on the final end-to-end performance. We disabled either the sampling approach or the variational approach and left all other components of the system untouched. Figure 11 shows the results for News. Disabling either the sampling approach or the variational approach slows down the execution compared to the "full" system. For analysis rule A1, disabling the sampling approach leads to a more than 11× slow down, because the sampling approach has, for this rule, a 100% acceptance rate because the distribution does not change. For feature extraction rules, disabling the sampling approach slows down the system by 5× because it forces the use of the variational approach even when the distribution for a group of variables does not change. For supervision rules, disabling the variational approach is 36× slower because the introduction of training examples decreases the acceptance rate of the sampling approach.

Optimizer Using different materialization strategies for different groups of variables positively affects the performance of DeepDive. We compare INCREMENTAL with a strong baseline NOWORKLOADINFO which, for each group, first runs the sampling approach. After all samples have been used, we switch to the variational approach. Note that this baseline is stronger than the strategy that fixes the same strategy for all groups. Figure 11 shows the results of the experiment. We see that with the ability to choose between the sampling approach and variational approach according to the workload, DeepDive can be up to 2× faster than NOWORKLOADINFO.

5. CONCLUSION

We described the DeepDive approach to KBC and our experience building KBC systems over the last few years. To improve quality, we argued that a key challenge is to accelerate the development loop. We described the semantic choices that we made in our language. By building on SQL, DeepDive is able to use classical techniques to provide incremental processing for the SQL components. However, these classical techniques do not help with statistical inference, and we described a novel tradeoff space for approximate inference techniques. We used these approximate inference techniques to improve end-to-end execution time in the face of changes both to the program and the data; they improved system performance by two orders of magnitude in five real KBC scenarios while keeping the quality high enough to aid in the development process.

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 No. N000141210041 and No. 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 U.S. government.

6. REFERENCES

[1] U. A. Acar, A. Ihler, R. Mettu, and O. Sümer. Adaptive inference on general graphical models. In *UAI*, 2008.

[2] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 2003.

[3] G. Angeli, S. Gupta, M. Jose, C. D. Manning, C. Ré, J. Tibshirani, J. Y. Wu, S. Wu, and C. Zhang. Stanford's 2014 slot filling systems. *TAC KBP*, 2014.

[4] O. Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *JMLR*, 2008.

[5] J. Betteridge, A. Carlson, S. A. Hong, E. R. Hruschka Jr, E. L. Law, T. M. Mitchell, and S. H. Wang. Toward never ending language learning. In *AAAI Spring Symposium*, 2009.

[6] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB*, 1999.

[7] E. Brown, E. Epstein, J. W. Murdock, and T.-H. Fin. Tools and methods for building watson. *IBM Research Report*, 2013.

[8] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[9] F. Chen, A. Doan, J. Yang, and R. Ramakrishnan. Efficient information extraction over evolving text data. In *ICDE*, 2008.

[10] F. Chen, X. Feng, C. Re, and M. Wang. Optimizing statistical information extraction programs over evolving text. In *ICDE*, 2012.

[11] Y. Chen and D. Z. Wang. Knowledge expansion over probabilistic knowledge bases. In *SIGMOD*, 2014.

[12] A. L. Delcher, A. Grove, S. Kasif, and J. Pearl. Logarithmic-time updates and queries in probabilistic networks. *J. Artif. Intell. Res.*, 1996.

[13] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, 2009.

[14] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. In *VLDB*, 2014.

[15] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll: (preliminary results). In *WWW*, 2004.

[16] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty. Building Watson: An overview of the DeepQA project. *AI Magazine*, 2010.

[17] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project: Back and forth between theory and practice. In *PODS*, 2004.

[18] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. *SIGMOD Rec.*, 1993.

[19] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 1992.

[20] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *ACL*, 2011.

[21] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A Monte Carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[22] E. T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[23] S. Jiang, D. Lowd, and D. Dou. Learning to refine an automatically extracted knowledge base using Markov logic. In *ICDM*, 2012.

[24] M. L. Koc and C. Ré. Incrementally maintaining classification using an RDBMS. *PVLDB*, 2011.

[25] Y. Li, F. R. Reiss, and L. Chiticariu. SystemT: A declarative information extraction system. In *HLT*, 2011.

[26] J. Madhavan, S. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale data integration: You can only afford to pay as you go. In *CIDR*, 2007.

[27] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, 2009.

[28] N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *WSDM*, 2011.

[29] A. Nath and P. Domingos. Efficient belief propagation for utility maximization and repeated inference. In *AAAI*, 2010.

[30] F. Niu, C. Ré, A. Doan, and J. Shavlik. Tuffy: Scaling up statistical inference in Markov logic networks using an RDBMS. *PVLDB*, 2011.

[31] F. Niu, C. Zhang, C. Ré, and J. Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *Int. J. Semantic Web Inf. Syst.*, 2012.

[32] S. E. Peters, C. Zhang, M. Livny, and C. Ré. A machine reading system for assembling synthetic Paleontological databases. *PloS ONE*, 2014.

[33] P. D. Ravikumar, G. Raskutti, M. J. Wainwright, and B. Yu. Model selection in gaussian graphical models: High-dimensional consistency of ℓ_1-regularized MLE. In *NIPS*, 2008.

[34] C. Ré, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang, S. Wu, and C. Zhang. Feature engineering for knowledge base construction. *IEEE Data Eng. Bull.*, 2014.

[35] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[36] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *VLDB*, 2007.

[37] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

[38] M. Wainwright and M. Jordan. Log-determinant relaxation for approximate inference in discrete Markov random fields. *Trans. Sig. Proc.*, 2006.

[39] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *FTML*, 2008.

[40] G. Weikum and M. Theobald. From information to knowledge: Harvesting entities and relationships from web sources. In *PODS*, 2010.

[41] M. Wick and A. McCallum. Query-aware MCMC. In *NIPS*, 2011.

[42] M. Wick, A. McCallum, and G. Miklau. Scalable probabilistic databases with factor graphs and MCMC. *PVLDB*, 2010.

[43] L. Yao, S. Riedel, and A. McCallum. Collective cross-document relation extraction without labelled data. In *EMNLP*, 2010.

[44] C. Zhang and C. Ré. Towards high-throughput Gibbs sampling at scale: A study across storage managers. In *SIGMOD*, 2013.

[45] C. Zhang and C. Ré. DimmWitted: A study of main-memory statistical analytics. *PVLDB*, 2014.