Doctopus: Budget-aware Structural Table Extraction from Unstructured Documents

Chengliang Chai
ccl@bit.edu.cn
Beijing Institute of Technology
China

Jiajun Li
lijiajun@bit.edu.cn
Beijing Institute of Technology
China

Yuhao Deng
dyh18@bit.edu.cn
Beijing Institute of Technology
China

Yuanhao Zhong
zyh04@bit.edu.cn
Beijing Institute of Technology
China

Ye Yuan
Guoren Wang
yuan-ye@bit.edu.cn
wanggr@bit.edu.cn
Beijing Institute of Technology
China

Lei Cao
caolei@arizona.edu
University of Arizona
United States

ABSTRACT

To fulfill the potential great value of unstructured documents, it is critical to extract structural data (e.g., attributes) from them, which can benefit various applications such as analytical SQL queries and decision-making. Multiple strategies, such as pre-trained language models (PLMs), can be employed for this task. However, these methods often struggle to achieve high-quality results, particularly when dealing with attribute extraction that requires intricate reasoning or semantic comprehension. Recently, large language models (LLMs) have proven to be effective in extracting attributes but incur substantial costs caused by token consumption, making them impractical for large-scale document set.

To best trade off quality and cost, we present Doctopus, a system designed for accurate attribute extraction from unstructured documents with a user-specified cost constraint. Overall, Doctopus combines LLMs with non-LLM strategies to achieve a good trade-off. First, the system employs an index-based approach to efficiently identify and process only relevant text chunks, thereby reducing the LLM cost. Afterwards, it further estimates the quality of multiple strategies for each attribute. Finally, based on the cost and estimated quality, Doctopus dynamically selects the optimal strategies through budget-aware optimization. We have built a comprehensive benchmark including 4 document sets with various characteristics and manually labeled ground truth using 1000 human hours. Extensive experiments on the benchmark show that compared with state-of-the-art baselines, Doctopus can improve the quality by 11% given the same cost constraint.

PVLDB Reference Format:
Chengliang Chai, Jiajun Li, Yuhao Deng, Yuanhao Zhong, Ye Yuan, Guoren Wang, and Lei Cao. Doctopus: Budget-aware Structural Table Extraction from Unstructured Documents. PVLDB, 18(11): 3695 - 3707, 2025.
doi:10.14778/3749646.3749647
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/mutong184/Doctopus.

*Ye Yuan and Guoren Wang are the corresponding authors.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749647

1 Introduction

Modern corporations host a large amount of unstructured data, comprising 80%-90% of all data. To gain valuable insights, many applications often have to convert collections of unstructured documents into structured relational tables and then run analytical SQL queries or machine learning predictions. However, the complexity and diversity of documents pose significant challenges in the process of extracting the attribute values of these tables.

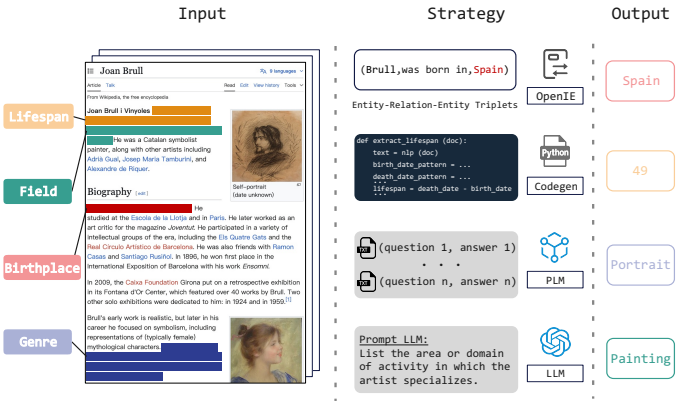

Figure 1: An Example of Attribute Extraction.

Existing Solutions. In Figure 1, a user aims to extract attributes from documents w.r.t. a set of painters, such as their personal information and artistic Genre, which can be solved with different extraction approaches such as open information extraction (OpenIE) [18], pre-trained language models (PLMs) [31], code generation (Codegen) [6], and LLMs. OpenIE extracts triplets (represented as subject, relation, object) from unstructured documents, which are then converted into structural tables where subjects represent rows, relations represent columns, and objects correspond to cell values. Codegen is an automated data extraction strategy that leverages

LLMs to generate extraction codes for documents. However, their effectiveness varies significantly w.r.t. different application scenarios. For example, code generation via LLMs is better than PLMs at extracting the `Lifespan` attribute, as it requires logical reasoning (i.e., `Lifespan` equals to the `death_date` minus `birth_date`) – a better fit to the LLM-generated code than the PLMs which often suffer from restricted reasoning abilities. However, for the `Genre` attribute, the code might mistakenly identify `Realistic` as the value, as seen in the text: "Brull's early work is realistic, but later he focused on symbolism." PLMs instead tend to perform well in this scenario due to their semantic comprehension abilities. In conclusion, no single approach above is consistently superior.

LLMs, on the other hand, excel in converting unstructured data into structured formats [6], thus generally delivering high-quality results. However, it is prohibitively expensive if feeding every entire document into the LLMs, because the commercial LLM services charge by the number of tokens. Unfortunately, real applications typically have cost budget constraint, and thereby it is not practical to always use expensive LLMs to extract attributes. In this work, our objective is therefore to solve an important problem, namely given a *budget constraint*, producing an execution plan that optimizes the overall extraction quality.

We solve this problem based on two key ideas: (1) designing an efficient index (or retrieval strategy) to avoid feeding the entire document to LLMs; and (2) applying different strategies appropriate to different cases, i.e., when extracting different attributes from different documents. The key observations supporting the ideas are (1) feeding only attributes-relevant chunks of text can significantly reduce the LLM cost without sacrificing quality much; (2) the cost-effective extraction approaches (e.g., Codegen) could be as good as or even better than expensive LLMs in some cases, such as extracting the value of `Lifespan`.

Challenges. To achieve this, we face several major challenges. First, since our goal is to maximize the overall extraction quality considering different strategies, it is critical to estimate the quality of each strategy in different scenarios. To this end, the most straightforward way is to incorporate a validation set to evaluate the quality. However, typically there are only a limited number of validation examples available in real applications, which tend to be insufficient to reliably estimate the quality of different strategies over different documents. On the other hand, obtaining a large number of *pseudo* validation examples by applying LLMs to examine a large number of documents is often not acceptable due to the significant LLM costs. Therefore, it is challenging to accurately estimate the quality case by case based on only a small validation set.

Second, when using LLMs, identifying relevant chunks of a given attribute is also challenging because (1) the representations of the attribute values might vary across different unstructured documents due to their complicated semantics; For example, the `Lifespan` may have various representations in different documents like "His life was cut short at the age of ...", "He lived an extraordinary life spanning ...", etc.; and (2) the query used to retrieve (*i.e.*, simply embedding an attribute and its brief description) is often not informative enough to capture all relevant chunks via the similarities between the embeddings of a query and a chunk. For example, using "Genre" and its description to query LLMs may miss relevant chunks like "He was known for his many oil paints, the majority of which were exhibited at the Miguel Lerdo de Tejada Library", which do not explicitly mention the attribute value "Painting". This makes the embeddings of relevant chunks not align well with the query embedding.

Our Proposal. To address the above challenges, we propose a novel system, `Doctopus`, which optimizes extraction accuracy (i.e., quality) under the constraint of a cost budget (i.e., number of tokens consumed by LLMs). The key idea is to make good use of LLM-based extraction strategies, as well as judiciously incorporating non-LLM strategies to save cost without sacrificing much quality.

To be specific, `Doctopus` first builds an index over chunks of documents and enriches each query to accurately retrieve the relevant chunks. However, given an attribute to be extracted, `Doctopus` is likely to return multiple relevant chunks from a document. If we only presented the most relevant one to an LLM for extraction, it would save LLM cost but risk missing the real match, while feeding all chunks to the LLM is costly. Hence, given all retrieved chunks, `Doctopus` instead selects a subset of chunks and presents them to LLMs as a combination. Each combination corresponds to a possible LLM-based strategy. Then from all optional strategies, LLM-based or not, `Doctopus` selects the most appropriate strategy that best trades off accuracy and cost. `Doctopus` conducts the selection at a fine granularity, i.e. using different strategies with respect to different documents to extract an attribute. Consequently, during this process `Doctopus` has to estimate the accuracy of optional strategies at the individual document level. This approach thus consists of three main components: *index-based attribution extraction*, *quality estimation for different strategies*, and *strategy selection with budget constraint*.

Attribute Extraction via Index. `Doctopus` first segments each document into semantically coherent chunks and encodes them into embedding vectors to well capture their semantics. Then, we index these chunks of documents for retrieval, and only feed relevant chunks to LLMs. This strategy, inspired by retrieval augmented generation (RAG), effectively avoids unnecessary LLM costs incurred on irrelevant text chunks.

Moreover, to address the challenge that a query is often not informative enough to accurately retrieve relevant chunks, we leverage LLMs as well as a validation set to generate attribute-related reference sentences to enrich the query. After summarizing these references, `Doctopus` embeds and concatenates them as the final query embedding to retrieve relevant chunks, increasing the likelihood of discovering chunks containing the attribute value.

Quality Estimation for Different Strategies. `Doctopus` represents the quality of each strategy as the product of two probabilities: (1) the probability of accurately identifying whether or not the input text of a strategy has an attribute, *i.e.*, the containment probability; (2) if so, the probability of accurately extracting it, *i.e.*, extraction probability. Estimating the containment probability is particularly challenging due to the varied semantics across different inputs, such as different chunks or the whole documents. Consequently, the overall probability calculated from a limited number of validation examples often are significantly off from the real accuracy on each individual input. As for extraction probability, we observe that it tends to be stable over different inputs, as long as these

inputs indeed contain the target attribute. Therefore, it could be estimated by the average calculated from the validation examples. To precisely estimate the containment probability of different inputs, we propose to train a generic model that produces a specific probability with respect to any given input. Our key insight here is that the DistilBERT model [28], a lightweight model broadly adopted in information retrieval tasks [10, 14, 22, 32], is able to precisely estimate this probability after fine-tuning with a small number of training examples. We develop data augmentation techniques like sentence reordering/removal and chunk combination, to automatically produce sufficient training examples to fine-tune the DistilBERT model.

Strategy Selection with Budget Constraint. Based on the cost and estimated accuracy of each strategy, the problem of selecting the most appropriate extraction strategy can be formulated as an NP-hard optimization problem (i.e., Group Knapsack Problem): selecting the best strategy for each attribute to optimize the overall extraction accuracy within a budget constraint. To address this, `Doctopus` employs the classical dynamic programming algorithm that effectively rations the given budget across attributes, eliminates redundant computations, and optimizes the overall quality.

In summary, we make the following contributions.

(1) We propose `Doctopus`, a structured table extraction framework that optimizes the extraction accuracy under a user-specified cost constraint. To the best of our knowledge, this is the first budget-aware optimization framework for LLM-powered table extraction.

(2) We introduce an indexing mechanism that reduces retrieval costs by segmenting documents into chunks. We further leverage LLMs to generate reference sentences, enhancing query informativeness. (Section 4)

(3) We develop a quality estimation mechanism that evaluates each extraction strategy based on two key metrics: the probability of the containment and the probability of successful extraction. This allows `Doctopus` to select the most effective strategy for each attribute while taking into account cost constraints. (Section 5)

(4) We present a dynamic programming algorithm that effectively allocates resources across multiple extraction strategies, achieving the optimal trade-off between cost and accuracy. (Section 6)

(5) We build a comprehensive benchmark with 4 real-world datasets, labeled by 20 graduate students with ≈1000 human hours. Extensive experiments over the benchmark demonstrate that compared with state-of-the-art baselines, `Doctopus` can improve the quality by 11% given the same cost constraint. Further, in comparison to state-of-the-art baselines, `Doctopus` can save the cost by 2.7× when achieving the same level of accuracy. (Section 7)

2 Problem Definition

Given a set of documents $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ and a set of user-specified attributes $\mathcal{A} = \{a_1, a_2, ..., a_m\}$, we aim to produce a table T with n tuples and m attributes, where each tuple $r_i \in T, i \in [1, n]$ is extracted from d_i based on the attribute set $\mathcal{A}$. The random variable representing the value of the j-th attribute of r_i is denoted as $r_{ij}, i \in [1, n], j \in [1, m]$, while v_{ij} denotes the attribute extracted from r_{ij}.

Example 1. *As shown in Figure 1, given a set of documents associated with some artists, a user wants to extract a table with attributes* $\mathcal{A} = \{Name, Birthplace, Lifespan, Genre, Field\}$. *From* d_1 *in the example, we can extract* $r_{11} =$ `Brull`, $r_{12} =$ `Spain` *and* r_1=`[Brull, Spain, 49, Portrait, Painting]`.

Next, we formally define *quality* and *cost*.

Quality. In this paper, we use *accuracy* to measure the extraction quality. Formally, we use v^*_{ij} to denote the ground truth value of the $j-$th attribute in r_i. Then, the overall accuracy is represented as follows:

$$Acc = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\mathbb{1}_{\{v_{ij}=v^*_{ij}\}}}{m \times n} \tag{1}$$

where $\mathbb{1}_{\{\cdot\}}$ denotes an indicator function that returns 1 if its argument is true; 0, otherwise. For example, $\mathbb{1}_{\{\texttt{"John Smith"="John M. Smith"}\}} = 1$ and $\mathbb{1}_{\{\texttt{"Canada"="Italy"}\}} = 0$. Note that $\mathbb{1}_{\{v_{ij}=v^*_{ij}\}}$ in Equation 1 represents whether the extracted attribute value v_{ij} and the ground truth v^*_{ij} refer to the same entity. If so, r_{ij} is considered to be extracted correctly.

The overall accuracy defined above requires the ground truth v^*_{ij}. However, `Doctopus` has to estimate the accuracy in the process of optimizing the extraction strategies where ground truth is not available. Therefore, we propose to use *expected accuracy* to measure the extraction quality as follows.

$$\mathbb{E}[\mathbb{1}_{\{v_{ij}=v^*_{ij}\}}] = P(v_{ij} = v^*_{ij}) \cdot 1 + P(v_{ij} \neq v^*_{ij}) \cdot 0 = P(v_{ij} = v^*_{ij}) \tag{2}$$

where $P(v_{ij} = v^*_{ij})$ denotes the probability of extracting the specific attribute r_{ij} correctly from a document. We will discuss how to compute the probability in Section 5. Therefore, this accuracy can be computed as the expected number of correctly extracted attributes out of all attributes:

$$Acc = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}P(v_{ij} = v^*_{ij})}{m \times n} \tag{3}$$

Cost. In this paper, we regard non-LLM strategies as cost-free since they are typically not computation intensive. For LLM-based strategies, the cost is calculated based on the total number of input/output tokens. As discussed above, different strategies present different input chunks to LLMs. We thus use c^s_{ij} to denote the number of input tokens that strategy s consumes to extract attribute a_j from d_i. Note that we omit the number of output tokens when computing the cost because it is rather small (only the attribute value) compared to the number of input tokens in this data extraction setting.

Our Problem. We use $\mathcal{S}_{ij}$ to denote the set of candidate strategies that can be used to extract the value of r_{ij}. To be specific, $\mathcal{S}_{ij} = \{\texttt{OpenIE}(d_i), \texttt{Codegen}(d_i), \texttt{PLM}(d_i), \texttt{LLM}(t_1), \texttt{LLM}(t_2), ...\}$. Each strategy $s \in \mathcal{S}_{ij}$ consists of the extraction approach and the input t. Thus, when we use non-LLM strategies, the input is directly d_i, while for LLM-based strategies, the input can be different combinations of the retrieved chunks. We use v^s_{ij} to denote the value extracted by the strategy $s \in \mathcal{S}_{ij}$. Then we use a mapping function $f : \{r_{ij}\}^{n,m}_{i=1,j=1} \rightarrow \mathcal{S}_{ij}$, such that for every $r_{ij} \in \{r_{ij}\}^{n,m}_{i=1,j=1}$, there exists $s \in \mathcal{S}_{ij}$ satisfying $f(r_{ij}) = s$, which indicates that r_{ij} is extracted by the strategy s. Our goal is to maximize accuracy under a budget constraint. Formally, we have:

$$f^* = \underset{f \in \mathcal{F}=\{f|f:\{r_{ij}\}\to\mathcal{S}_{ij}\}}{\arg\max} \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} P(v_{ij}^s = v_{ij}^*)}{m \times n}$$
$$s.t. \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}^s \le B, s = f(r_{ij}) \tag{4}$$

where B denotes the budget constraint on the total number of tokens consumed. The above equation aims to generate the optimal execution plan f^* that judiciously selects the most appropriate strategy for each attribute value to be extracted, in order to maximize quality within budget constraints.

Next, we overview the Doctopus framework that addresses this problem.

3 Doctopus Framework

Given a document collection $\mathcal{D}$, a set of user-specified attributes $\mathcal{A}$ and a cost constraint B, Doctopus selects an extraction strategy that maximizes the extraction accuracy, where the options include both index-based LLM strategies and non-LLM strategies.

As shown in Figure 2, Doctopus first builds the index (line 1), that is, dividing each document into chunks, embedding these chunks into vectors, and then indexing the vectors for later chunk retrieval (Section 3.1). Then, given a user-specified attribute set $\mathcal{A}$, Doctopus *samples* a small subset of documents to build a validation set (line 2) for subsequent reference generation and quality estimation (Section 3.1). The reason for generating references is that an attribute itself as a query is not informative enough to accurately retrieve the relevant chunks. We then enrich the query using these references. At a high level, we collect references by (1) analyzing the documents in the validation set (line 4); and (2) directly generating via LLMs (line 5), followed by combining them to produce the final query embedding (line 6) to retrieve relevant chunks and form the candidate strategies (line 9).

Subsequently, in Section 3.2, we estimate the quality of each strategy (line 10) as the product of the containment probability and the extraction probability. Finally, based on the cost of each strategy, Doctopus selects the most appropriate strategy to extract each specific attribute over different documents (line 11 - 15), achieving an optimal balance between accuracy and cost by solving Equation 4 (Section 6).

Next, we illustrate the above steps in more detail.

3.1 Attribute Extraction via Index

The key of this component is to leverage an index to accurately and efficiently identify the chunks that potentially contain the attribute and then only feed these chunks to LLMs to avoid unnecessary cost.

Index Building. As shown in the upper left corner of Figure 2, Doctopus starts by automatically dividing each document into semantically coherent chunks using existing NLP tools, which ensures that each attribute is likely to be extracted from a single chunk. Afterwards, these chunks are encoded into fixed-length embeddings using E5Model [30]. These embeddings are then organized into a high-dimensional vector index known as Product Quantization (PQ) [12], which facilitates the rapid retrieval of chunks by considering the semantic similarity between attributes and chunks. More details of the implementation are introduced in Section 4.

Algorithm 1: Doctopus Framework

Input: Document set $\mathcal{D}$, attribute set $\mathcal{A}$, cost budget B.
Output: The extracted table T.

1 $\mathcal{I}$ = Index_Building($\mathcal{D}$);
2 $\mathcal{D}_v$ = Validation_Set_Construction($\mathcal{D}$);
3 **for** *each* $a_j \in \mathcal{A}$ **do**
  // Attribute Enrichment
4 e_j^V = Reference($a_j, \mathcal{D}_v$);
5 e_j^L = Reference($a_j, LLMs$);
6 $e_j = e_j^V \oplus e_j^L$;
// Quality Estimation
7 **for** *each* $a_j \in \mathcal{A}$ **do**
8 **for** *each* $d_i \in \mathcal{D}$ **do**
9 $\mathcal{S}_{ij}$ = Strategy_Gen($d_i, a_j, e_j, \mathcal{I}$);
10 $P(v_{ij}^s = v_{ij}^*)$ = Quality_Estimation($\mathcal{D}_v, a_j, \mathcal{S}_{ij}$);
// Accuracy Optimization with Budget Constraint
11 f^* = Optimal_Execution_Plan($|\mathcal{D}|, |\mathcal{A}|, B, \mathcal{S}_{ij}$);
12 **for** *each* $d_i \in \mathcal{D}$ **do**
13 **for** *each* $a_j \in \mathcal{A}$ **do**
14 r_{ij} = Extract($d_i, a_j, f^*(r_{ij})$);
15 **return** T;

Remark. We choose PQ rather than other alternatives like Hierarchical Navigable Small World (HNSW) because of its lower memory consumption and faster index construction, which suits large-scale document processing. Although HNSW could potentially offer faster query response, the memory overhead would limit its scalability, especially when processing millions of documents. This trade-off between memory and accuracy makes PQ a more practical choice w.r.t. our specific requirements.

Attribute Enrichment based Retrieval. However, in addition to the semantic representation of the chunks, an accurate retrieval also depends on whether the query (i.e., the attribute) is informative enough. For example, suppose that we aim to extract the attribute Genre from documents. We can simply embed "Genre" as the search key to retrieve chunks using the index, which may not yield high-quality results because the relevant chunks may not explicitly mention the attribute as discussed in Section 1. Hence, the embeddings of sentences in relevant chunks are not necessarily similar to that of the attribute. Although the user can provide a description about the attribute as a search key, it can be regarded as a tedious prompt engineering task.

To address this, we propose to enrich the query from two perspectives. On the one hand, our aim is to *identify the patterns of the reference sentences where the attributes can be extracted*, and we *enrich the query with these patterns* as search key to achieve more accurate retrieval. This identification can be achieved by analyzing a validation set of documents built by humans or LLMs. However, purely considering the references obtained from the validation

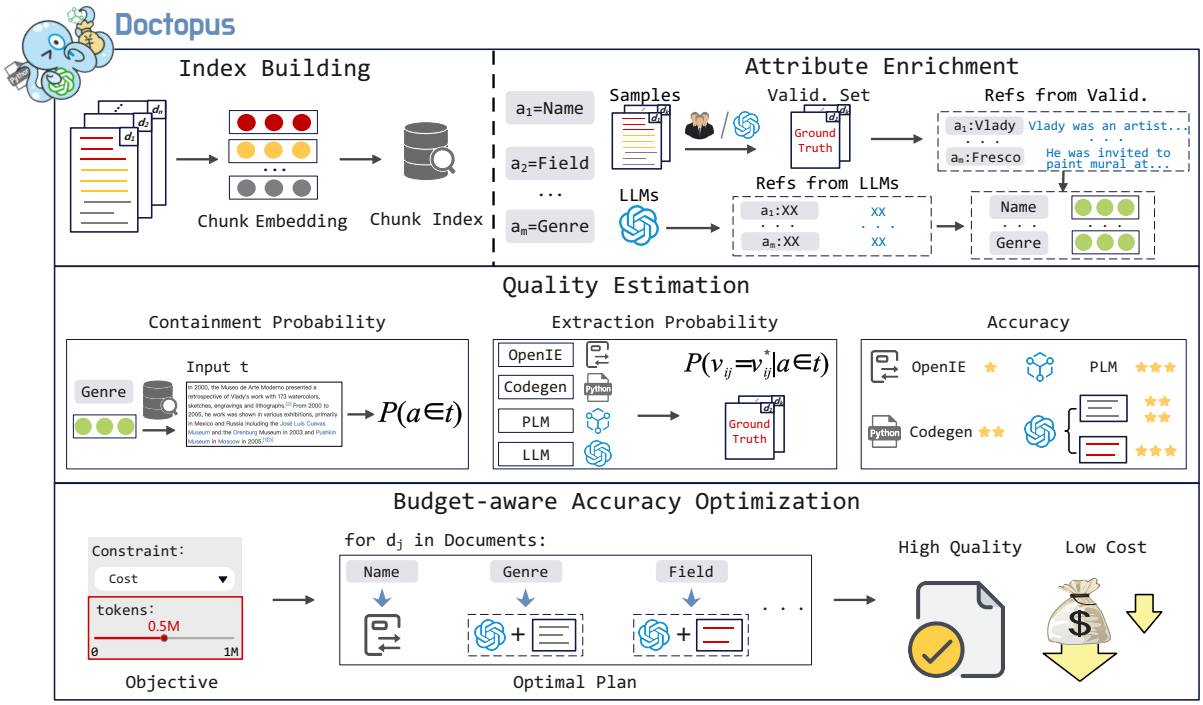

Figure 2: Overall Framework of Doctopus.

set may not generalize well to all documents. To address this, on the other hand, we leverage the strong generalization ability of LLMs to directly generate diverse references. Afterwards, putting the above two types of references together, we judiciously select some representative references, embed and concatenate them as the final query embedding to retrieve relevant chunks and form the candidate strategies. More details will be discussed in Section 4.

3.2 Quality Estimation for Multiple Strategies

In this part, we discuss how to estimate the quality of both LLM-based and non-LLM strategies when extracting different attributes from different documents based on a validation set – essential for solving Equation 4. As discussed in Section 1, the extraction accuracy of each strategy is determined by two key factors: (1) whether the input of each strategy contains the ground truth and (2) whether the strategy is capable of extracting the value correctly given that the input contains the ground truth.

Formally, we use *containment probability* $P(a \in t)$ to denote the likelihood that attribute a is present in the input text t. Recap that for non-LLM strategies, since they do not need to consider the token consumption (i.e., the cost), they can take the entire document as input. For the LLM-based strategy, since we aim to save cost, we only feed relevant chunks into LLMs (i.e., t corresponds to a combination of several chunks). On the other hand, we use *extraction probability* $P(v^s_{ij} = v^*_{ij} | a \in t)$ to denote the likelihood that the attribute value can be extracted correctly, given that the input t includes the attribute a. Thus, as shown in Figure 2, the overall quality can be represented as their product.

Then, we have to compute the two types of probability. For containment probability, we fine-tune the DistilBERT model based on the validation set. To be specific, given a, for each $d \in \mathcal{D}_v$, we have identified the chunk(s) containing a. Therefore, a together with the chunks can serve as positive training examples. Naturally, other chunks that do not contain a correspond to negative examples. However, since the size of the validation set is small, training examples, especially positive ones, are often not sufficient. Therefore, we propose to augment the training examples by combining more chunks to existing chunks, reordering sentences, and randomly removing sentences.

For the extraction probability, as observed in Section 7, it is relatively stable for the same attribute on different inputs as long as the input chunks contain the attribute. Thus, we compute it by applying each extraction approach over the documents in $\mathcal{D}_v$, comparing the extraction results with the ground truth, and deriving the probability using statistical methods rather than relying on a model to predict the probabilities with respect to different inputs.

In Section 5, we will introduce in detail how to compute these probabilities and derive an accurate quality estimation.

3.3 Accuracy Optimization with Budget Constraint

Once we have a method to estimate the quality of each extraction strategy, the next step is to choose the most appropriate strategy w.r.t. each attribute under the given budget constraint, i.e., solving Equation 4. The most straightforward way is to iterate through all possible strategies and identify the best one, with a complexity of

$O(|\mathcal{S}_{ij}|^{n\times m})$. To address this, we first show that it is an NP-hard problem by mapping it to the classical Knapsack problem. We then propose a dynamic programming algorithm to solve Equation 4, which processes each attribute $a_j \in \mathcal{A}$ with different strategies and records the accuracy that each strategy achieves on the attribute. Each strategy computes the budget consumed by the corresponding extraction, which in turn determines how much budget is available for the remaining extractions. Thus, it is able to effectively ration the given budget and optimally allocate them for incoming attribute extractions. Once the optimal execution plan is identified, a backtracking process reconstructs the best combination of strategies. By breaking the problem into subproblems and reusing their solutions, the algorithm eliminates redundant computations and significantly improves efficiency. After determining the most suitable strategy for each attribute, Doctopus processes the documents iteratively, extracts the specified attributes with the corresponding strategies and then outputs the final result.

4 Attribute Enrichment based Retrieval

This section presents techniques to enrich each attribute using references related to those attributes, aiming to improve the retrieval of relevant chunks. To achieve both high quality and diversity, Doctopus obtains references in two ways. First, Doctopus gets some references from a validation set. Second, Doctopus uses LLMs to generate some references based solely on the information of the attribute to enhance diversity. The two types of references are then embedded into vectors which are combined to establish the final search key for chunk retrieval.

Reference Generation via a Validation Set. First, as discussed in Section 3, we sample a small subset of documents denoted by $\mathcal{D}_v \subset \mathcal{D}$ and construct the validation set with the assistance of LLMs or humans. This validation set will also be utilized for quality estimation (Section 5).

Human Involvement. To ensure the quality of the validation set, the most straightforward method is to ask human experts to manually extract each attribute value along with its corresponding reference. However, this can be quite costly. Thus, in practice, we feed documents from $\mathcal{D}_v$ and the attributes associated with their descriptions as prompts into LLMs. Subsequently, we ask LLMs to thoroughly analyze each document d_i, extract the value of each attribute a_j, and output the corresponding reference denoted by re_i^j. The references for a_j obtained from $\mathcal{D}_v$ form the set R_V^j with $|R_V^j| = |\mathcal{D}_v|$. Finally, due to the inherent limitations of the LLMs themselves, they may introduce bias. To avoid this, we ask humans to check and verify the incorrect results.

Automatic LLM-based Construction. As discussed above, although humans are only employed to verify the results, manually reviewing each document remains laborious. Consequently, since LLMs typically are proficient in extracting attribute values and references with high quality, we opt to directly utilize the results processed by LLMs as the validation set and references, as demonstrated in our experiment (Section 7).

Token complexity analysis. For each dataset, we sample a ratio r of documents and process them with LLMs. Assuming an average document length of L tokens, this incurs $r \times |\mathcal{D}| \times L$ tokens.

Cluster to Remove Redundancy. For a particular attribute, there must exist redundant references in $|\mathcal{D}_v|$. Therefore, using all of them for retrieval tends to introduce too many candidate chunks. Hence, we group them into several clusters using the k-means algorithm. Then, we select the centers of these clusters as representatives and concatenate them as e_j^V.

Reference Generation via LLMs. However, solely relying on the references obtained from the validation set might not generalize to accurately retrieve all documents. Therefore, we employ LLMs to generate references using the attribute information as prompts, aiming to improve the diversity of the references. Although the generated references are related to the attribute, their text representations do not necessarily exist in any document of $\mathcal{D}$. We thus propose a generation-then-selection strategy that begins by instructing LLMs to generate references. Subsequently, we choose the k references most relevant to the document collection, and embed and concatenate them as e_j^L.

Direct generation. Specifically, for each attribute a_j, we initially utilize LLMs to produce h potential values for a_j. Subsequently, for every value, LLMs create h references. Collectively, the references related to a_j form the set R_M^j, with a cardinality of $|R_M^j| = h^2$. Next, we select k highly relevant references from them.

Token complexity analysis. For each attribute a_j, LLMs generate h^2 candidate references. With an average reference length of R tokens, this consumes $|\mathcal{A}| \times h^2 \times R$ tokens.

Reference selection. First, we compute a relevance score for each reference in R_M^j. A higher score indicates a higher relevance to the document set. To be specific, for each reference $re \in R_M^j$, within each document, we find the most similar chunk based on the cosine similarity of their embeddings. The relevance score for re is calculated by summing up the similarities of all documents within the set $\mathcal{D}$. Then, we select the k references with the highest scores.

Finally, we concatenate $e_j = e_j^V \oplus e_j^L$ as the search key. Afterwards, based on the index $\mathcal{I}$, Doctopus uses a threshold to prune the chunks that are irrelevant to the attribute to avoid unnecessary costs. To be specific, given e_j and the embedding $emb(ch)$ of a chunk ch in d_i, if $dist(emb(ch), e_j) > \tau$, we discard the chunk , where $dist()$ represents the normalized Euclidean distance calculated using the product quantization index $\mathcal{I}$.

Automatically Setting the τ. The threshold τ is also important for good retrieval. A large threshold returns irrelevant chunks, increasing the extraction cost, while a small threshold might miss relevant chunks, thus sacrificing the quality. Therefore, Doctopus uses the validation set to automatically identify an appropriate threshold (see Section 7 for details).

5 Quality Estimation

In this section, we illustrate how to estimate the accuracy, i.e., Equation 1, for different strategies. As discussed in Section 3.2, the accuracy is determined by containment probability and extraction probability. Next, we introduce in details how the accuracy is estimated based on the two probabilities and then discuss how to compute these probabilities.

To be specific, we cover the case that the ground truth is NULL, i.e., an attribute does not exist in a document (v^*_{ij} = NULL). Thus, accuracy $P(v^s_{ij} = v^*_{ij})$ can be expressed as:

$$\begin{aligned} P(v^s_{ij} = v^*_{ij}) = P(v^s_{ij} = v^*_{ij} | v^*_{ij} \neq \texttt{NULL}) \cdot P(v^*_{ij} \neq \texttt{NULL}) \\ + P(v^s_{ij} = v^*_{ij} | v^*_{ij} = \texttt{NULL}) \cdot P(v^*_{ij} = \texttt{NULL}) \end{aligned} \tag{5}$$

where $P(v^s_{ij} = v^*_{ij} | v^*_{ij} = \texttt{NULL})$ denotes the probability of correctly extracting the attribute when it exists in the document, which can be expanded further as follows.

$$\begin{aligned} P(v^s_{ij} = v^*_{ij} | v^*_{ij} = \texttt{NULL}) = P(a \in t) \cdot P(v^s_{ij} = v^*_{ij} | a \in t, v^*_{ij} = \texttt{NULL}) \\ + P(a \notin t) \cdot P(v^s_{ij} = v^*_{ij} | a \notin t, v^*_{ij} = \texttt{NULL}) \end{aligned} \tag{6}$$

Similarly, $P(v^s_{ij} = v^*_{ij} | v^*_{ij} \neq \texttt{NULL})$ can be expressed as:

$$\begin{aligned} P(v^s_{ij} = v^*_{ij} | v^*_{ij} \neq \texttt{NULL}) = P(a \in t) \cdot P(v^s_{ij} = v^*_{ij} | a \in t, v^*_{ij} \neq \texttt{NULL}) \\ + P(a \notin t) \cdot P(v^s_{ij} = v^*_{ij} | a \notin t, v^*_{ij} \neq \texttt{NULL}) \end{aligned} \tag{7}$$

Obviously, $P(v^s_{ij} = v^*_{ij} | a \notin t, v^*_{ij} \neq \texttt{NULL}) = 0$ because it is impossible to extract the correct value when it does not exist in the input text t. In addition, $P(v^s_{ij} = v^*_{ij} | a \in t, v^*_{ij} = \texttt{NULL}) = 0$ because v^*_{ij} = NULL and $a \in t$ are in conflict. Putting Equation 5, 6 and 7 together, we have:

$$\begin{aligned} P(v^s_{ij} = v^*_{ij}) = P(a \in t) \cdot P(v^s_{ij} = v^*_{ij} | a \in t, v^*_{ij} \neq \texttt{NULL}) \cdot P(v^*_{ij} \neq \texttt{NULL}) \\ + P(a \notin t) \cdot P(v^s_{ij} = v^*_{ij} | a \notin t, v^*_{ij} = \texttt{NULL}) \cdot P(v^*_{ij} = \texttt{NULL}) \end{aligned} \tag{8}$$

In Equation 8, $P(a \in t)$ and $P(a \in t)$ refer to the containment probability, $P(v^s_{ij} = v^*_{ij} | a \in t, v^*_{ij} \neq \texttt{NULL})$ and $P(v^s_{ij} = v^*_{ij} | a \notin t, v^*_{ij} = \texttt{NULL})$ correspond to the extraction probability, and the remaining terms can be considered as the prior probability of the existence of the attribute in $\mathcal{D}$. Next, we discuss how to compute them.

Prior probability. $P(v^*_{ij} \neq \texttt{NULL})$ and $P(v^*_{ij} = \texttt{NULL})$ can be easily estimated based on the validation set. Specifically, we denote the total number of NULL values of the attribute a_j in $\mathcal{D}_v$ as N_j, and $P(v^*_{ij} = \texttt{NULL}) = \frac{N_j}{|\mathcal{D}_v|}$. Then, $P(v^*_{ij} \neq \texttt{NULL}) = 1 - P(v^*_{ij} = \texttt{NULL})$.

Containment Probability Estimation. The key idea is to fine-tune a pre-trained language model to predict $P(a \in t)$ and $P(a \notin t)$. Next, we first introduce the model construction process, employing the documents in the validation set as training data. We then augment the training set to overcome the problem that the validation set tends to be relatively small.

Model Fine-tuning. We fine-tune the lightweight DistilBERT model as follows: the attribute and its corresponding chunk t are concatenated and fed into the model. The model then produces a probability score, i.e., $P(a \in t)$, through a linear layer followed by a sigmoid activation. The training process employs the binary cross-entropy loss, defined as:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right], \tag{9}$$

where $y_i \in \{0, 1\}$ denotes the ground-truth label (1 if t contains a, 0 otherwise) and p_i is the predicted probability. Then, we use the fine-tuned model to predict the containment probability.

Training Data Augmentation. The training dataset obtained by $\mathcal{D}_v$ may not sufficient enough, especially the positive examples. To enhance the amount and diversity of the training dataset, we apply three different data augmentation techniques:

(i) Chunk Combination: Increase positive examples by combining the attribute chunk with irrelevant chunks.

(ii) Sentence Reordering: Shuffle sentences within chunks to create varied contexts while keeping the semantic meaning. Sentences before the attribute may be moved after it.

(iii) Random Sentence Deletion: Randomly delete sentences from training examples, but not from the attribute chunk in positive examples.

After fine-tuning the model, the input text, concatenated with the attribute information, is fed into the model, which then predicts the containment probability $P(a \in t)$.

Extraction Probability Estimation. Then, we have to estimate the extraction probability for different strategies, i.e., the probability that each strategy can accurately extract an attribute given $P(a \in t)$ or $P(a \notin t)$.

Specifically, we split the training set of the DistilBERT model into two parts for each attribute a_j: H^+_j comprising chunks that contain the attribute and H^-_j comprising chunks without the ground truth value. Then, we feed the chunks to LLM to extract the values of the specified attribute and then compare these extracted values to the ground truth in the validation set. For each attribute a_j and strategy s, we compute the average accuracy, i.e., $P(v^s_{ij} = v^*_{ij} | a \in t, v^*_{ij} \neq \texttt{NULL}) = \frac{\sum_{i=1}^{|H^+|} \mathbb{1}_{\{v^s_{ij} = v^*_{ij}\}}}{|H^+|}$ and $P(v^s_{ij} = v^*_{ij} | a \notin t, v^*_{ij} = \texttt{NULL}) = \frac{\sum_{i=1}^{|H^-|} \mathbb{1}_{\{v^s_{ij} = \texttt{NULL}\}}}{|H^-|}$) as the extraction probability. Note that $\mathbb{1}_{\{v^s_{ij} = v^*_{ij}\}}$ can be calculated through various methods, such as exact match, text F1 score, entity matching model, BertScore, BLEU or employing LLMs. To be specific, when using the exact match, $\mathbb{1}_{\{v^s_{ij} = v^*_{ij}\}} = 1$ if the extracted value v^s_{ij} exactly matches the ground truth v^*_{ij}. If the text F1 score is used, $\mathbb{1}_{\{v^s_{ij} = v^*_{ij}\}} = 1$ if the text F1 score between the extracted value v^s_{ij} and the ground truth v^*_{ij} is greater than 0.5. Furthermore, an entity match model, such as DITTO [20] as well as BertSocre and BLEU, can also compute the probability. Finally, LLM can directly determine whether $\mathbb{1}_{\{v^s_{ij} = v^*_{ij}\}}$ equals to 1. We perform experiments using these alternate methods (refer to Section 7.9.2). For non-LLM strategies, Doctopus estimate their extraction probability in a similar way. The key difference is that non-LLM strategies take each entire document in $\mathcal{D}_v$ as input instead of text chunks.

6 Budget-aware Accuracy Optimization

Based on our problem definition (Section 2), given a specific document d_i and an attribute a_j, we have a set S_{ij} of candidate strategies. In this section, we first discuss how to generate the candidate set (Section 6.1), followed by choosing the most appropriate strategy w.r.t. each extraction. We first prove the hardness of the problem

and then propose a dynamic programming algorithm to solve it (Section 6.2).

6.1 Candidate Strategies Generation

In Section 4, we first use embedding e_j to retrieve l chunks relevant to a_j. Obviously, there exist 2^l possible combinations of chunks as the input of an LLM, corresponding to 2^l different strategies. Enumerating all these strategies is prohibitively expensive. To reduce the search space, we propose a pruning method to reduce the number of candidates. Please refer to [1] for details.

6.2 Dynamic Programming Algorithm

Problem Complexity. After determining the set of strategies $\mathcal{S}_{ij}$ for each r_{ij}, we estimate the quality of each strategy in $\mathcal{S}_{ij}$ using the method discussed in Section 5. Then, we need to solve Equation 4 to find the optimal execution plan to maximize the extraction accuracy. Due to limited space, the proof showing that solving Equation 4 is NP-hard has been included in our technique report [1]. We then solve this problem with a dynamic programming algorithm.

Dynamic Programming Algorithm. We use the classical DP algorithm to address the Knapsack problem. The key idea is to iteratively allocate the budget to attributes while tracking the optimal accuracy at each budget level. Note that during the quality estimation phase, the accuracy $P(v_{ij}^s = v_{ij}^*)$ and cost c_{ij}^s of each strategy s for the j-th attribute in the i-th document are precomputed and serve as inputs to the DP algorithm.

As illustrated in Algorithm 2, we have a total of $n \times m$ attributes to extract. Then, we define a table dp where $dp[g][b], g \in [0, n \times m], b \in [0, B]$ represents the maximum accuracy that can be achieved by considering the first g attributes with budget b (line 1), and define $sel[g][b]$ as the strategy s selected for the g−th attribute under budget b (line 2). We iterate all the $n \times m$ attributes and map the current g−th attribute back to the corresponding document d_i and attribute a_j by computing $i = \lfloor (g-1)/m \rfloor + 1$ and $j = (g-1) \bmod m + 1$ (line 4). Then, we define the state transition formula as below:

$$dp[g][b] = \max_{s \in \mathcal{S}_{ij}} \{dp[g-1][b - c_{ij}^s] + P(v_{ij}^s = v_{ij}^*) \text{ if } b \geq c_{ij}^s\} \tag{10}$$

Specifically, for each level of budget b from 0 to B, we iterate over all strategies s in the set of strategies $\mathcal{S}_{ij}$ and check whether the cost c_{ij}^s of selecting strategy s is within the current budget b (line 7). If so, the accuracy currently achieved by choosing the strategy s for r_{ij} is calculated as $value = dp[g-1][b - c_{ij}^s] + P(v_{ij}^s = v_{ij}^*)$, where $dp[g-1][b - c_{ij}^s]$ represents the maximum accuracy that can be achieved for the first $g-1$ attributes using the budget $b - c_{ij}^s$ (line 8). If $value$ exceeds current $dp[g][b]$, the algorithm updates $dp[g][b]$ to the new maximum accuracy (line 10) and records the selected strategy $sel[g][b]$ for the g−th attribute (line 11).

Optimal Execution Plan Generation. After obtaining the complete dp table, the optimal execution plan is contracted as follows: Starting with the total budget B and the $n \times m$−th attribute, we map the current g−th attribute back to the corresponding document d_i and the attribute a_j (line 14). If a strategy s is selected for extracting r_{ij}, it is added to the final execution plan f^* (line 15), and the remaining budget b is reduced by the corresponding cost $c_{ij}^{sel[g][b]}$ (line 16). The step continues until all attributes have been processed. Finally, the algorithm returns the optimal execution plan f^* (line 17), which maximizes accuracy under the budget B.

Algorithm 2: Optimal_Execution_Plan

Input: Number of documents n, number of attributes m, total budget B, the set of candidate strategies $\mathcal{S}_{ij}$.
Output: $f^* : \{r_{ij}\}_{i=1,j=1}^{n,m} \rightarrow \mathcal{S}_{ij}$

```
1  dp[][] = 0;
2  sel[][] = −1;
3  for g = 1 to n × m do
4      (i, j) = Get_Index(g, m);
5      for b = 0 to B do
6          for each s ∈ S_{i,j} do
7              if b ≥ c_{ij}^s then
8                  value = dp[g − 1][b − c_{ij}^s] + P(v_{ij}^s = v_{ij}^*);
9                  if value > dp[g][b] then
10                     dp[g][b] = value;
11                     sel[g][b] = s;
12 b = B;
13 for g = n × m to 1 do
14     (i, j) = Get_Index(g, m);
15     f*(r_{ij}) = sel[g][b];
16     b = b − c_{ij}^{sel[g][b]};
17 return f*;
```

Time Complexity Analysis. The time complexity of Algorithm 2 is determined by three nested loops in the first computation phase (lines 3–11). For each document and each attribute within a document ($n \times m$ iterations), the algorithm processes each budget level from 0 to B and iterates over all the strategies in $\mathcal{S}_{ij}$. This results in a total complexity of $O(nmB|\mathcal{S}_{ij}|)$ for the first phase. The second phase (lines 13–16) has a linear complexity of $O(nm)$ as it iterates all attributes. Therefore, the overall time complexity of the algorithm is $O(nmB|\mathcal{S}_{ij}|)$.

Token Complexity Analysis. The token complexity of Algorithm 2 is the sum of tokens consumed by selected strategies, i.e., $\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}^s$, where c_{ij}^s denotes the token cost of strategy s for attribute a_j in document d_i. Our dynamic programming algorithm ensures that this sum does not exceed the user-specified budget B.

7 Experiment

In this section, we first outline our experimental setup. Then, we conduct extensive experiments to compare Doctopus with 8 baseline methods on 4 datasets to answer the following questions.

- **Q1:** How does Doctopus compare against other baselines in accuracy and cost?
- **Q2:** How do different attribute enrichment methods affect the performance of Doctopus?

- **Q3:** How do different extraction approaches affect the performance of Doctopus?
- **Q4:** How does Doctopus perform with different LLMs?
- **Q5:** How does Doctopus perform in computing the containment probability?
- **Q6:** How is the efficiency of Doctopus compared with other baselines?
- **Q7:** How stable is the extraction probability for all strategies?
- **Q8:** Other ablation studies.

7.1 Experimental Settings

Table 1: Statistics of Datasets.

Dataset	#-Docs	#-Attrs	Avg. #-Tokens	NULL (%)
Law [11]	600	9	5926	10.8%
Wikiart [2]	1000	8	714	12.8%
Financial [3]	1500	8	258	15.2%
Sports [4]	100	10	1645	17.1%

Datasets. We develop four benchmark datasets with 3,200 documents across various domains to evaluate our system. Dataset statistics are in Table 1.

Law includes 600 documents from 3,000 Federal Court of Australia case reports (2006-2009). Each document averages 5,926 tokens and 9 attributes.

Wikiart includes 1,000 documents describing the biographies of various artists. Each document averages 714 tokens and 8 attributes.

Financial includes 1,500 documents that provide details about various companies. On average, each document comprises 258 tokens and 8 attributes.

Sports consists of 100 Wikipedia pages about NBA players. We transform each webpage into a document that includes only text. Each document typically includes 1,645 tokens and provides information on 10 attributes.

Ground Truth Creation. For each dataset, we initially identify the key attributes with the help of human experts. Following this, we employ LLMs to scan all documents and extract these attributes. Ultimately, the extracted attributes are verified by 20 graduate students using approximately 1,000 hours.

Baselines. We compare Doctopus with various baselines.

(1) Lotus [24] utilizes semantic operators powered by LLMs, like *sem_map* and *sem_extract*, to extract attributes from unstructured documents.
(2) Palimpzest (PZ) [21] offers the *convert* operator to extract attributes from unstructured documents through the use of LLMs. We employ the same LLM with us for PZ for a fair comparison.
(3) Evaporate [6] is a code generation method that first samples several documents and prompts LLMs to generate various candidate extraction codes. Then it leverages the weak supervision technique to aggregate the results of multiple codes. Since we only use LLMs to generate codes relying on several documents and apply the codes to the rest, the LLMs costs can be neglected.
(4) OpenIE6 [18] employs the iterative grid labeling technique to extract triples (subject, relation, object) from unstructured documents. The extracted triples are then organized into structured tables by mapping subjects to rows, relations to columns, and objects to cell values.
(5) DebertaV3 [13] is a pre-trained language model commonly used in QA tasks to extract information that is relevant to a user query from a provided document context, which can be utilized to extract attribute values. We incorporate this PLM-based method as one of our non-LLM strategies.
(6) FrugalGPT [8] selects an execution order of different approaches based on their predicted accuracy and cost to answer NL tasks. To compare with it, we regard the NL task as extracting attributes and incorporate the same extraction approaches for a fair comparison.
(7) LLM-Extract still utilizes our budget-aware accuracy optimization framework but excludes non-LLM strategies.
(8) Doctopus is our full-fledged solution, which incorporates OpenIE6, Evaporate, DebertaV3 and GPT-4o as our extraction approaches.

Evaluation Metrics. We evaluate accuracy, cost, and efficiency across all datasets. For accuracy, we use Equation 1 for overall accuracy. As mentioned in Section 5, $\mathbb{1}_{\{v_{ij}^s = v_{ij}^*\}}$ has multiple calculation methods. We use the text F1 score from [6] for $\mathbb{1}_{\{v_{ij}^s = v_{ij}^*\}}$ and report overall accuracy. Additional experiments using various accuracy methods are discussed in Section 7.9.2. For cost, we count the total number of tokens input to the LLM (GPT-4o) through the OpenAI API. For efficiency, we report the average time of different extraction methods per document.

Hyper-parameter Setting. We sample 10% of each dataset as the validation set. We use LLM to extract attributes, and humans verify any inaccuracies. The same validation set is used for FrugalGPT. We generate references using the validation set and LLMs, selecting 5 references each. For fine-tuning the DebertaV3 model, we set the batch size as 32 and the learning rate as $3e-5$ with a linearly decreasing learning rate schedule. Notably, the chunk size affects both accuracy and efficiency. While utilizing smaller chunks can reduce the number of LLM input tokens, it might also lose relevant contextual information, thereby sacrificing the accuracy of attribute extraction. Furthermore, a smaller chunk size could produce more chunks, which potentially hurts the retrieval efficiency. On the other hand, employing larger chunks could include more related information, thereby enhancing the extraction accuracy. Nonetheless, this may lead to increased token consumption. Therefore, it's important to set the chunk size automatically to achieve an optimal trade-off between accuracy and efficiency across all datasets.We use the SemanticChunker [5] in LangChain to set the chunk size.

We set the threshold τ automatically. Specifically, for each document $d_i \in \mathcal{D}_v$, Doctopus first uses a small τ to obtain a set of chunks H_j^i using e_j. Then, all elements in H_j^i are combined and form the set of H_j, which represents all the retrieved chunks in $\mathcal{D}_v$ using the small τ. Since $\mathcal{D}_v$ has been analyzed by LLMs or humans, we can partition H_j into two distinct subsets: H_j^-, which includes chunks that do not contain the attribute and are therefore considered irrelevant to the query, and $H_j^+ = H_j \setminus H_j^-$, which comprises relevant chunks. The maximum Euclidean distance between the chunk embeddings $emb(ch)$ in H_j^+ and the embedding e_j serves as the threshold, i.e., $\tau = \max\{dist(emb(ch), e_j) | ch \in H_j^+\}$. Intuitively,

this τ threshold will exclude irrelevant chunks. We summarize the steps where LLMs are used in Doctopus and make the corresponding prompts available. Please refer to our GitHub repository.

Remark. When multiple attributes are mentioned in the same or adjacent chunks, we will merge these chunks and input them into the LLMs to extract them concurrently. Besides, estimating joint extraction quality for correlated attributes could further optimize cost-accuracy trade-offs. Future work will extend our quality estimation framework to account for attribute combinations.

7.2 Comparison with Baselines (Q1)

Figure 3 illustrates the comparison of Doctopus with baselines on accuracy (Y-axes) across different cost constraints (X-axes). Doctopus offers the best accuracy-cost balance, and Doctopus leads in accuracy under these constraints. Specifically, Doctopus, PZ, and Lotus excel across all datasets. Lotus and PZ, despite high accuracy, incur substantial costs by processing entire documents with LLMs. Meanwhile, Doctopus matches PZ and Lotus in accuracy but at a lower cost. On the Law dataset, Doctopus, PZ, and Lotus achieve approximately 75% accuracy. However, PZ and Lotus process 3.8 million and 5.8 million tokens, respectively, 3.2× and 4.8× times more than Doctopus. This is due to an efficient method that targets specific content for LLM processing, cutting costs while maintaining accuracy. In cases like Lifespan, non-LLM strategies Evaporate can outperform LLM-based methods by boosting accuracy and cutting costs. Lotus uses more tokens as it requires two operators, *sem_map* and *sem_extract*, for processing, doubling LLM usage per document. Conversely, PZ processes the document once for simultaneous attribute extraction.

Doctopus outperforms FrugalGPT as it uses a uniform pipeline for all attributes and documents, reaching a local optimum, while Doctopus achieves global optimization by considering diverse attributes and documents. On dataset Wikiart, FrugalGPT achieves 77% accuracy, whereas Doctopus reaches 85% with 0.75 million tokens. Doctopus outperforms LLM-Extract because non-LLM strategies excel in extracting specific attributes. By integrating both non-LLM and LLM-based strategies, Doctopus could achieve higher accuracy on the same budget.

Non-LLM strategies OpenIE6, DebertaV3, and Evaporate are less accurate than Doctopus. OpenIE6 struggles with adapting to diverse documents due to its limitation to well-structured sentences. DebertaV3 underperforms as DebertaV3 finds it challenging to generalize across varied attributes and documents. The performance of Evaporate is suboptimal because the generated codes inherently rely on a limited number of rules, which are often unreliable when handling complex documents or attributes.

Constructing a validation set is cost-effective and beneficial. For example, on the Financial dataset, Doctopus uses only 0.05 million tokens for validation. With this cost, Doctopus achieves 84% accuracy, surpassing FrugalGPT's 78%. Doctopus significantly outperforms non-LLM strategies Evaporate (52%), DebertaV3 (33%), and OpenIE6 (20%). This is due to Doctopus's adaptive strategy, balancing cost and accuracy.

7.3 Evaluation of Reference (Q2)

We evaluate the attribute enrichment method in Doctopus by comparing with three baselines.

(1) Ref-Val only generates the reference for each attribute through the validation set.

(2) Ref-LLM generates the reference directly through LLM.

(3) No-Reference does not incorporate the attribute enrichment method to enhance retrieval. It just simply encodes the attributes and a piece of description into embeddings for retrieval.

In Figure 4, Doctopus outperforms No-Reference as embedding attributes with descriptions lacks adequate information, leading to missed chunks. Ref-Val underperforms due to relying solely on validation set references, which fails to generalize and reduces accuracy. Similarly, Doctopus outperforms Ref-LLM because LLM-generated reference patterns do not align with the document set, impacting accuracy.

7.4 Evaluation of Strategy Set (Q3)

To demonstrate the effectiveness of different strategies considered in Doctopus, we compare Doctopus with methods as follows:

(1) Doctopus Without OpenIE (D-W-O): Doctopus excludes the extraction approach OpenIE6.

(2) Doctopus Without Codegen (D-W-C): Doctopus excludes the extraction approach Evaporate.

(3) Doctopus Without PLM (D-W-P): Doctopus excludes the extraction approach DebertaV3.

As shown in Figure 5, Doctopus outperforms all the methods because Evaporate, OpenIE6 and DebertaV3 are all good at extracting some specific attributes. For example, Evaporate excels in using rules for extracting attributes, like computing Lifespan by subtracting birth_date from death_date. Thus, it is important to consider all of them into the extraction process and sufficiently take their advantages like Doctopus.

7.5 Evaluation of Different LLMs (Q4)

Doctopus is evaluated using different LLMs (*i.e.*, GPT-3.5 and GPT-4o mini) against FrugalGPT and PZ. As shown in Figure 6, Doctopus consistently outperforms FrugalGPT and PZ across various LLMs. The attribute-enrichment method improves cost-effectiveness without losing accuracy by retrieving the most relevant chunks for LLMs. Additionally, Doctopus selects the best approach for each attribute and document, balancing cost and accuracy.

7.6 Accuracy of Containment Model (Q5)

We evaluate the accuracy of our containment probability estimation model (i.e., DistilBERT) with the data augmentation techniques. We divide the training data of the DistilBERT model into a training set (90%) and a test set (10%), and use accuracy as the evaluation metric. We compare our containment probability estimation model trained with the data augmentation techniques (*i.e.*, DistilBERT-DA) against GPT-4o and the DistilBERT model without using data augmentation (*i.e.*, DistilBERT-base). As shown in Figure 7, DistilBERT-DA is competitive with GPT-4o and outperforms DistilBERT-base because it is fine-tuned with our augmented data, which better captures the relationship between attribute and document content.

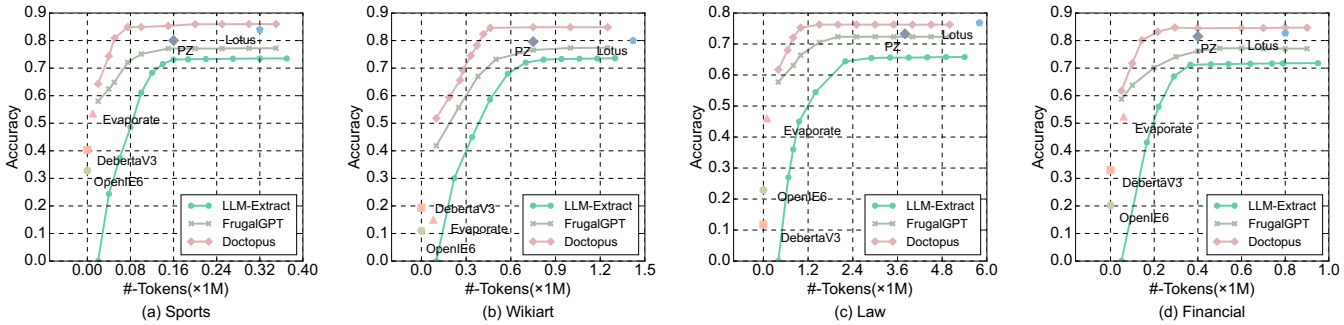

Figure 3: Overall Performance Comparison with Baselines.

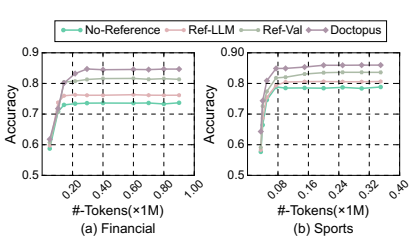

Figure 4: Evaluation of Attribute Enrichment Method.

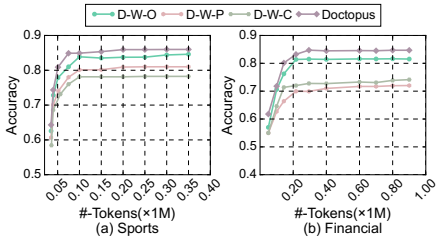

Figure 5: Performance of Doctopus under Different Strategy Sets.

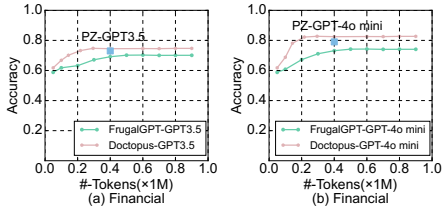

Figure 6: Performance of Doctopus using Different LLMs.

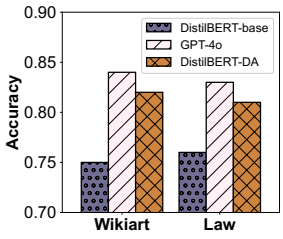

Figure 7: Containment Probability Evaluation.

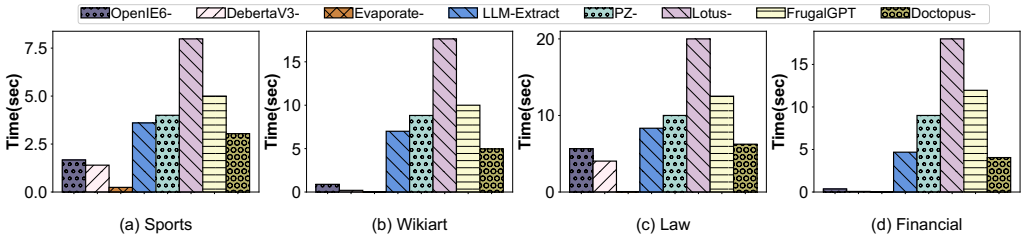

Figure 8: Overall Runtime Comparison with Baselines.

7.7 Evaluation of Overall Runtime (Q6)

Figure 8 illustrates overall runtimes. `Evaporate` is the most efficient, leveraging the extraction code. `DebertaV3` and `OpenIE6` are efficient but less accurate, as they avoid LLMs. `Doctopus` is more efficient than `PZ` and `Lotus`, which process the entire document via LLM. `Doctopus` is superior to `FrugalGPT`, since `FrugalGPT` requires multiple extraction attempts per attribute and document. Finally, `Doctopus` surpasses `LLM-Extract`, with `LLM-Extract` using LLMs solely for extraction.

7.8 Evaluation of Extraction Probability (Q7)

We assess that when an attribute is in the input text, extraction probabilities across documents are stable. We cluster each dataset into 5 groups based on document summarization embeddings, randomly select 3 attributes per dataset, and apply non-LLM strategies within each cluster to extract these attributes, recording the accuracy. Additionally, we extract attribute-containing chunks using LLMs and report their accuracy. Figure 9 shows minimal accuracy variation across different clusters for the same attribute and strategy, supporting our statistical method for computing extraction probability without document-specific considerations.

7.9 Ablation Study (Q8)

7.9.1 Evaluation of Validation Set. We assess the impact of human involvement in building the validation set versus using only LLM. Figure 10 shows `VS-LLM`, a version of `Doctopus` using LLM for the validation set. While `Doctopus` outperforms `VS-LLM` due to human-enhanced quality estimation, `VS-LLM` also performs well as the LLM constructs a high-quality set.

7.9.2 Methods of Computing $\mathbb{1}_{\{v_{ij}^s = v_{ij}^*\}}$. We evaluate the performance of `Doctopus` by calculating Equation 1 with different methods, namely exact match, entity matching model [20], BertScore, BLEU and use of LLMs, as described in Section 5. As illustrated in Figure 11, `Doctopus` achieves better performance when Equation 1

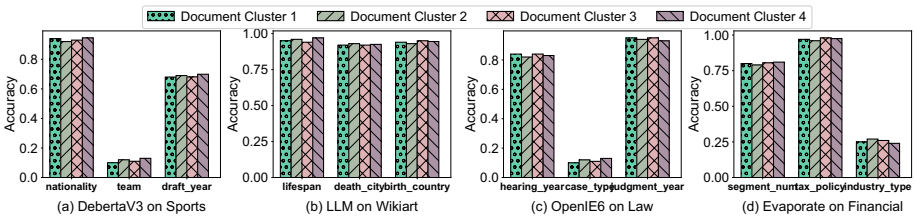

Figure 9: Evaluation of Extraction Probability.

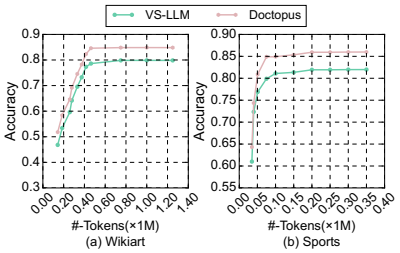

Figure 10: Evaluation of Validation Set.

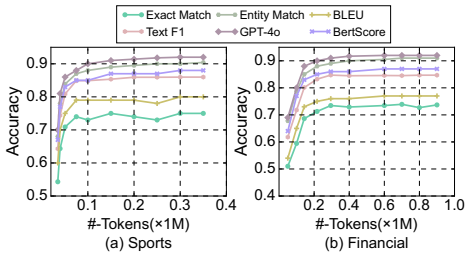

Figure 11: Performance of Doctopus Under Different Evaluation Methods.

is calculated using either the GPT-4o or entity matching model. This is because LLMs and the entity matching model better capture the semantic relationship between extracted values and the ground truth, thus estimating the quality of each strategy more precisely and improving extraction accuracy.

8 Related Work

Structural Data Extraction extracts structured information from unstructured data, which has been studied for decades. One approach involves extracting triples by identifying entities and their relationships from text, shifting from rule-based or statistical methods [19, 23, 26, 27] to deep learning. For instance, `OpenIE6` [18] uses iterative grid labeling, while `MacroIE` [33] employs a BERT-based encoder to learn token span representations and form graphs to identify triplets. These often struggle with complex documents and implicit relationships. Alternatively, code generation is used for extraction, such as `Evaporate` [6] utilizing LLMs to produce code for extracting attributes from HTML and PDF files, but LLM-generated code alone is often insufficiently accurate for diverse documents.

PLMs are used for data extraction [7, 13, 16, 25, 31, 35]. For instance, `DebertaV3` [13] employs a QA pre-trained model to extract information relevant to queries. `Text-to-Table` [31] uses a sequence-to-sequence framework to transform text into structured tables. `STable` [25] features a permutation-based decoder for flexible and efficient text-to-table inference. However, these methods are not effective in extracting information not explicitly stated in the text, which requires domain-specific knowledge for extraction.

Recent works explore using LLMs for data extraction. `Lotus` [24] and `PZ` [21] prompt LLMs with NL queries to extract specific attributes from unstructured documents, but heavily rely on LLMs, leading to high computational costs. Other studies [9, 17, 29] use LLMs for unstructured data analysis without optimizing these costs.

Cost Reduction of LLMs has been recently studied to balance accuracy and cost of LLMs. Typically, some works consider both cheap approaches and expensive LLMs to handle NL tasks [8, 15, 34]. For example, `FrugalGPT` [8] orders various LLM strategies for a task, optimizing cost by assessing quality and cost. It trains a reward model to evaluate strategy quality, following the order to decide on accepting each result based on a reward function. However, for all tasks, `FrugalGPT` follows the same order of different strategies to obtain the result, resulting in suboptimal performance.

9 Conclusion and Future Work

We present `Doctopus`, a system for cost-effective attribute extraction from unstructured documents. It uses an index-based approach to process relevant text chunks. We assess the quality of various strategies and propose a dynamic programming algorithm to combine LLMs with non-LLM methods, optimizing cost and accuracy. Real-world experiments verify the cost-effectiveness of `Doctopus`.

New LLM versions will continually be released, challenging `Doctopus` to adapt to LLM life-cycles. The fast evolution of LLMs requires periodic updates for `Doctopus` to integrate with the latest models—such as re-evaluating their quality—to maintain optimal performance, possibly involving significant effort. We will explore this promising direction as future work. In addition, `Doctopus` does not currently integrate multiple LLMs and assumes each document attribute has a single value, which may not suit all industrial cases. Future work could address this by combining various LLMs to enhance their strengths and enable the extraction of multiple values per document, thus increasing robustness.

ACKNOWLEDGMENTS

Chengliang Chai is supported by the NSF of China (62472031), the National Key Research and Development Program of China (2024YFC3308200), Beijing Nova Program, CCF-Baidu Open Fund (CCF-Baidu202402), and Huawei. Yuhao Deng is supported by the NSFC (624B2023) and the BIT Research and Innovation Promoting Project (2024YCXZ004). Ye Yuan is supported by the Beijing Natural Science Foundation (L241010), the National Key Research and Development Program of China (2022YFB2702100), and the NSFC (61932004, 62225203, U21A20516). Guoren Wang is supported by the NSFC (62427808, U2001211), and the Liaoning Revitalization Talents Program (XLYC2204005). Lei Cao is supported by the NSF (DBI-2327954) and Amazon Research Awards.

References

[1] [n. d.]. https://github.com/mutong184/Doctopus/blob/main/Doctopus__tech_report_.pdf

[2] [n. d.]. https://www.wikiart.org/

[3] [n. d.]. https://finance.yahoo.com/

[4] [n. d.]. https://en.wikipedia.org/wiki/Lists_of_NBA_players

[5] [n. d.]. https://python.langchain.com/docs/how_to/semantic-chunker/

[6] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proc. VLDB Endow.* 17, 2 (2023), 92–105. https://doi.org/10.14778/3626292.3626294

[7] Ansel Blume, Nasser Zalmout, Heng Ji, and Xian Li. 2023. Generative Models for Product Attribute Extraction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: EMNLP 2023 - Industry Track, Singapore, December 6-10, 2023*, Mingxuan Wang and Imed Zitouni (Eds.). Association for Computational Linguistics, 575–585. https://doi.org/10.18653/V1/2023.EMNLP-INDUSTRY.55

[8] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *CoRR* abs/2305.05176 (2023). https://doi.org/10.48550/ARXIV.2305.05176 arXiv:2305.05176

[9] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Samuel Madden, and Nan Tang. 2023. Symphony: Towards Natural Language Query Answering over Multi-modal Data Lakes. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*. www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p51-chen.pdf

[10] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, Ricardo Baeza-Yates and Francesco Bonchi (Eds.). ACM, 6491–6501. https://doi.org/10.1145/3637528.3671470

[11] Filippo Galgani and Achim Hoffmann. 2010. LEXA: Towards Automatic Legal Citation Classification. In *AI 2010: Advances in Artificial Intelligence (Lecture Notes in Computer Science, Vol. 6464)*, Jiuyong Li (Ed.). Springer Berlin Heidelberg, 445 –454.

[12] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.

[13] Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=sE7-XhLxHA

[14] Gautier Izacard, Patrick S. H. Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot Learning with Retrieval Augmented Language Models. *J. Mach. Learn. Res.* 24 (2023), 251:1–251:43. https://jmlr.org/papers/v24/23-0037.html

[15] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 14165–14178. https://doi.org/10.18653/V1/2023.ACL-LONG.792

[16] Yizhu Jiao, Ming Zhong, Sha Li, Ruining Zhao, Siru Ouyang, Heng Ji, and Jiawei Han. 2023. Instruct and Extract: Instruction Tuning for On-Demand Information Extraction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 10030–10051. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.620

[17] Saehan Jo and Immanuel Trummer. 2024. ThalamusDB: Approximate Query Processing on Multi-Modal Data. *Proc. ACM Manag. Data* 2, 3 (2024), 186. https://doi.org/10.1145/3654989

[18] Keshav Kolluru, Vaibhav Adlakha, Samarth Aggarwal, Mausam, and Soumen Chakrabarti. 2020. OpenIE6: Iterative Grid Labeling and Coordination Analysis for Open Information Extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 3748–3761. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.306

[19] Taesung Lee, Zhongyuan Wang, Haixun Wang, and Seung-won Hwang. 2013. Attribute extraction and scoring: A probabilistic approach. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou (Eds.). IEEE Computer Society, 194–205. https://doi.org/10.1109/ICDE.2013.6544825

[20] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60. https://doi.org/10.14778/3421424.3421431

[21] Chunwei Liu, Matthew Russo, Michael J. Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael J. Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. *CoRR* abs/2405.14696 (2024). https://doi.org/10.48550/ARXIV.2405.14696 arXiv:2405.14696

[22] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query Rewriting in Retrieval-Augmented Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 5303–5315. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.322

[23] Christina Niklaus, Matthias Cetto, André Freitas, and Siegfried Handschuh. 2018. A Survey on Open Information Extraction. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, Emily M. Bender, Leon Derczynski, and Pierre Isabelle (Eds.). Association for Computational Linguistics, 3866–3878. https://aclanthology.org/C18-1326/

[24] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. LOTUS: Enabling Semantic Queries with LLMs Over Tables of Unstructured and Structured Data. *CoRR* abs/2407.11418 (2024). https://doi.org/10.48550/ARXIV.2407.11418 arXiv:2407.11418

[25] Michal Pietruszka, Michal Turski, Lukasz Borchmann, Tomasz Dwojak, Gabriela Nowakowska, Karolina Szyndler, Dawid Jurkiewicz, and Lukasz Garncarek. 2024. STable: Table Generation Framework for Encoder-Decoder Models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2024 - Volume 1: Long Papers, St. Julian's, Malta, March 17-22, 2024*, Yvette Graham and Matthew Purver (Eds.). Association for Computational Linguistics, 2454–2472. https://aclanthology.org/2024.eacl-long.151

[26] Swarnadeep Saha and Mausam. 2018. Open Information Extraction from Conjunctive Sentences. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, Emily M. Bender, Leon Derczynski, and Pierre Isabelle (Eds.). Association for Computational Linguistics, 2288–2299. https://aclanthology.org/C18-1194/

[27] Swarnadeep Saha, Harinder Pal, and Mausam. 2017. Bootstrapping for Numerical Open IE. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, 317–323. https://doi.org/10.18653/V1/P17-2050

[28] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR* abs/1910.01108 (2019). arXiv:1910.01108 http://arxiv.org/abs/1910.01108

[29] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Y. Levy. 2021. From Natural Language Processing to Neural Databases. *Proc. VLDB Endow.* 14, 6 (2021), 1033–1039. https://doi.org/10.14778/3447689.3447706

[30] Liang Wang, Nan Yang, Xiaolong Huang, Jiao Binxing, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text Embeddings by Weakly-Supervised Contrastive Pre-training. *Cornell University - arXiv,Cornell University - arXiv* (Dec 2022).

[31] Xueqing Wu, Jiacheng Zhang, and Hang Li. 2022. Text-to-Table: A New Way of Information Extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 2518–2533. https://doi.org/10.18653/V1/2022.ACL-LONG.180

[32] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. Corrective Retrieval Augmented Generation. *CoRR* abs/2401.15884 (2024). https://doi.org/10.48550/ARXIV.2401.15884 arXiv:2401.15884

[33] Bowen Yu, Yucheng Wang, Tingwen Liu, Hongsong Zhu, Limin Sun, and Bin Wang. 2021. Maximal Clique Based Non-Autoregressive Open Information Extraction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 9696–9706. https://doi.org/10.18653/V1/2021.EMNLP-MAIN.764

[34] Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. 2023. Large Language Model Cascades with Mixture of Thoughts Representations for Cost-efficient Reasoning. *CoRR* abs/2310.03094 (2023). https://doi.org/10.48550/ARXIV.2310.03094 arXiv:2310.03094

[35] Guineng Zheng, Subhabrata Mukherjee, Xin Luna Dong, and Feifei Li. 2018. OpenTag: Open Attribute Value Extraction from Product Profiles. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, Yike Guo and Faisal Farooq (Eds.). ACM, 1049–1058. https://doi.org/10.1145/3219819.3219839