Automating the Generation of Hardware Component Knowledge Bases

Luke Hsiao
Stanford University, USA
lwhsiao@stanford.edu

Sen Wu
Stanford University, USA
senwu@cs.stanford.edu

Nicholas Chiang
Gunn High School, USA

Christopher Ré
Stanford University, USA
chrismre@cs.stanford.edu

Philip Levis
Stanford University, USA
pal@cs.stanford.edu

Abstract

Hardware component databases are critical resources in designing embedded systems. Since generating these databases requires hundreds of thousands of hours of manual data entry, they are proprietary, limited in the data they provide, and have many random data entry errors.

We present a machine-learning based approach for automating the generation of component databases directly from datasheets. Extracting data directly from datasheets is challenging because: (1) the data is relational in nature and relies on non-local context, (2) the documents are filled with technical jargon, and (3) the datasheets are PDFs, a format that decouples visual locality from locality in the document. The proposed approach uses a rich data model and weak supervision to address these challenges.

We evaluate the approach on datasheets of three classes of hardware components and achieve an average quality of 75 F1 points which is comparable to existing human-curated knowledge bases. We perform two applications studies that demonstrate the extraction of multiple data modalities such as numerical properties and images. We show how different sources of supervision such as heuristics and human labels have distinct advantages which can be utilized together within a single methodology to automatically generate hardware component knowledge bases.

CCS Concepts • **Information systems** → *Data mining*; • **Hardware** → *Electronic design automation.*

Keywords Knowledge Base Construction, Design Tools

LCTES '19, June 23, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6724-0/19/06…$15.00
https://doi.org/10.1145/3316482.3326344

ACM Reference Format:
Luke Hsiao, Sen Wu, Nicholas Chiang, Christopher Ré, and Philip Levis. 2019. Automating the Generation of Hardware Component Knowledge Bases. In *Proceedings of the 20th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '19), June 23, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3316482.3326344

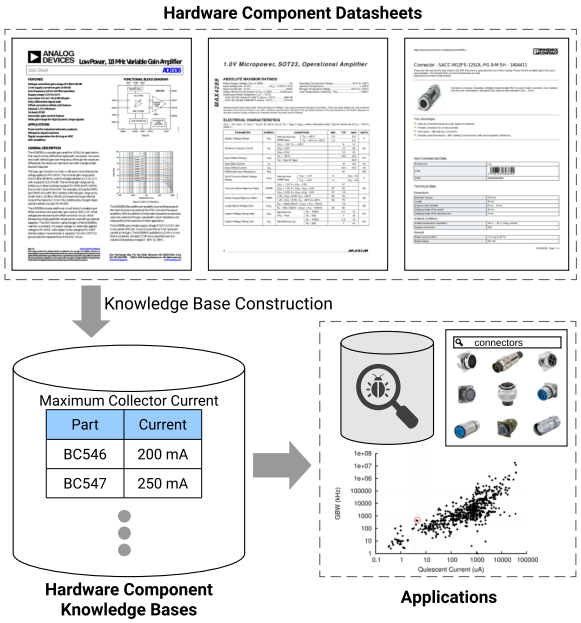
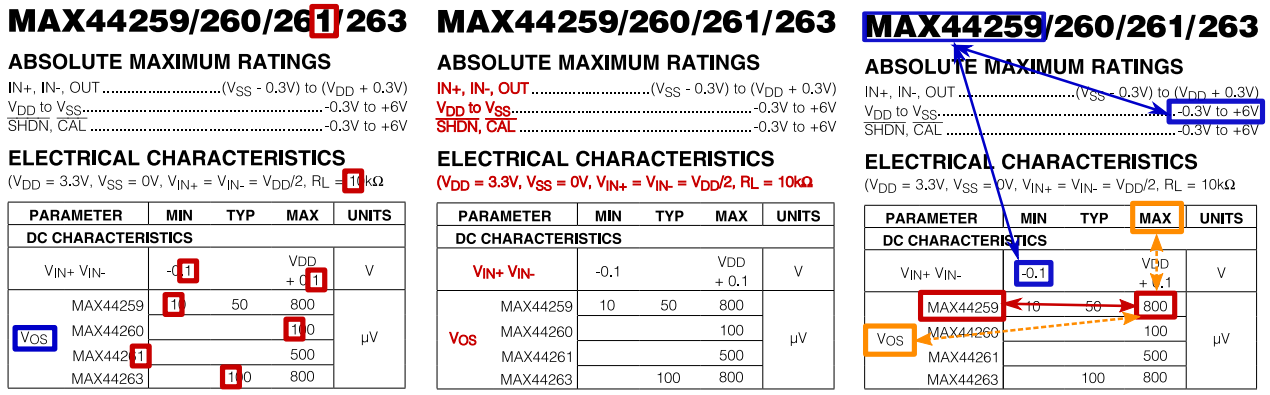


Figure 1. Hardware component knowledge bases are populated from datasheets and serve valuable applications such as cross validation, selecting components based on optimal electrical characteristics, or building rich search interfaces.

1 Introduction

Creating embedded systems often requires developing new hardware. Searching for components that best meet system requirements constitutes a significant portion of design time. Downloading a datasheet is easy, but figuring out *which* datasheet to download is hard. The needed information is hidden in the datasheet itself, a complex document that is impenetrable to standard search engines. Requirements are



(a) Relational data: a keyword search for "V_{OS}" and "1" may match 1000s of documents as both terms are commonly used. Instead, engineers want to query relational data, e.g., whether a specific part has a minimum "V_{OS}" value of "1 µV".

(b) Jargon: datasheets use extensive technical jargon such as the symbols highlighted in red. Understanding a datasheet requires both technical expertise and deep experience.

(c) Input format: PDF documents lack structural information (e.g., explicit tables), so relationships must be inferred from the rendering of the text, vectors, and images in a document using cues like alignments, proximity, and sizes.

Figure 2. An example document highlighting the challenges of extracting information from PDF datasheets.

typically multi-dimensional and quantitative, so selecting the right component involves ranges across multiple properties, such as voltage, gain, and limits. Finally, there are often many (e.g., thousands) different versions of a component with equivalent functionality but differences in cost, energy, or size. Today's hardware engineers search by visiting many different web search pages, delicately tuning parameters on each one to get a handful (not zero, not hundreds) of results, manually aggregating the results, then inspecting individual datasheets for information not available in web search forms.

This laborious process means that quickly designing hardware requires a library of components in one's head, gained through deep experience. For people without these years of experience, hardware design remains a formidable challenge: maker forums have detailed discussions on picking the right transistor [1], and entire research papers hinge on careful hardware component selection [10].

The challenge of finding hardware components are in stark contrast to the ease of finding good software libraries. Software library information is easily accessible and searchable: searches can find easy-to-use libraries such as web servers, graphics, or data analysis. Searches are textual, so can be easily answered by crawling documentation, package descriptions, or community boards such as Stack Overflow. Any given search typically turns up at most a small number of well-maintained libraries for a given purpose; there are not hundreds of graphing packages comparable to `matplotlib` or hundreds of secure socket libraries comparable to `libssl`.

Hardware component databases are valuable tools for hardware developers. As shown in Figure 1, applications and tools can use these knowledge bases to cross-validate existing databases, answer questions like "which operational amplifiers should I use to build this gain circuit", or even to query non-textual data like product thumbnails.

Services like Digi-Key, Mouser, and Parts.io help hardware developers by building proprietary databases: they offer component search pages to drive billions of dollars in sales [6]. These databases are generated manually. People who have enough technical expertise to understand a datasheet (e.g., whether *Vcc* and *Vdd* are interchangeable in a given setting) enter their data by hand. Human entry, however, is prone to random errors or errors based on a particular individual's biases [9]. Furthermore, these databases are incomplete. The cost of data entry means that those databases contain a limited subset of the available information, limited to floating point numbers or small enumerations (e.g., type of resistor).

1.1 Learning to Construct Component Databases

This paper proposes making hardware component information both accessible and cheap by automating the generation of databases from datasheets using state-of-the-art machine learning. This problem requires machine learning because datasheets are complex, richly-formatted documents that rely on many implicit signals and structures to communicate information. Addressing datasheet complexity has traditionally required manual human intervention. Extracting information from datasheets has three key challenges: relational data, jargon, and input format. Figure 2 shows examples of these challenges drawn from a sample datasheet.

First, hardware component information is relational in nature. Users typically want to search for quantitative values of a variety of electrical characteristics (Figure 2a). This causes traditional search tools that treat documents as unstructured text to be ineffective since text-based search alone cannot adequately express these complex relationships and keywords commonly match 1000s of documents.

Second, datasheets describe components using technical detail and jargon in a wide variety of ways (see Figure 2b).

Extracting their data requires capturing this domain knowledge in a learning system and precludes relying on untrained crowdsourcing services such as Amazon Mechanical Turk.

Third, datasheets are distributed in Portable Document Format (PDF), and vendors vary significantly in how they present data using textual, structural, tabular, and visual cues (Figure 2c). These cues are understandable to humans but challenging for machines to interpret. Further, the wide variety and non-uniformity of these cues make them impossible to address accurately by simply applying heuristics.

1.2 Proposed Approach

We propose a general methodology for automating the generation of hardware component knowledge bases. Our methodology builds hardware component knowledge bases by reading thousands of PDF datasheets of multiple component types as input and populates relational databases as output.

We use three machine-learning techniques to address the challenges of hardware datasheets. First, we use a rich data model that captures the multiple modalities of information provided in a PDF document, rather than modeling input as unstructured text. This allows us to encode features based on textual, structural, and visual information. Second, we use weak supervision to efficiently translate domain knowledge into training data. Weak supervision provides us a way to combine and benefit from a wide variety of signals such as heuristics and expert human annotations. Third, we train a model that is robust to the data variety in hardware datasheets. Use of machine learning shifts database errors away from random, human errors toward more systematic errors that weak supervision can iteratively address and reduce.

Other domains have turned to automated methods for generating knowledge bases as a solution to making information accessible [17, 24]. These domains, however, used automated methods focused on unstructured text. In contrast, hardware datasheets are richly formatted documents with immense data variety that present dense numerical, graphical, and pictorial information written for a technical audience. Our approach builds on the Fonduer knowledge base construction framework [23]. We defer a complete description of the contributions beyond this prior work to Section 2.

1.3 Contributions

This paper makes three contributions:

1. A general methodology for building hardware component knowledge bases using state-of-the-art machine-learning techniques (Section 3).
2. The evaluation of our methodology for automating the generation of hardware component knowledge bases for multiple hardware components, extracting both textual and non-textual information (Section 4).
3. Application studies that highlight how these databases make hardware component selection easier (Section 4.3).

2 Background and Related Work

Component databases are a key resource in embedded hardware development. Generating these databases is laborious and error-prone. Experts with sufficient technical knowledge to read datasheets must be used to enter data. As a result, these databases are small unless they are proprietary databases owned by large component search companies. Their small size limits the practical utility of many tools [2, 11, 18, 19]. For example, Drew et. al. presented a tool for automatically checking breadboarded circuits, but its underlying knowledge base only supports six types of components [7]. Similarly, Ramesh et. al. demonstrate that with a database of components, one can automatically produce an embedded device hardware design from software, but defer generating a sufficient library to future work [20].

Recent developments in machine learning and knowledge base construction have demonstrated success in automating the creation of queryable knowledge bases in domains such as paleontology, electronics, and genomics [23]. We build on this prior work and apply these techniques into the domain of supporting embedded system development by targeting hardware component knowledge bases, which have great value but are error-prone and laborious to produce.

2.1 Knowledge Base Construction

The process of knowledge base construction takes a corpus of documents as input and outputs a relational database with a user-defined schema. This database is populated using information extracted from the input documents. We use the following terminology to describe the process.

A **mention**, m, represents a noun, i.e., a real-world person, place, or thing, which can be grouped and identified by its **mention type**, T. For example, "part number" is a mention type, while "BC546" is a corresponding mention. A relationship of n mentions is an n-ary **relation**, $R(m_1, m_2, \ldots, m_n)$, which corresponds to a **schema**, $S_R(T_1, T_2, \ldots, T_n)$. A **candidate** is an n-ary tuple, $c = (m_1, m_2, \ldots, m_n)$, which represents a potentially correct instance of a relation R. For instance, a "part number" and a "price" represent a relation with a schema, $S_R(T_1, T_2)$, where "BC546" and "$1.00" represent a candidate, $c = (m_1, m_2)$, of a 2-ary relation, $R(m_1, m_2)$.

In order to automate knowledge base construction, machine-learning-based systems model this process as a classification task. Candidates are extracted from the input documents and assigned a Boolean random variable where a true value signals that the candidate is a valid instance of a relation. In order to make that determination, each candidate is assigned a set of **features** as signals for which Boolean value a classifier should assign. Then, these systems maximize the probability of correctly classifying each candidate based on its features and a set of examples, called **training data**.

Ultimately, a supervised machine-learning algorithm requires three inputs: (1) candidates, (2) their features, and (3)

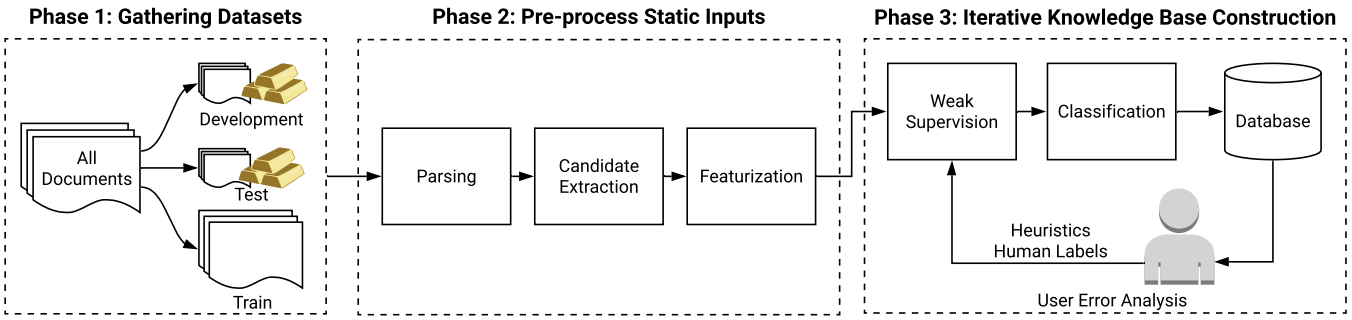


Figure 3. An overview of our methodology for automating the generation of hardware component knowledge bases.

training data. It then outputs a marginal probability for each of the input candidates. Finally, users specify a **threshold** for the output marginal probabilities. We classify candidates whose probability of being true is greater than the threshold as true, and vice versa. We quantify our results using an **F1 score**, which is the harmonic mean of the **precision** (the ratio of candidates classified as true which are correct) and **recall** (the ratio of correct candidates classified as true).

2.2 Weak Supervision

Training data is a vital input for knowledge base construction systems powered by machine learning. Traditionally, training data was incredibly costly to obtain, since it required domain experts to tediously label data. In recent years, *weak supervision* has emerged as a popular technique for generating training data. Weak supervision takes the approach of getting multiples sources of potentially lower-quality labels more efficiently, such as crowdsourcing [12, 25], leveraging existing knowledge bases [14], or using heuristics [21].

Users provide weak supervision in the form of **labeling functions**. A labeling function takes each candidate as input, and labels it as true, false, or abstains from voting. Labeling functions can use arbitrary heuristics, which allows them to capture a variety of weak supervision approaches. Because each labeling function can abstain from voting, each labeling function will potentially cover different subsets of the input data and, due to the varying quality of the weak supervision sources, may conflict with each other. We then follow the data programming paradigm [22] by using a generative probabilistic model to estimate the accuracy of each labeling function. These estimates are applied as weights to the output of each labeling function, resulting in a final probabilistic label for each candidate that can serve as training data. With this approach, rather than relying on manual labels alone, we can combine manual labels with programmatic heuristics and iteratively generate large amounts of training data.

2.3 The Fonduer Framework

We build upon the work of Fonduer, which provides a framework for extracting candidates from richly formatted data such as PDF datasheets [23]. In particular, Fonduer provides a feature library which captures signals from multiple modalities of information (e.g., textual, structural, tabular, and visual), and a rich data model for each document that we utilize for weak supervision in the form of labeling functions.

While [23] showed that extracting information from PDF datasheets is possible by extracting a few numerical values from transistor datasheets, this paper goes beyond their work in three ways. First, where Wu et al. showed that extracting component information was possible, we provide a practical methodology that details how to do so. Second, we show that our approach is generalizable by extracting hardware component information from three different types of components, and by extracting both graphical and textual data, whereas Wu et al. only show extraction of textual data from a single component. Third, our work demonstrates applications end-to-end, from dataset creation to application studies that use these knowledge bases, whereas Wu et al. focus on the creation of the knowledge base alone.

3 Methodology

We divide the process of generating hardware component knowledge bases into three phases: (1) gathering datasets, (2) pre-processing candidates and features as the static inputs to a machine-learning model, and (3) iterating until we achieve the desired quality (Figure 3). We highlight challenges and hard-earned best practices for each phase. The implementation of each computational block of the pipeline (e.g., parsing, candidate extraction, featurization, etc.) is detailed in [23].

3.1 Phase 1: Gathering Datasets

Generating hardware component knowledge bases begins with a high quality corpus of documents. Specifically, we require a corpus of documents that allows us to capture non-textual signals like document structure, and we require accompanying gold labels in order to evaluate the final quality of the knowledge base.

Acquiring Document Metadata The majority of manufacturers distribute hardware datasheets as PDF documents. These datasheets contain tables of relational information. However, unlike HTML or XML documents, which contain

structural metadata, PDF documents only contain characters, vectors, and images, and their rendering coordinates. While a datasheet may visually present data in a structured table, the underlying format contains no explicit metadata about document structure. We require supplementary metadata in addition to the raw characters, vectors, and images contained within a document. To satisfy this requirement, we use Adobe Acrobat to acquire metadata by generating an HTML representation for each PDF document[1]. While the conversion process introduces noise, the HTML metadata provides valuable information about document structure that complements the visual information in the PDF.

Preparing for Evaluation To evaluate the quality of the final knowledge base, we must have *gold labels*, or ground truth labels, which we can compare against (e.g., by calculating an F1 score). Because it is typically infeasible to obtain a large amount of gold labels, we instead only acquire gold labels for a small but representative subset of the input corpus. This subset is further divided into a set used for error inspection during development, and to a set used to assess generalization during final validation.

3.2 Phase 2: Pre-process Static Inputs

Machine-learning algorithms require two static inputs: candidates and their features. The third input, training data, is iteratively generated and refined in Phase 3. To generate candidates and features, we must (1) parse the input corpus into a richly formatted data model, (2) extract candidates, and (3) featurize each of these candidates.

Parsing Manufacturers distribute datasheets as richly formatted PDF documents that convey information through textual, structural, tabular, and visual cues. Therefore, it is vital that we preserve this rich metadata when we parse these documents into a data model. Each subsequent step in the methodology relies on the data model. Implementations where input documents are parsed as unstructured text will lack information like tabular or visual alignments, which are vital in determining whether a candidate is correct.

Candidate Extraction Recall from Section 2 that we define candidates as an n-ary tuple of mentions, each of which belong to a particular mention type. To extract candidates, we first define mention types for each of the mentions in the candidate, then we extract the cross product of all mentions of each type to form candidates. Because of this cross product, there can be a combinatorial explosion of candidates, most of which are false. To combat this class imbalance and improve performance, we apply filters at both the mention and candidate levels. For example, if a mention type is a numerical value, we can filter at the mention level by constraining mentions to numerical values within a specific range. At the candidate level, we can filter based on the candidate as a whole, e.g., discarding candidates in which all of its component mentions are not on the same page of the document. This highlights a fundamental tension between optimizing for system performance and optimizing for end-to-end quality. If we do not filter any candidates, there is an extreme class imbalance towards negative candidates that lowers end-to-end quality. Filtering improves performance by reducing the number of candidates considered and helps reduce the class imbalance. But, after a certain point, additional filtering lowers overall recall and subsequently, also decreases end-to-end quality.

Featurization Next, we featurize each of the extracted candidates using the default set of features provided by Fonduer [23]. Fonduer leverages the data model to compute features that capture signals from multiple modalities of information, such as structural, tabular, and visual features in addition to standard natural-language features such as part-of-speech and named-entity-recognition tags. It then creates a feature vector for each candidate indicating which of the features each candidate expresses. In our experience, we find that final end-to-end quality is best when features from all modalities are present.

3.3 Phase 3: Iterative Knowledge Base Construction

Finally, we use labeling functions to unify multiple sources of supervision, such as heuristics and human labels, which allow us to systematically capture domain expertise. These labeling functions are then used to generate training data used to train a machine-learning classifier. However, because each of these sources may have differing levels of quality, we iteratively refine them, and, in turn, also refine our training data, to achieve an acceptable quality of training data. We then train a discriminative model with this training data to generate final knowledge bases. With this approach, we have a classic classification problem and can apply logistic regression[2] for text-based relation extraction and a convolutional neural network for image-based relation extraction.

To aid in this process, we provide three best practices for developing labeling functions for hardware datasheets. First, use labeling functions that operate on multiple modalities of information. For example, do not rely on labeling functions that use tabular information alone to determine alignments; use visual alignment as well. Using multiple modalities helps leverage the redundant information in the underlying data, resulting in more robust supervision.

Second, class imbalance (where there are many more negative candidates than positive candidates) is a prevalent challenge. Because of this imbalance, labeling functions should output true-else-abstain, or false-else-abstain, and should not output true-else-false or vice versa. Labeling functions

[1] Prior work has explored different approaches for extracting subsets of this metadata directly from PDF documents [5, 13, 16], but challenges remain.

[2] Due to the high sparsity of the features, we use the sparse version of logistic regression to reduce memory usage during training.

Table 1. Summary of the datasets used in our evaluation based on their size, number of documents, average number of pages per document, and the number of relations extracted.

Dataset	Size	#Docs	#Pgs/Doc	#Rels
Bipolar Junction Transistors	3GB	7.0k	5.5	4
Circular Connectors	3GB	5.1k	3.2	1
Operational Amplifiers	5GB	3.3k	23.3	2

that do not abstain typically have large amounts of conflicts, which lowers the computed weight for that labeling function. Instead, repurpose accurate labeling functions that label true-else-false as filters during candidate extraction.

Third, when debugging and developing labeling functions, evaluate them on the development set, not the test set. Tune and refine the labeling functions by inspecting the true positive, false positive, and false negative candidates. Prefer labeling functions that compensate for class imbalance, and only include labeling functions that are accurate greater than 50 % of the time. Typically, fewer than 20 accurate labeling functions are sufficient to achieve high quality.

4 Evaluation

In this section, we evaluate our methodology and examine end-to-end quality and scalability. We perform application studies that illustrate how these datasets can be used to make hardware component selection easier.

4.1 Evaluation Setup

We evaluate our methodology using three distinct datasets: bipolar junction transistors, operational amplifiers, and circular connectors. We extract relations from each dataset.

4.1.1 Datasets

Table 1 shows a summary of our three datasets, primarily sourced from Digi-Key. All of the documents in each dataset are processed to evaluate end-to-end quality (Section 4.2.1). Datasheets were selected by downloading all of the PDF datasheets available on Digi-Key in the respective product category. Datasheets which were duplicates, corrupted (i.e., could not be processed by Adobe Acrobat), or required optical character recognition (OCR) were filtered out. These datasets represent immense data variety in terms of both format and style from many manufacturers, who used over 285 unique versions of software tools to author these datasheets, ranging from general purpose tools like Microsoft Word and OpenOffice, to more specialized tools like TopLeaf, QuarkXPress, and AutoCAD.

Transistors Transistors are one of the most commonly used and fundamental electrical components. Posts on selecting the correct transistor frequently appear on maker forums [1]. We select transistor datasheets from over 20 unique manufacturers and extract four binary relations primarily contained within tables: minimum and maximum storage temperatures, polarity, and maximum collector-emitter voltages, along with their associated part numbers. Our output is four database tables with the schema (document, part number, attribute value, probability). *We use this dataset to evaluate how our methodology performs using heuristics as a weak supervision source.*

Operational Amplifiers [10], required an operational amplifier with very specific characteristics. To find potential parts, Huang et al. scraped Digi-Key to explore the tradeoff between two electrical characteristics. In contrast, we generate that knowledge base using our machine-learning approach. Operational amplifiers are more complex and described by datasheets that are 4× longer on average than transistors. We assess datasheets from over 30 unique manufacturers and extract two unary relations, the gain bandwidth product and the quiescent current, in order to compare our result with that in [10]. Our output is two database tables with the schema (document, attribute value, probability). *We use this dataset to evaluate our methodology on using human labels as a weak supervision source.*

Circular Connectors Circular connectors, the third largest category of items on Digi-Key with over 490 000 products from 50 manufacturers, provides a diverse dataset. The sheer number of circular connectors makes it difficult to quickly find the right one, especially since the most important information about circular connectors are not numerical values, but how they look. To that end, we extract a single, non-textual, unary relation—thumbnail images—and output a database table with the schema (document, thumbnail, probability). *We use this dataset to evaluate our methodology when extracting non-textual information like images.*

4.1.2 Evaluation Metric

We evaluate the end quality of our knowledge bases using *precision*, *recall*, and *F1* score, defined as follows:

$$\texttt{precision} = \frac{tp}{tp + fp} \tag{1}$$

$$\texttt{recall} = \frac{tp}{tp + fn} \tag{2}$$

$$\texttt{F1} = 2 \times \frac{\texttt{precision} \times \texttt{recall}}{\texttt{precision} + \texttt{recall}} \tag{3}$$

where:

tp = True positives. How many candidates predicted to be positive are true.
fp = False positives. How many candidates predicted to be positive are false.
fn = False negatives. How many candidates predicted to be negative are true.

Table 2. End-to-end quality in term of precision, recall, and F1 score for each dataset.

Dataset	Relation	Prec.	Rec.	F1
Trans.	Min. Storage Temp.	1.00	0.58	0.74
	Max. Storage Temp.	0.95	0.61	0.74
	Polarity	0.88	0.92	0.90
	Max. Collector-Emitter Volt.	0.85	0.77	0.81
Op. Amps.	Gain Bandwidth Product	0.72	0.76	0.74
	Quiescent Current	0.65	0.54	0.59
Circ. Conn.	Product Thumbnails	0.63	0.83	0.72

4.1.3 Implementation Details

We implemented our approach in Python, using Fonduer 0.6.2, and PostgreSQL 9.6.9 for database operations. We used a machine with 4 physical CPUs (each of which was a 14-core 2.4 GHz Xeon E4–4657L), 1 TB of memory, 2 × NVIDIA GeForce TITAN X GPUs, 12 × 3 TB disk drives, and Ubuntu 16.04.3 as the operating system. We used Adobe Acrobat Pro to generate an HTML representation for each PDF document to support structural features. See Appendix A for details on how to access the datasets and our implementation online.

4.2 Evaluation Results

We perform several experiments to evaluate our methodology in terms of end-to-end quality and performance.

4.2.1 End-to-End Quality

Table 2 shows the precision, recall, and F1 score of each of the relations we extract from each dataset. We achieve, on average, 75 F1 points. Unlike large manually created knowledge bases that have been cultivated and curated for years, our knowledge bases were created in a matter of weeks. As expected, we achieve lower F1 scores for relations that are more complex. In the transistor dataset, for example, we achieve 90 F1 points for transistor polarities, which take one of two values: "NPN" or "PNP", and usually apply to all parts on a datasheet. However, our score for operational amplifier quiescent currents is only 59 F1 points. This is because quiescent current typically differs for each part listed in a datasheet and is often associated using visual alignments alone which makes them more sensitive to noise.

In addition, we find that using heuristics as a weak supervision source generally results in higher precision, but lower recall (as shown in the transistor dataset), while using human labels as a weak supervision source results in a relatively higher recall to precision ratio (as shown in the operational amplifier dataset). The reason for this is twofold. First, providing supervision using heuristics inherently targets specific patterns or features in the data (e.g., that two words are in the same tabular row). Consequently, the machine learning model learns a signal more precisely since the heuristic is applied systematically across the dataset, but may return lower recall since other signals are not directly considered. In contrast, providing human annotations for supervision inherently targets specific candidates, not patterns. As a result, human labels will typically cover a more broad set of features resulting in higher recall, yet potentially at the cost of lower precision (see Section 4.4).

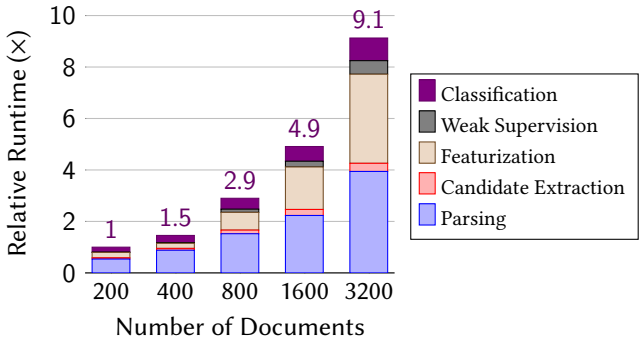


Figure 4. Relative end-to-end runtime for each computational stage when scaling the number of documents from the transistor dataset for a single relation extraction task.

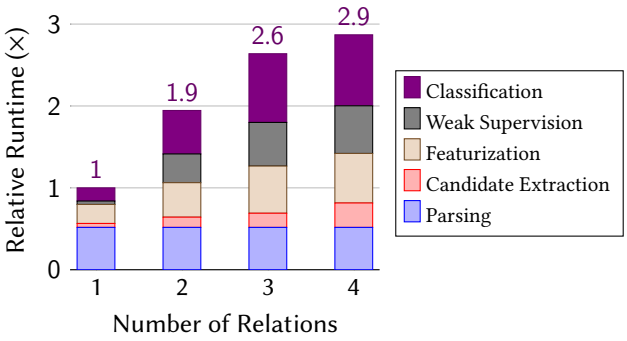


Figure 5. Relative end-to-end runtime for each computational stage when scaling the number of relations extracted from 1000 documents of the transistor dataset.

4.2.2 Scalability and Performance

We perform two experiments to evaluate the scalability and performance of our methodology. In Figures 4 and 5, we show examples of the end-to-end runtime that each computational step of our methodology requires. By rerunning weak supervision and classification alone, we incrementally refine our generated training data. This fine tuning allows amortization of the costs of parsing our corpus, extracting candidates, and featurizing those candidates.

Parsing scales with the number of input documents, while candidate extraction, featurization, weak supervision, and classification scales with the number of candidates. Figure 4 shows the relative end-to-end runtime when scaling the number of input documents from the transistor dataset. In

Table 3. Quality of our approach vs. Digi-Key for available relations compared to ground truth data.

Relation	Source	Prec.	Rec.	F1
Polarity	Digi-Key	**1.00**	0.67	0.80
	Our Approach	0.94	**0.94**	**0.94**
Max Collector-Emitter Volt.	Digi-Key	**0.97**	0.67	**0.79**
	Our Approach	0.75	**0.77**	0.76
Gain Bandwidth Product	Digi-Key	**0.91**	0.62	0.74
	Our Approach	0.88	**0.84**	**0.86**
Quiescent Current	Digi-Key	**0.93**	0.45	0.61
	Our Approach	0.89	**0.80**	**0.84**

Figure 5, we measure the end-to-end runtime of each computational step when increasing the number of relations extracted from 1000 documents of the transistor dataset. Increasing the number of relations or the number of documents parsed are proxies for increasing the number of candidates. From these figures, we see that our methodology scales sublinearly with documents and relations.

The end-to-end runtime across our datasets was on the order of 10s of hours; this allows us to generate hardware component knowledge bases in a matter of weeks. Our implementation has significant room for optimization to reduce the iteration time required to build these knowledge bases. Optimization efforts could lower system requirements below the thresholds currently needed to process large datasets.

4.2.3 The Benefits of a Machine Learning Approach

The process of manually creating large knowledge bases like Digi-Key is expensive and prone to human error. We compare our generated knowledge bases with Digi-key on four relations from a small set of transistor and operational amplifier datasheets using ground truth labeled by domain experts. Of the relations we extract, only these four relations are present in Digi-Key's database. Consequently, only these four relations can be directly compared.

On average, we improve on the quality of Digi-Key's knowledge base by 12 F1 points, primarily by improving recall by 24 % while only losing about 9 % in precision (Table 3). Digi-key outperforms our approach in terms of F1 score when extracting maximum collector-emitter voltages. This is primarily due to noise introduced during PDF parsing (see Section 5). For example, in Figure 6, some PDF parsers may ignore vertical alignments for the cells boxed in blue, and instead collapse all those values into a single sentence per cell. This results in inaccurate structural information, which makes it challenging to correctly associate part numbers with their attribute values.

By inspecting these discrepancies we find that errors in Digi-Key data for these relations fall into three categories.

Symbol	Rating		Rating	Unit
V_{CEO}	Collector-Emitter Voltage	TIP29, TIP30	40	V
		TIP29A, TIP30A	60	
V_{CBO}	Collector-Base Voltage	TIP29B, TIP30B	80	
		TIP29C, TIP30C	100	

Figure 6. Non-textual signals like alignments are vital in associating parts and attribute values.

Parameter	Symbol	Limit	Unit
Collector-base voltage	V_{CBO}	-160	V
Collector-emitter voltage	$V_{(BR)CEX}$	-160	V
Collector-emitter voltage	V_{CEO}	-145	V

Figure 7. Ambiguous datasheets lead to human errors.

1. Recall: In 66 % of these discrepancies, Digi-Key only extracts a subset of the parts or values described in the datasheet. For example, a datasheet may express multiple valid gain values based on how an amplifier is configured, but Digi-Key only extracts one of multiple valid gain values for each amplifier.
2. Neglecting hierarchy: In 29 % of discrepancies, Digi-Key ignores part family information. For example, failing to relate a value to a part family as a whole (e.g., BC546), when all the children of that part family (e.g., BC546A, BC546B, BC546C) share a value.
3. Inconsistent interpretation: 5 % of the discrepancies occur because Digi-Key interprets a value inconsistently. For example, both $V_{(BR)CEX}$ and V_{CEO}, can be generally referred to as a "collector-emitter voltage". Ambiguity may cause human annotators to unintentionally extract the wrong value (Figure 7).

Importantly, these error classes are not systematic; they do not follow a regular, consistently applied pattern. Further, these error classes can also vary depending on the individual inputting data. In contrast, a machine-learning approach shifts the class of errors from random to systematic, which can be readily identified and reduced.

Using information posted on websites like Digi-Key is common practice, but this study also highlights how supplier summaries are limited when used as a hardware component knowledge base. Specifically, supplier summaries are, often by design, not exhaustive. Instead, these supplier summaries focus on the components carried and sold by the supplier, and may only represent a biased fraction of all available components. While these summaries typically maintain very high precision, this selectivity also significantly limits their recall and, consequently, their use as a general hardware component knowledge base.

4.3 Application Studies

In this section, we study two example applications powered by our hardware component knowledge bases, and demonstrate how these machine-generated knowledge bases make hardware component selection easier.

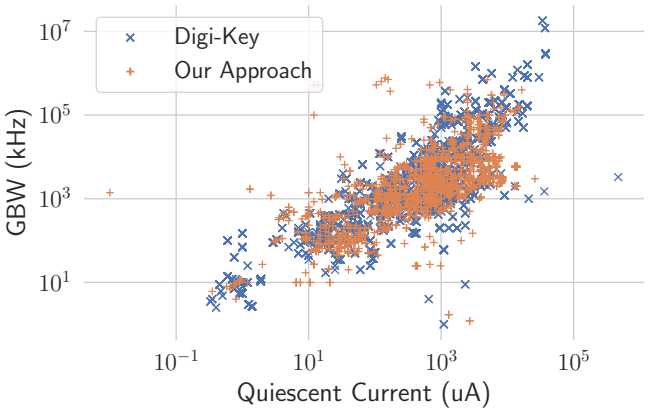


Figure 8. We recreate the figure in [10] using our machine-learning approach and find that we largely overlap with the human-curated data published by Digi-Key.

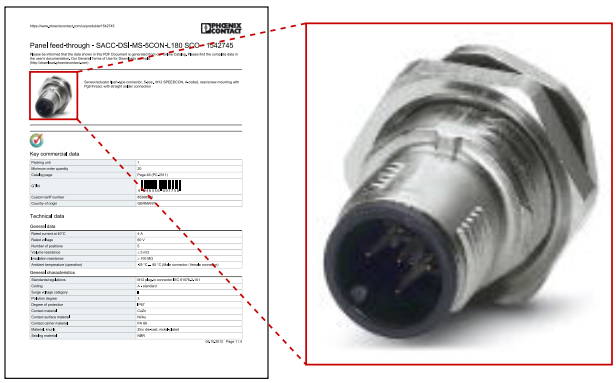


Figure 9. An example of a circular connector datasheet that contains an image of a product thumbnail.

4.3.1 Electrical Characteristic Analysis

Analyzing electrical characteristics is a key part of the process of selecting hardware components. [10] found that the key to their sensor device was the capacity to detect an ultrasonic signal reliably and accurately within a constrained power budget. To accomplish this, they needed a series of operational amplifiers that could provide 1000 × gain, and were highly motivated to minimize the number of operational amplifiers used in their circuit in order to minimize noise and physical size.

To aid their search, they performed a survey of operational amplifiers by scraping information from Digi-Key, and plotted the gain bandwidth product against the quiescent supply current of each amplifier. Their data is shown in Figure 8 in blue. We extract these same two electrical characteristics from our dataset of operational amplifier datasheets. We filter each characteristic independently using a threshold of 0.75. Then we combine the results based on their reading-order appearance in their datasheets and plot our results in orange. Our plot contains fewer data points because we only extract data from a subset of the operational amplifier documents on Digi-Key.

We largely overlap with the data published by Digi-Key. Based on a sample of the outliers, we find that these outliers result from both errors in Digi-Key's database (e.g., marking a value as kHz rather than MHz) and errors in our output (e.g., misclassifying an erroneous current value as quiescent current). However, with a machine-learning approach, these errors are more systematic in nature, and so can be more readily identified and corrected. [10] ultimately selected a Micrel MIC861/863, which has a gain of 400 kHz and a quiescent current of 4.6 µA. Using our dataset, we are able to identify—with the correct gain bandwidth product and quiescent current values—the same families of amplifiers as potential candidates. While both Digi-key and our database are imperfect, this first-order analysis suggests that our approach can generate queryable knowledge bases at scale and with human-like quality, which can serve as a powerful foundation for analysis tools. Our approach can also be applied to new domains, where existing databases may not exist.

4.3.2 Enabling Multimodal Search

With over 490 000 products listed, circular connectors are the third largest product category on Digi-Key. Unlike transistors or operational amplifiers, one of the most important characteristics of circular connectors is their appearance. Without hardware component knowledge bases, the process of finding a compatible connector would require searching for and tediously inspecting the contents of each datasheet.

We extend Fonduer to go beyond traditional text extraction and demonstrate that our approach can be applied to extract non-textual information like images of product thumbnails (Figure 9). We use a convolutional neural network to extract signals from pixel-level information to identify and classify product thumbnails directly from datasheets. More specifically, we select a pre-trained ResNet-18 network provided by `torchvision`[3] and refine its weights based on our dataset. We achieve 72 F1 points on this task and produce a database of product thumbnails that can be used to make component selection easier. We argue that our methodology can be applied with similar effectiveness to both textual information and images; further, it may prove to be effectively extended to additional information modalities.

4.4 Discussion

Traditionally, machine-learning systems rely on large amounts of manually labeled data. In response, techniques like weak supervision, which typically use heuristics and human annotations to programmatically generate training data, have risen in popularity.

[3] https://github.com/pytorch/vision

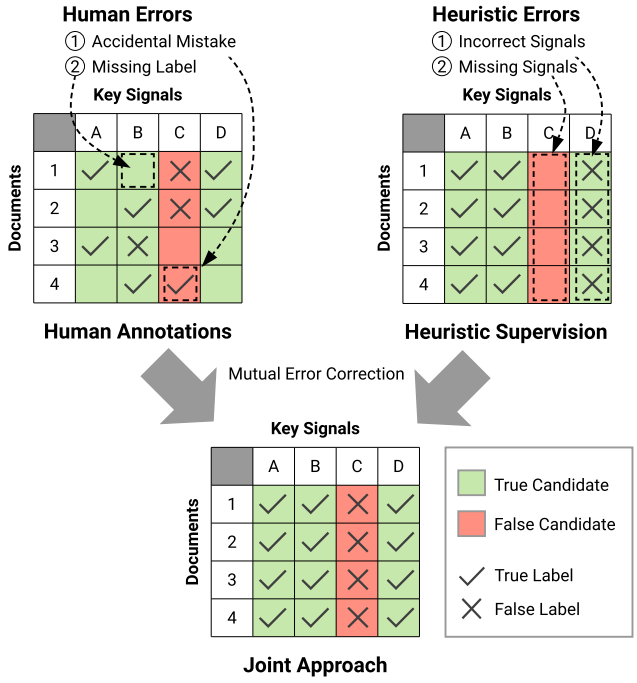
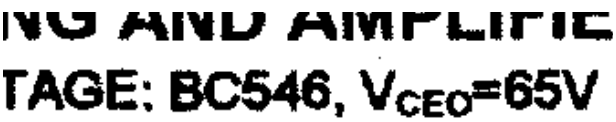
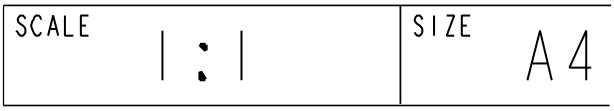


Figure 10. Our methodology benefits from using both the recall of human annotations and the systematic consistency and precision of a heuristics-based weak supervision.



(a) Scanned documents



(b) Vector-drawn text

Parameter	Conditions	AD620A Min	Typ	Max
Common-Mode Rejection				
1 kΩ Source Imbalance	V_{CM} = 0 V to ±10 V			
G = 1		73	90	
OUTPUT				
Output Swing	R_L = 10 kΩ			
	V_S = ±2.3 V to ±5 V	–V_S + 1.1		+V_S – 1.2
DYNAMIC RESPONSE				
Small Signal –3 dB Bandwidth				
G = 1			1000	

(c) Breaking cell boundaries

Figure 11. Real-world examples of formatting challenges that the techniques of our methodology do not address.

From our experiments, we observe that heuristic-based supervision and human annotations have distinct characteristics that give rise to unique advantages. Human annotators operate in the context of candidates; they identify specific instances of true or false candidates. This provides sparse, but precise information that is agnostic to the underlying features or patterns of the candidates. In contrast, heuristic-based supervision operates in the context of patterns. For example, entire subsets of candidates might be labeled true or false based on a pattern they share. As a result, these heuristics operate precisely based on the underlying features.

These characteristics are shown in Figure 10. In this figure, each square represents a candidate. Each candidate in a document has key signals, or features, that correspond to underlying patterns associated with the candidate. Human annotations provide true and false labels for individual candidates and often cover a wide range of signals that a machine-learning model can learn from. In contrast, heuristics are applied strictly based on key signals and only label those exact signals with precision.

In support of this intuition, we find that the heuristic supervision used in the transistor dataset results in higher precision, while the human annotations used in the operational amplifier dataset results in higher recall (Table 2). In light of these observations, we propose that a joint approach leveraging the benefits of both supervision from heuristics and supervision from human annotations successfully incorporates the advantages of each individual approach. For example, when certain candidates are difficult to label heuristically, even a small amount of precise human annotations can provide sufficient signal for a machine-learning model to learn from and apply systematically.

5 Future Work

Our methodology is effective for building hardware component knowledge bases of relational information, such as electrical characteristics and their values listed in datasheets. However, several important limitations remain which we highlight below to guide future work.

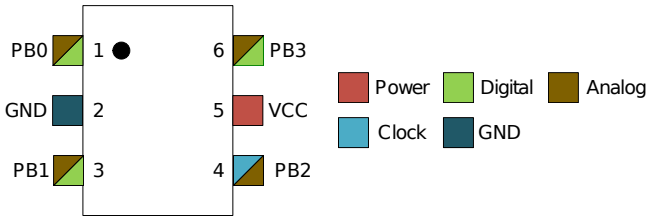
5.1 Parsing PDF Documents

PDF documents are the de facto standard for publishing hardware component information. As the primary input, any noise or errors introduced during the PDF parsing process propagate through the rest of the pipeline and negatively affect quality. In the context of datasheets, PDF parsing tools do not fare well in at least three challenging scenarios.

First, PDF documents consisting of scanned images requiring OCR (e.g., Figure 11a) introduce noise that is difficult to eliminate downstream. For example, OCR software may interpret a scanned document containing the text "`50 °C`" as "`50 OC`", "`500C`", or even as "`5000`" depending on the quality of the original scan and the quality of the OCR software.

CLASSIFICATION		A	B
h_{FE}	BC856	125~250	220~475
	BC857	125~250	220~475
	BC858	125~250	220~475

(a) Relationships to specific parts are implied by column headers alone. This example table is specifying that BC856A, BC857A, and BC858A have an h_{FE} of 125~250, while those with a B suffix have a value of 220~475.



(b) Relationships may also be specified using color matching.

Figure 12. Real-world examples of implicit relationships not addressed by these techniques.

Second, because PDF merely specifies characters, vectors, or images and the locations to render them, even native-digital PDF documents that include text as vectors rather than characters can cause OCR issues. For example, some manufacturers publish datasheets where, rather than inputting text as characters, they render text as vectors, resulting in documents that contain little to no text (Figure 11b).

Third, manufacturers author datasheets using a gamut of software tools and design them to be understood visually by readers. Recall that our methodology leverages structural information, such as tabular alignment, in a document. To extract this metadata, we pair PDF documents with a more structured format like HTML. When datasheets break the assumptions of common PDF parsing tools, such as allowing content to cross cell borders (Figure 11c), parsers introduce additional errors which degrade the resultant data quality.

5.2 Understanding Implicit Relationships

A further challenge left for future work is understanding information that is implicitly expressed in a document. For example, a document header may contain "BC546…547A/B", rather than explicitly listing "BC546A, BC546B, BC547A, BC547B" as part numbers. In this case, we must implicitly understand how to expand and associate these part numbers and suffixes. Some datasheets also use these suffixes in isolation to reference a family of part numbers (Figure 12a). This challenge is exacerbated when relationships are expressed using color coding or symbolic legends (Figure 12b).

5.3 Tooling and Performance

Tooling and implementation performance are additional areas ripe for future work. The process of manually labeling gold data for the development and testing sets is a tedious process that could be greatly optimized with improved tooling. For example, in contrast to the common workflow of users having to enter information into a spreadsheet, tools could reduce the tediousness of manual labeling by presenting users candidates and allowing users to directly mark the candidates true or false.

Also, due to architectural limitations, our implementation has performance bottlenecks (Section 4). The current architecture of our implementation scales vertically; this requires use of a machine with computing power and memory sufficient to run large datasets. Using a distributed architecture such as Ray [15] would allow our approach to scale horizontally and leverage modern cloud computing platforms.

5.4 Open Information Extraction

Our methodology extracts precise, pre-defined relations from the corpus of documents. This requires the user to explicitly define relations to extract and create new labeling functions and gold data for each relation. This process scales linearly with the number of target relations. In response, researchers have proposed techniques in *open information extraction* for extracting large sets of relations without requiring predefined specifications [3, 4, 8]. However, these techniques do not yet support richly formatted data such as hardware component datasheets.

6 Conclusion

Embedded system design productivity benefits from hardware component information that is both available and accessible. Unfortunately, troves of hardware component information is inaccessibly locked away in datasheets. We present a general methodology for automating the generation of queryable hardware component knowledge bases directly from their PDF datasheets. We use state-of-the-art machine-learning techniques based on weak supervision to overcome some of the known challenges of extracting relational information from richly formatted datasheets.

Our approach leverage domain expertise from both heuristics and human labels. Utilizing weak supervision, we combine benefits from the sparse but accurate signals of human annotations with the precise and systematic application of heuristics to yield a more robust and effective method of generating knowledge bases.

We evaluate our methodology by applying it to a dataset of over 15 000 PDF datasheets for transistors, operational amplifiers, and circular connectors. We extract multiple relations and multiple modalities of information from these datasheets such as numerical values from tables, text from paragraph descriptions, and images of product thumbnails, achieving an average of 75 F1 points. On average, we improve recall by 24 % at a cost of 9 % in precision and find that our methodology improves on existing human-curated knowledge bases by 12 F1 points.

A Artifact Appendix

A.1 Abstract

This artifact provides a Python package and datasets which can be used to replicate our methodology for automating the generation of hardware component knowledge bases. We describe how the software and datasets can be obtained, how the software and dependencies can be installed, and how the software can be used to run our experiments. Our artifact outlines the workflow from the input data (PDF and HTML documents, along with gold labels) to the final quality metrics we used to evaluate the approach. We also include scripts used for our analysis and performance experiments.

A.2 Artifact Checklist (Meta-Information)

- **Program:** We provide several Python and bash scripts.
- **Data set:** PDF/HTML datasets for transistor, operational amplifier, and circular connector datasheets provided. See Table 1 for descriptions (approx. 3.5 GB compressed).
- **Run-time environment:** We evaluated on Ubuntu 16.04.3, using Python 3.6 and PostgreSQL 9.6.9. We use Fonduer 0.6.2 as a weak supervision framework.
- **Hardware:** We used a machine with 4 physical CPUs (each of which was a 14-core 2.4 GHz Xeon E4–4657L), 1 TB of memory, 2 × NVIDIA GeForce TITAN X GPUs, 12 × 3 TB disk drives. However, a modern consumer-level machine without a GPU can also be used to validate results by scaling down the size of the dataset.
- **Execution:** Python programs with command-line interfaces.
- **Metrics:** Quality (F1 score, precision, recall) and runtime.
- **Output:** PostgreSQL databases and log files containing timings and metrics. Database information is also exported to comma-separated values (CSV) files for convenience in analysis and visualizations.
- **Experiments:** End-to-end quality, runtime when scaling number of documents and number of relations.
- **How much disk space required (approximately)?:** 50 GB for all software, full datasets, and full databases. 12 GB for just the software and raw, uncompressed datasets.
- **How much time is needed to prepare workflow (approximately)?:** 30 min.
- **How much time is needed to complete experiments (approximately)?:** Depends on hardware and the portion of the dataset used. Approximately 3 d for the full datasets of all components, however results can be validated on smaller datasets taking on the order of 10 h.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** MIT License.
- **Data licenses (if publicly available)?:** Creative Commons Attribution Non Commercial Share Alike 4.0 International.
- **Workflow framework used?:** No CK framework used. Used manual execution of scripts.
- **Archived (provide DOI)?:**
  - Dataset DOI: https://doi.org/10.5281/zenodo.2647543.
  - Software DOI: https://doi.org/10.5281/zenodo.2647667

A.3 Description

A.3.1 How Delivered

Our software (source code, documentation, and scripts) are available on GitHub: https://github.com/lukehsiao/lctes-p27.

A.3.2 Hardware Dependencies

We recommend testing on a server with 16 or more cores, 64 GB or more of memory, and an NVIDIA GPU. We used a machine with 4 physical CPUs (each of which was a 14-core 2.4 GHz Xeon E4–4657L), 1 TB of memory, 2 × NVIDIA GeForce TITAN X GPUs, 12 × 3 TB disk drives. However, a high-end, consumer-grade machine (e.g., quad-core, 32 GB of memory, no GPU) is also sufficient to validate our experiments in a reasonable time on a subset of the datasets.

A.3.3 Software Dependencies

We recommend Ubuntu 16.04.3, Python 3.6, Fonduer 0.6.2, PostgreSQL 9.6.9, and Poppler Utilities 0.36.0 or greater. We have only tested our software on an x86–64 system and our instructions assume these minimum versions.

A.3.4 Datasets

Our datasets are publicly available and archived using Zenodo. Our software includes helper scripts to download and unpack these datasets for each hardware component. You must navigate to each subcomponent directory before running the script. For example, start in the root of the repository and run the following to download the transistor dataset:

```
$ cd hack/transistors/
$ ./download_data.sh
```

Each dataset is already divided into a training, development, and testing set. Manually annotated gold labels are provided in CSV form for the development and testing sets in the software repository.

A.4 Installation

Assuming you are using Ubuntu 16.04 or greater, you can install all system dependencies by running the following.

```
$ sudo apt install build-essential curl poppler-utils
    ↪ postgresql postgresql-contrib virtualenv libxml2
    ↪ -dev libxslt-dev python3-dev
$ sudo apt build-dep python-matplotlib
```

You can then clone the repository by running:

```
$ git clone https://github.com/lukehsiao/lctes-p27.git
```

Then use a Python virtual environment and the provided Makefile to install the Python package and dependences which can be used to generate hardware component knowledge bases. From within the root of the repository, run:

```
$ virtualenv -p python3 .venv
$ source .venv/bin/activate
$ make dev
```

Finally, we recommend you configure a new PostgreSQL user, `demo`, with a password of `demo`, and create a database for each hardware component.

```
$ psql -c "create user demo with password 'demo'
    ↪ superuser createdb;" -U postgres
$ psql -c "create database transistors with owner demo;"
    ↪  -U postgres
$ psql -c "create database opamps with owner demo;" -U
    ↪ postgres
$ psql -c "create database circular_connectors with
    ↪ owner demo;" -U postgres
```

For a more detailed description, see https://github.com/lukehsiao/lctes-p27.

A.5 Experiment Workflow

Ensure that all datasets have been downloaded, installation is complete, and PostgreSQL databases have been created. Our experiments can be broken into three components: (1) end-to-end knowledge base construction, (2) scaling experiments, and (3) analysis and plotting. Because the analysis and plotting scripts rely on the output of knowledge bases, for convenience, we use intermediate files in CSV format and provide these intermediate CSV results from a run on the full dataset.

End-to-end Knowledge Base Construction After completing the installation steps, three command-line programs should be available in your virtual environment: `transistors`, `circular_connectors`, and `opamps`. These programs can be used to run end-to-end knowledge base construction on their respective hardware datasets.

Scaling Experiments In the `scripts/` directory, we provide `scaling_docs.sh` and `scaling_rels.sh`, which can be run to generate the runtime logs used to create Figures 4 and 5. These scripts should be customized for your machine (e.g., whether or not to utilize a GPU).

Analysis and Plotting Next, we provide the `analysis` program for transistors and operational amplifiers which can be used to compare the results of our approach with Digi-Key (Section 4.2.3 and Table 3). Running the command below will output two sets of scores for each relation: one for our generated knowledge base, and one for Digi-Key's knowledge base, compared against manually-collected ground truth data.

```
$ analysis --ce-v-max --polarity --gain --current
```

Finally, we provide `plot_opo.py` in the `scripts/` directory which can be used to generate Figure 8. To do so, this script leverages the intermediate CSV files output from the `opamps` program. Because we include intermediate files from a full run, this plot can be generated without running end-to-end knowledge base construction using our results, or after running `opamps` end-to-end to use the results of the most recent run.

A.6 Evaluation and Expected Result

End-to-end Knowledge Base Construction Each of the three programs will produce log files containing the final quality metrics of that run for each relation extracted (see example below).

```
Scoring on Entity-Level Gold Data w/ b=0.245
==================================================
Corpus Precision 0.742
Corpus Recall 0.807
Corpus F1 0.773
```

These scores on the full datasets of each hardware component are the scores entered in Table 2.

Scaling Experiments These same log files also include runtimes of each computational step in our methodology (see example below).

```
...
Candidate Extraction Time (min): 0.3
Featurization Time (min): 25.2
...
```

We aggregate the runtimes of each phase for each run (i.e., while scaling up the number of documents or relations extracted) and plot the results in Figures 4 and 5.

Analysis and Plotting The `analysis` program will print scores to the terminal, which are used to populate Table 3. In addition, this program will output a set of CSV files that can be used to manually inspect discrepancies in the knowledge bases (Section 4.2.3). See the README included in the repository for more details.

The output of `plot_opo.py` is a PDF figure named `opo.pdf`, which is included directly as Figure 8.

A.7 Experiment Customization

While the source code is entirely available, and users can freely modify any part of the code (e.g., labeling functions), we expose a few command-line parameters for convenience in running experiments. These parameters include the number of documents from the training set to use, the amount of parallelization to use, whether or not to utilize a GPU during training, and which relations of each dataset to extract. The full set of options for each program can be viewed by using the `-h` flag. In addition the `plot_opo.py` script also allows a user to customize the threshold values used (we default to 0.75, see Section 4.3.1).

A.8 Methodology

Submission, reviewing and badging methodology:

- http://cTuning.org/ae/submission-20190109.html
- http://cTuning.org/ae/reviewing-20190109.html
- https://www.acm.org/publications/policies/artifact-review-badging

Acknowledgments

We gratefully acknowledge the support of DARPA under Nos. FA87501720095 (D3M) and FA86501827865 (SDH), NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF1763315 (Beyond Sparsity) and CCF1563078 (Volume to Velocity), ONR under No. N000141712266 (Unifying Weak Supervision), the Moore Foundation, NXP, Xilinx, LETI-CEA, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, the Okawa Foundation, and American Family Insurance, and members of the Stanford DAWN project: Intel, Microsoft, Teradata, Facebook, Google, Ant Financial, NEC, SAP, VMware, and Infosys. We also acknowledge the support of the Intel/NSF CPS Security grant No. 1505728, the Stanford Secure Internet of Things Project, and the Stanford System X Alliance. Finally, we thank Mark Horowitz and Björn Hartmann for their feedback on early versions of this work. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of DARPA, NIH, ONR, or the U.S. Government.

References

[1] 2015. Choosing the right transistor for a switching circuit. https://electronics.stackexchange.com/questions/29029/choosing-the-right-transistor-for-a-switching-circuit

[2] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 331–342.

[3] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, Vol. 1. 344–354.

[4] Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. 2007. Open information extraction from the web.. In *IJCAI*, Vol. 7. 2670–2676.

[5] Hui Chao and Jian Fan. 2004. Layout and content extraction for pdf documents. In *International Workshop on Document Analysis Systems*. Springer, 213–224.

[6] Dave Doherty. 2019. About Digikey. https://www.digikey.com/en/resources/about-digikey

[7] Daniel Drew, Julie L Newcomb, William McGrath, Filip Maksimovic, David Mellis, and Björn Hartmann. 2016. The toastboard: Ubiquitous instrumentation and automated checking of breadboarded circuits. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 677–686.

[8] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, et al. 2011. Open information extraction: The second generation. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[9] Benoît Frénay and Michel Verleysen. 2014. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems* 25, 5 (2014), 845–869.

[10] William Huang, Ye-Sheng Kuo, Pat Pannuto, and Prabal Dutta. 2014. Opo: a wearable sensor for capturing high-fidelity face-to-face interactions. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*. ACM, 61–75.

[11] Antonio Iannopollo, Stavros Tripakis, and Alberto Sangiovanni-Vincentelli. 2019. Constrained synthesis from component libraries. *Science of Computer Programming* 171 (2019), 21–41.

[12] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. 2015. Comprehensive and reliable crowd assessment algorithms. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 195–206.

[13] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. 2007. Tableseer: automatic table metadata extraction and searching in digital libraries. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 91–100.

[14] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics, 1003–1011.

[15] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 561–577.

[16] Ermelinda Oro and Massimo Ruffolo. 2009. Trex: An approach for recognizing and extracting tables from pdf documents. In *2009 10th International Conference on Document Analysis and Recognition*. IEEE, 906–910.

[17] Shanan E Peters, Ce Zhang, Miron Livny, and Christopher Ré. 2014. A machine reading system for assembling synthetic paleontological databases. *PLoS one* 9, 12 (2014), e113523.

[18] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. Retrofab: A design tool for retrofitting physical interfaces using actuators, sensors and 3d printing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 409–419.

[19] Raf Ramakers, Kashyap Todi, and Kris Luyten. 2015. PaperPulse: an integrated approach for embedding electronics in paper designs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2457–2466.

[20] Rohit Ramesh, Richard Lin, Antonio Iannopollo, Alberto Sangiovanni-Vincentelli, Björn Hartmann, and Prabal Dutta. 2017. Turning coders into makers: the promise of embedded design generation. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*. ACM, 4.

[21] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment* 11, 3 (2017), 269–282.

[22] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data Programming: Creating Large Training Sets, Quickly. In *Advances in Neural Information Processing Systems*. 3567–3575.

[23] Sen Wu, Luke Hsiao, Xiao Cheng, Braden Hancock, Theodoros Rekatsinas, Philip Levis, and Christopher Ré. 2018. Fonduer: Knowledge Base Construction from Richly Formatted Data. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 1301–1316.

[24] Ce Zhang, Vidhya Govindaraju, Jackson Borchardt, Tim Foltz, Christopher Ré, and Shanan Peters. 2013. GeoDeepDive: statistical inference using familiar data-processing languages. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 993–996.

[25] Yuchen Zhang, Xi Chen, Dengyong Zhou, and Michael I Jordan. 2014. Spectral methods meet EM: A provably optimal algorithm for crowdsourcing. In *Advances in neural information processing systems*. 1260–1268.