Robust Hierarchical Structures for Document Analytics

Ruiying Ma
UC Berkeley EECS
ruiyingm@berkeley.edu

Yiming Lin
UC Berkeley EECS
yiminglin@berkeley.edu

Aditya G. Parameswaran
UC Berkeley EECS
adityagp@berkeley.edu

ABSTRACT

Document analytics is seeing a resurgence thanks to the rise of Large Language Models (LLMs). However, most document analytics systems treat documents as plain text, ignoring the fact that documents are often organized hierarchically into sections and subsections, and that this hierarchical structure improves analytics for both LLMs and humans. Inferring the true underlying structure is unrealistic due to the complexity of real-world documents. Prior work attempts this inference but provides no guarantees on how well the inferred structure matches the true one. Instead, we target a *robust* and *compact* variant that is feasible to infer and useful in practice. Robustness ensures that the text under each subsection header is a superset of the text under the same header in the true structure. Compactness seeks to minimize this superset, reducing LLM cost or human cognitive load. We propose SHED, a two-stage workflow for inferring a robust and compact structure. The first stage is pluggable with an infinite family of approaches, each guaranteeing robustness for a specific document class. We theoretically characterize the document space using these classes and their hierarchical relationships. Empirically, SHED improves robustness and compactness by up to 82.44% in F-1 score over five baselines. Finally, we propose SHED-RAG as an application of SHED for retrieval and generation (RAG), showing its usefulness by improving accuracy by up to 75.94% over four RAG baselines across four datasets.

1 INTRODUCTION

A large fraction of the world's data exists in unstructured formats, such as PDF and Word documents [20, 48]. We're seeing a resurgence of interest in processing and making sense of these documents, thanks to Large Language Models (LLMs). However, the length and complexity of such documents poses a challenge for both human and LLM understanding [4, 44]. At least for LLMs, the standard approach is to simply treat documents as a *bag of words*—that is, documents are converted into plain text using OCR (Optical Character Recognition), and then relevant subsets of this text (e.g., identified via embedding similarity) can be forwarded to LLMs for question answering or analysis [23, 43, 70, 72, 80, 82], through a process of Retrieval-Augmented Generation (RAG) [26, 36]. However, documents are typically organized hierarchically with sections and subsections—with each such header having specific human-interpretable visual patterns in the form of numbering, font, size, and capitalization. Instead of treating documents as bags of words, *reconstructing the underlying hierarchical structure of these documents can improve the accuracy of document analytics*, for both LLM and human understanding. Consider the following.

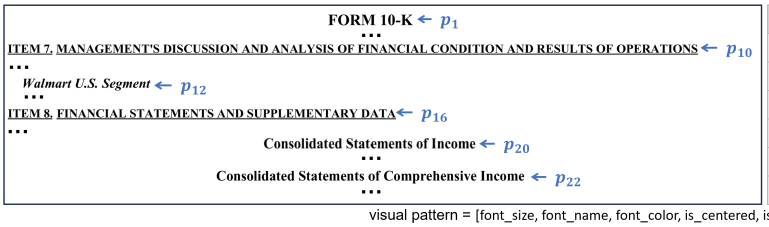
Example 1. Impact of Hierarchical Structure. Consider a Walmart Inc. 10-K document with 170 pages in Figure 1. Its hierarchical structure can be represented as a tree shown in Figure 1b, where nodes represent headers, and edges represent semantic (containment) relationships. Following Lin et al. [41], we call this tree a *Semantic Hierarchical Tree (SHT)*. For example, v_i in Figure 1b corresponds to the header phrase p_i in Figure 1a, while edge $v_{16} \rightarrow v_{20}$ indicates that the document portion headed by *Consolidated Statements of Income* (corresponding to v_{20}) is a subsection under the section titled *ITEM 8...* represented by v_{16}. Consider the question *"What is the overall FY2018–FY2020 average operating income?"*. The correct answer is in the subsection corresponding to v_{20}, with the header *Consolidated Statements of Income*, and content about *operating income*—providing this entire subsection (including the header and content) is sufficient to answer this question. However, existing RAG variants [13, 15, 18, 19, 60, 79, 86], which treat documents as bags of words, would simply retrieve the most relevant content based on embedding similarities [29, 34]. Here, they may retrieve the content underlying v_{12} and v_{20} (the two boxes in Figure 1b), since the content is similar to the question, even though v_{12} is a false positive, since the operating income information in v_{12} is for the *Walmart U.S. Segment*. Without knowing the structure—and specifically the headers—for v_{12} and v_{20}, it is impossible for the LLM to know which one is accurate. □

Similarly, human understanding of long documents hinges on SHTs [69], improving the accessibility of scientific [8], legal [14], and medical [54] documents. An SHT not only aids humans in locating which subsections they should review (top-down), but also provides them *context* for where they are in the document when reviewing a given text portion (bottom-up) [9, 35]. For example, when reading the text containing *"operating income"* under v_{20}, the headers are essential to understand what the text refers to.

Lack of guarantees in prior approaches. Past work on SHT inference from documents includes approaches based on LLMs [10, 39, 45], custom models [5, 31, 46, 73], or rules [16, 52, 53, 57, 64] (see related work for details), of which LLM-based approaches are viewed to currently be state-of-the-art. Unfortunately, *prior approaches for inferring SHTs from documents provide no real guarantees* for how well such SHTs match the "true" SHT (i.e., one labeled by humans), except in very specific settings that are rare in practice, as we will discuss later [41]. Identifying the true SHT is unrealistic due to heterogeneous layouts, semantic ambiguity, and length [82].

Robustness for SHTs. Instead of inferring the true SHT, we instead introduce a new objective of *robustness*. A robust SHT is one where, for each (section or subsection) header, we require the text "under it" to be a superset of the corresponding text in the true SHT. Robustness is essential for correctness [2, 38], as discussed below:

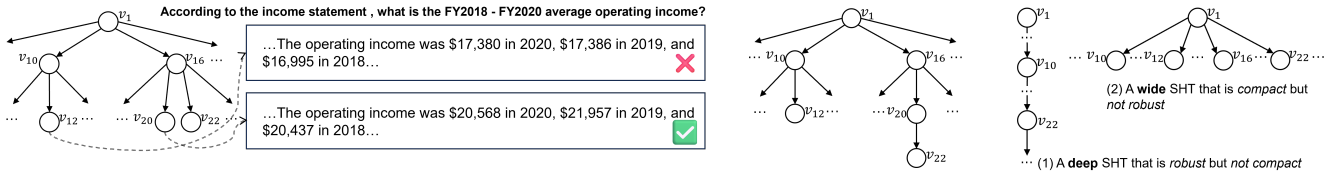
Example 2. Robust Retrieval. Figure 1c shows a robust SHT; the document portion corresponding to v_{22} is now part of that of v_{20}, as the edge $v_{16} \rightarrow v_{22}$ in the true SHT (Figure 1b) becomes a direct path connecting v_{16} to v_{22}. When attempting to answer the question in Figure 1b using this robust SHT, suppose v_{20} is retrieved and its document portion (where the correct answer resides) is returned to the LLM to generate the answer. Here, in addition to



	label	visual pattern
p_1	section_header	[32.0, Times-Roman, black, True, False, True, null]
p_{10}	section_header	[22.0, Times-Roman, black, False, True, True, numeric]
p_{12}	section_header	[18.0, Times-Italic, black, True, False, False, null]
p_{16}	section_header	[22.0, Times-Roman, black, False, True, True, numeric]
p_{20}	section_header	[20.0, Times-Bold, black, True, False, False, null]
p_{22}	section_header	[20.0, Times-Bold, black, False, False, False, null]

visual pattern = [font_size, font_name, font_color, is_centered, is_underlined, is_capitalized, list_type]

(a) The header phrases and their visual patterns.



(b) The true SHT. **(c) A robust SHT.** **(d) Deep and wide SHTs.**

Figure 1: Hierarchical structures of a financial document Walmart Inc. form 10-K. The node v_i in an SHT corresponds to the header phrase p_i.

the portion providing the correct answer (i.e., one corresponding to v_{20} in the true SHT), the extra portion corresponding to v_{22} is also returned, which is permissible as long as all true portions (the ones corresponding to v_{20}) are included—as most modern LLMs are able to filter out extraneous irrelevant content. □

Compactness for SHTs. While a robust SHT guarantees that the returned context for any given header (i.e., the text portions underneath) is a superset of the true one, we also want to *minimize the size of the returned context* to reduce either processing cost for LLMs or human cognitive load, depending on the setting—we call this property *compactness*. Consider the deep SHT in Figure 1d-1, which forms a chain of nodes with a single leaf. This tree is robust but not compact (e.g., retrieving the context of v_{10} returns all the content from p_{10} to the end of the document). In contrast, the wide SHT in Figure 1d-2, where all nodes except v_1 are in the same layer, is compact but not robust (e.g., the document portion of v_{16} no longer contains that of v_{22}, and v_{16} is no longer an ancestor of v_{22}). We therefore aim to infer SHTs ***that are guaranteed to be robust, while being as compact as possible***, ensuring correctness while reducing cost. (This is analogous to ensuring 100% recall, while maximizing precision, when retrieving document subsections.)

Inferring a robust and compact SHT via SHED (Section 3). Inferring an SHT that guarantees robustness and is compact is non-trivial. While current tools can identify headers accurately [11, 21, 30, 49, 63, 72, 77], i.e., section or subsection headers, corresponding to nodes in the tree, they fail to assemble these (header) nodes into a robust tree. We propose SHED[1], a framework for inferring a robust and compact hierarchical structure for documents.

Intuitively, headers with identical visual patterns are typically of the same type, and therefore belong in similar "levels" in the SHT, e.g., section headers are often formatted similarly, and would be at similar levels in the SHT. Taking phrases labeled as headers as input nodes, SHED employs a two-stage workflow: it first groups nodes with identical visual patterns into clusters and infers relationships among these clusters; it subsequently derives edges among nodes from the cluster relationships to assemble a compact SHT.

The first stage is *semantic depth (SD) inference*, where SHED assigns each cluster a *semantic depth*. These semantic depths may differ from the depths in the true SHT, but their relative ordering indicates hierarchical relationships: the cluster of a node's parent must have a smaller semantic depth than that of the node. We introduce two approaches for SD inference: *local-first*, which uses the header immediately prior to a cluster's first occurrence to infer its semantic depth, and *global-first*, which uses all headers prior to that occurrence. The latter generalizes Lin et al.'s approach [41], which only focuses on inferring the true SHT rather than robust variants, and only consider a narrow class of *well-formatted* documents that, as we will show in Section 4, are rare, occupying <25% of real-world documents. The second stage, *SHT assembly*, constructs a compact SHT by deriving edges from the relative ordering of the inferred semantic depths, with tree depth bounded by the number of clusters, independent of the SD inference approach used in the first stage. Empirically, ***SHED (with local-first) improves robustness and compactness by up to 82.44% in F-1 over five baselines.***

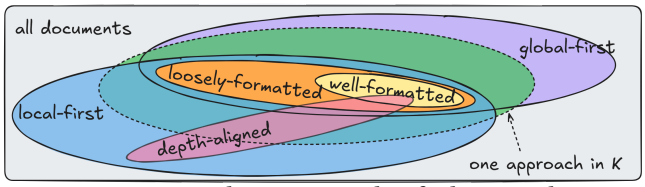


Figure 2: Our contribution: Hierarchy of robustness classes.

Characterization of Robustness Classes (Section 4). While our two-stage framework results in high robustness and compactness as we see in our experiments, we wanted to additionally characterize the space of documents for which we can guarantee robustness for each approach, and also go beyond the two SD inference approaches, local- and global-first. To do so, we establish a necessary and sufficient condition for robustness for each approach, a property we call *SD conformance*. Leveraging SD conformance, we construct a Venn-diagram of relationships for various approaches, as shown in Figure 2—this diagram shows the sets of documents for which each approach guarantees robustness. We prove that neither local- nor global-first dominate, but there is a class of documents we call *depth-aligned*, for which local-first guarantees robustness, but global-first does not. We further generalize beyond local- and global-first to *an infinite family of approaches $\mathcal{K}$, each guaranteed to return a robust*

[1] **S**emantic **H**ierarchical Structure **E**xtraction for **D**ocuments.

SHT for a specific class of documents while optimizing for compactness. We additionally define a class of *loosely-formatted* documents which lies at the intersection of all approaches in $\mathcal{K}$, including local- and global-first, and extends far beyond *well-formattedness* (by 2.57× empirically). To our knowledge, this is ***the first formal characterization of various document classes and effectiveness of the corresponding approaches***.

Application for RAG (Section 5). We then develop a RAG technique called SHED-RAG that leverages SHTs, to show the practical usefulness of robust and compact SHTs. Specifically, SHED-RAG retrieves nodes in the SHT inferred by SHED that are most relevant to a given question and organizes them into a hierarchical context for generation, leveraging both their content (e.g., corresponding document portions) and hierarchical context (e.g., headers of the nodes and their ancestors), ***achieving up to 75.94% accuracy improvement over baseline RAG approaches***.

We make the following contributions in this paper:

- We formalize the novel concepts of robustness and compactness of SHTs for document analytics and design SHED, a two-stage workflow with a first stage that is pluggable with an infinite family of approaches, outperforming five baselines by up to 82.44% in robustness and compactness.
- We theoretically characterize various classes of documents for which SHED guarantees robustness while optimizing compactness, for each SD inference approaches, organized as a hierarchy.
- We propose SHED-RAG for SHT-based RAG application, improving question answering accuracy by up to 75.94% over four RAG baselines across four datasets.

2 ROBUST SEMANTIC HIERARCHICAL TREES

We focus on documents with rich visual formatting (e.g., PDF and Word files). We first review relevant concepts related to documents, including semantic hierarchical trees, as defined in [41], in Section 2.1, and then define robust and compact variants in Section 2.2.

2.1 Preliminaries

Documents and Phrases. Given a document D, we operate on a plain text serialized representation as a list of phrases, $D = \{p_1, p_2, ..., p_n\}$, indexed based on where they appear in D. Each phrase p_i is associated with a label $l_i \in \{$`section_header`, `text`, `table`, `image`, $\ldots\}$. This labeling can be performed at no cost using open-source document recognition tools [11, 30, 49], which typically extract phrases in a top-down, left-to-right manner, consistent with the human reading order. Although different tools may define different label sets, labels such as `section_header` (which broadly refers to the headers such as titles, section and subsection headers) and `text` (representing content) are common, making it feasible to identify the hierarchical structure for a document as described below. We say a phrase is a *header phrase* if its label is `section_header`.

Visual Pattern. Humans typically infer hierarchical structures from visual features of the header phrases such as font and layout position. For example, header phrases in larger, bold fonts that are centered may indicate section headers, whereas header phrases in smaller, non-bold fonts that are not centered may indicate subsection headers. We similarly rely on the *visual patterns* of phrases, defined below, to infer hierarchical structures. For each phrase $p_i \in D$, let $vp(p_i)$ be the visual pattern associated with p_i, encoding various visual features. $vp(p_i)$ is represented as a feature vector, with features such as [`font_size`, `font_name`, `font_color`, `is_underlined`, `is_centered`, `is_capitalized`, `list_type`, $\ldots$]. Here, `font_size`, `name`, and `color` represent the style of the phrase. The remaining features capture aspects such as whether the phrase is underlined, center-aligned, fully capitalized, or part of a numeric or alphabetical list.

Semantic Hierarchical Tree. We are now ready to define a Semantic Hierarchical Tree (SHT) for a document D, given phrases $p_i \in D$, their labels l_i, and visual patterns $vp(\cdot)$.

Definition 1. Semantic Hierarchical Tree. An SHT for a document is a directed rooted tree $T = (V, E)$, with a one-to-one correspondence between the nodes V and the header phrases. Let v_i denote the node corresponding to the header phrase p_i with index i. For any two nodes $v_i, v_j \in V$, where $i < j$, v_i must precede v_j in the pre-order traversal of the tree.

Note that the node set $V = \{v_{i_1}, v_{i_2}, \ldots, v_{i_n}\}$ corresponds to a subset of the phrases $P_i \subseteq D, P_i = \{p_{i_1}, p_{i_2}, \ldots, p_{i_n}\}$, namely, only those that are header phrases, and thus the node indices $i_1, i_2, \ldots, i_n$ are not necessarily consecutive. Definition 1 additionally implies that the indices of v_i and its descendants are smaller than that of any right sibling of v_i (if any). For example, "Section 2" and all its subsection headers should appear before "Section 3". Further note that V is fixed (comprising all header phrases $p_{i_1}, \ldots, p_{i_n}$), while E is not, indicating the existence of multiple SHTs that can represent a document. We define the *true* SHT as the one that human users believe best reflects the document's hierarchical structure.

Example 3. Figure 1b shows the (partial) true SHT for Walmart Inc. form 10-K. v_{16} corresponds to p_{16}; its child v_{20} corresponds to p_{20}, a subsection header of p_{16}. The pre-order traversal preserves a strictly increasing sequence of node indices, $v_1, \ldots, v_{10}, \ldots, v_{22}$. □

Text Span. The *text span* of a node v_i, denoted by $ts(v_i)$, refers to the text portions (i.e., phrases) associated with v_i in D. Given an SHT T, $ts(v_i)$ denotes the *sequential* list of phrases starting from the header phrase corresponding to v_i and ending immediately before the header phrase corresponding to the next non-descendant node v_j in T, defined below, i.e., $ts(v_i) = [p_i, p_{i+1}, \ldots, p_{j-1}]$. Formally, node v_j is a non-descendant of v_i in T with the smallest index $j > i$. For example, the text span of v_{20} in Figure 1b is $[p_{20}, p_{21}]$, since its next non-descendant node is v_{22}.

2.2 Robust and Compact SHTs

Constructing the true SHT may be unrealistic due to semantic ambiguities in documents. A key insight of this paper is that *it is often sufficient to construct a robust SHT for downstream tasks*, rather than the true one, as defined below.

Definition 2. Robust SHT. An SHT T of a document D is *robust* if $\forall\, v_i \in V$, $ts'(v_i) \subseteq ts(v_i)$, where ts' and ts denote the text spans inferred from T' (the true SHT) and T, respectively.

A robust SHT T guarantees that the text span $ts(v_i)$ of any node v_i in T is a *superset* of that of v_i in the true SHT T'. This robustness property provides correctness guarantees for a variety of tasks. For example, when performing retrieval (e.g., as part of RAG) over a robust SHT T, the retrieved context for v_i will always include the true context for v_i in T'. This property ensures correctness of

question answering results as long as the retrieval method identifies the right header (section, subsection, ...) where the answer lies, and the LLM gives the right answer when provided a superset of the text portion under this header (i.e., its true text span).

Example 4. Figure 1c presents a robust tree T related to the true one T' in Figure 1b, where v_{22} is now a child of v_{20}. $ts(v_{22})$ contains $ts'(v_{20})$ and $ts'(v_{22})$, forming a superset of the true context $ts'(v_{20})$ that includes the answer to the question in Figure 1a. □

A robust SHT also guarantees that any parent-child pairs in the true SHT remain as ancestor-descendant pairs, a property we will use for SHT inference in the following sections. The formal statement is given below:

Theorem 1. Consider an SHT T of a document D. T is robust *if and only if* for every pair of nodes v_i, v_j where v_j is a child of v_i in the true SHT T', v_j is a descendant of v_i in T.

Proof. ($\Leftarrow$) Let T' be the true SHT. Any parent-child pair in T' remains an ancestor-descendant pair in SHT T. Suppose v_k and $v_{k'}$ are the next non-descendant of v_i in T and T', respectively. For $v_{k'-1}$, either $v_{k'-1} = v_i$, or it is the last descendant of v_i in T'. For the former case, $k' - 1 = i < k$. For the latter case, since v_i is a parent of $v_{k'-1}$ in T', it is also an ancestor of $v_{k'-1}$ in T. Therefore, $v_{k'-1}$ is a descendant of v_i in T, indicating $k' - 1 < k$. Thus $ts'(v_i) = [p_i, \ldots, p_{k'-1}] \subseteq [p_i, \ldots, p_{k-1}] = ts(v_i)$. T is robust.

($\Rightarrow$) For each parent-child pair (v_i, v_j) in the true SHT T', $p_j \in ts'(v_i)$. Since T is robust, we have $ts'(v_i) \subseteq ts(v_i)$. Therefore, $p_j \in ts(v_i)$, indicating that v_i is an ancestor of v_j in T. □

While we want T to be robust, at the same time, we must optimize T for efficiency by minimizing the text span of each node. For example, in retrieval for RAG using T, the retrieved context for node v_i is its text span $ts(v_i)$ in T. While $ts(v_i)$ must include the true text span $ts'(v_i)$, it should also contain as little excess content $ts(v_i) \backslash ts'(v_i)$ as possible. Therefore, to minimize the text span of each node of T while preserving robustness, we want to minimize $\mathrm{C} = \sum_{v_i \in V} ts(v_i) \backslash ts'(v_i)$; we refer to C as *compactness*.

We can now formally define our problem.

Definition 3. Robust and Compact SHT Inference. Given a document $D = \{p_1, \ldots, p_n\}$, where each phrase p_i has a label l_i and visual pattern $vp(p_i)$, infer an SHT T from D such that T is robust, while minimizing compactness $\mathrm{C} = \sum_{v_i \in V} ts(v_i) \backslash ts'(v_i)$.

Example 5. For the SHTs of the Walmart Inc. form 10-K, we prefer Figure 1c over the deep and wide SHTs in Figure 1d. The wide SHT is not robust: $ts(v_{16})$ incorrectly excludes $ts'(v_{20})$. The deep SHT is not compact: $ts(v_{20})$ unnecessarily includes all content from p_{20} to the end of D. In contrast, Figure 1c achieves robustness, while also being more compact. For robustness, $ts(v_{16})$ is a superset of its true text span, including $ts'(v_{20})$. For compactness, $ts(v_{20})$ includes only the small text span of v_{22} in addition to $ts'(v_{20})$. □

3 SHT INFERENCE: TWO APPROACHES

In this section, we present our two-stage workflow, SHED, for inferring a robust and compact SHT $T = (V, E)$ from a document D, starting with an overview in Section 3.1. As mentioned earlier, the set of nodes V correspond to the set of header phrases. To assemble V into an SHT T, we rely on the intuition that the semantic hierarchy represented by an SHT depends on the *visual patterns* of nodes $vp(\cdot)$. Recall that these visual patterns are represented by feature vectors that encode various visual features (e.g., `font_size`, `font_name`) of the phrases corresponding to the nodes.

3.1 Overall Workflow

The overall workflow of SHT inference consists of two stages:
1) Semantic depth (SD) inference (Section 3.2 and Section 3.3). The semantic relationships between nodes (i.e., the edges in the SHT) closely correspond to their visual patterns: certain visual patterns (e.g., those of section headers) tend to indicate parent nodes, while others (e.g., those of subsection headers) indicate children. Therefore, we first cluster the nodes based on their visual patterns—so each visual pattern corresponds to a cluster—and then infer what we refer to as the *semantic depth* of each cluster. Semantic depth does not directly specify the exact "level" of nodes with the given visual pattern in the SHT; instead, it constrains semantic relationships among nodes with different visual patterns (i.e., edges in the SHT), as described below.
2) SHT assembly (Section 3.4). We then build the SHT T by inferring edges based on semantic depths of the clusters. Specifically, we use the *relative ordering* of semantic depths to determine the allowed semantic relationships: the parent of a node in the SHT must belong to a cluster with a *smaller* semantic depth.

As shown in Figure 3, there are multiple approaches for SD inference $\texttt{Infer-SD}_1$, $\texttt{Infer-SD}_2$, ...; the approaches in Sections 3.2 and 3.3 correspond to $\texttt{Infer-SD}_1$, which we refer to as local-first, and $\texttt{Infer-SD}_2$, which we refer to as global-first, respectively. Both local- and global-first approaches rely on the node with the smallest index in each cluster to determine the semantic depth of each cluster. This node, which we call the *indexing node* of the cluster, corresponds to the first occurrence of the visual pattern associated with the cluster in the document. The local-first approach infers the semantic depth of each cluster using *local* structural cues, examining only the node whose header phrase immediately precedes that of the cluster's indexing node. In contrast, global-first uses *global* structural cues by considering all nodes whose header phrases precede that of the indexing node. Within our two-stage workflow, both approaches produce SHTs that are correct and compact, as we will establish in Section 3.5.

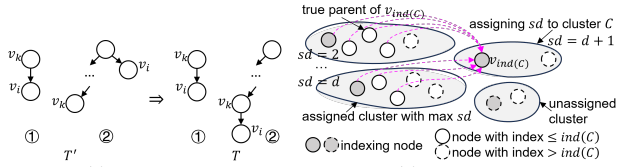
The robustness of the SHTs inferred by local- and global-first approaches will be analyzed in Sections 4.1 and 4.2. We further combine local- and global-first into an infinite family of approaches in Section 4.3. Each approach in this family also ensures SHT robustness for a particular class of documents, as shown in Section 4.4. It is worth noting that the global-first approach is inspired by the SHT inference algorithm in ZenDB [41], which is designed for a narrowly defined class of documents termed *well-formatted* in [41]. We show in Section 4.4 that well-formatted documents constitute only a small subset of real-world documents, and the global-first approach, as a generalization within our two-stage framework, infers robust SHTs for a substantially broader class of documents.

We begin by discussing our first SD inference approach.

3.2 Semantic Depth Inference: Local-First

In local-first SD inference, we start by clustering the header nodes V, based on their visual patterns, into clusters $C = \{C_1, \ldots, C_m\}$. Two

Figure 3: The overall workflow of SHED for SHT Inference.



(a) Local-first. **(b) Global-first.**

Figure 4: Intuition of SD Inference.

Algorithm 1 `Infer-SD`$_1(V, C)$ // Local-First

1: $sd(C_1) \leftarrow 0$
2: **for** $C \in C \backslash \{C_1\}$ in ascending order of $ind(C)$ **do**
3: $\quad v_k \leftarrow \arg\max_{v_i \in V, i < ind(C)} : i$
4: $\quad sd(C) \leftarrow sd(C(v_k)) + 1$
5: **return** sd ▷ *a mapping from a cluster to its semantic depth*

nodes $v_i, v_j \in V$ belong to the same cluster if they share identical visual patterns, i.e., $vp(p_i) = vp(p_j)$. For each cluster C_j, we will infer its semantic depth, denoted $sd(C_j)$. We assume that cluster C_1 contains only the root node v_1 and set its $sd(C_1) = 0$.

Next, we describe the inference of $sd(C)$ for each subsequent cluster. We leverage the local-first property, i.e., inferring semantic depth using the node that immediately precedes the indexing node of each cluster, as detailed below. We infer $sd(C_j)$ from the node v_i whose header phrase p_i is the *first* instance of the visual pattern corresponding to C_j encountered in the document. Formally, let $i = ind(C_j)$ denote the *index of a cluster*, defined as the smallest index among the indices of the nodes belonging to C_j. We call $v_{ind(C)}$ the *indexing node* of C. For example, if a cluster $C_2 = \{v_{10}, v_{16}\}$, its index is $ind(C_2) = 10$, and v_{10} is its indexing node. We use $C(v)$ to denote the cluster that contains node v.

For the indexing node v_i of a cluster C_j, suppose v_k is the node with the largest index $k < i$. In other words, v_k immediately precedes v_i in the pre-order traversal of the true SHT T', indicating one of the following two cases, as shown in Figure 4a: ① v_k is the true parent of v_i, or ② v_k is a descendant of the true parent of v_i in T'—but we are unaware of which case we are in. Recall that, to ensure robustness, we require the true parent of v_i to remain an ancestor in the inferred SHT, as stated in Theorem 1. In both cases, if we move v_i to be a child of v_k, the true parent of v_i in T' remains an ancestor of v_i in T, ensuring robustness. Based on this intuition, we assign $C(v_i)$ a larger semantic depth than $C(v_k)$. Note however the latter case ② may lead to reduced compactness, but we accept this lack of compactness as the price for ensuring robustness.

Algorithm 1 shows local-first SD inference. Starting with $sd(C_1) = 0$ (Line 1), we scan the clusters in increasing order of their indices (Line 2-4). For cluster C, we find v_k with the largest index $k < ind(C)$ (Line 3). We then set $sd(C)$ to be $sd(C(v_k)) + 1$ to ensure robustness of the SHT inferred subsequently. The scanning order of Algorithm 1 ensures that $sd(C(v_k))$ has already been inferred when computing $sd(C)$, since $ind(C(v_k)) \leq k < ind(C)$ and $C(v_k)$ is scanned before C (Line 2).

Algorithm 2 `Infer-SD`$_2(V, C)$ // Global-First

1: $sd(C_1) \leftarrow 0$
2: **for** $C \in C \backslash \{C_1\}$ in ascending order of $ind(C)$ **do**
3: $\quad v_k \leftarrow \arg\max_{v_i \in V, i < ind(C)} : sd(C(v_i))$
4: $\quad sd(C) \leftarrow sd(C(v_k)) + 1$
5: **return** sd

Example 6. Figure 5a illustrates Algorithm 1 using only the nodes explicitly shown in Figure 1c, with all other nodes hidden for simplicity. For example, to infer sd for C_4, we identify the node $v_k = v_{16}$ immediately prior to its indexing node v_{20} and assign $sd(C_4) = sd(C(v_{16})) + 1 = sd(C_2) + 1 = 2$. □

3.3 Semantic Depth Inference: Global-First

Next, we provide an alternate global-first approach for SD inference[2] (Algorithm 2). Like Algorithm 1, Algorithm 2 assigns semantic depths to clusters based on their indices: clusters with smaller indices (i.e., visual patterns whose first occurrence appears earlier in the document) have smaller semantic depths. Formally, after sorting the clusters in increasing order of their indices (Line 2), Algorithm 2 assigns each cluster C a semantic depth that is one greater than the largest semantic depth among all preceding clusters (Line 3-4).

In effect, the semantic depth of each cluster C is determined *globally* via its indexing node $v_{ind(C)}$: Algorithm 2 considers *all* nodes with smaller indices than $v_{ind(C)}$. We don't know which of these nodes is the true parent of $v_{ind(C)}$ (shown as dashed lines in Figure 4b), so we opt for the worst case and set $sd(C)$ to be one larger than the largest semantic depth among their clusters, ensuring that $sd(C)$ is greater than the semantic depth of the cluster of the true parent of $v_{ind(C)}$, as illustrated in Figure 4b.

Example 7. Using the same nodes and clusters as in Figure 5a, Algorithm 2 infers $sd(C_i) = i - 1$ for each $i \in [1, 5]$, with $sd(C_i) < sd(C_j)$ whenever $i < j$, as shown in Figure 5b. □

3.4 SHT Assembly

Given nodes V and inferred semantic depths sd of their clusters C (using local- or global-first), we now assemble T by inferring the parent of each non-root node in V.

We infer the parents of nodes in increasing order of node indices, just like how humans make sense of the structure of a document by reading the document sequentially. Suppose we have a partial SHT T containing all nodes with indices smaller than i, and we now infer the parent v_k of node v_i. Let v_j denote the true parent of v_i (which is unknown). Using the information available (i.e., V, C, and sd), we can derive conditions that the true parent v_j must satisfy, so that selecting v_k ensures robustness while optimizing compactness $\mathsf{C} = \sum_{v \in V} ts(v) \backslash ts'(v)$. For simplicity, we assume that T is the (partial) true SHT, i.e., T matches the true SHT so far. We define *candidate parents* of v_i as the nodes that satisfy conditions **1)** and **2)** as stated below; this set must include the true parent v_j. We then select v_k from this set to ensure robustness and optimize compactness, without knowing v_j.

1) Coverage of Text Span: $p_i \in ts(v_j)$; i.e., the text span of v_j inferred from T must contain the header phrase p_i of v_i (e.g., a

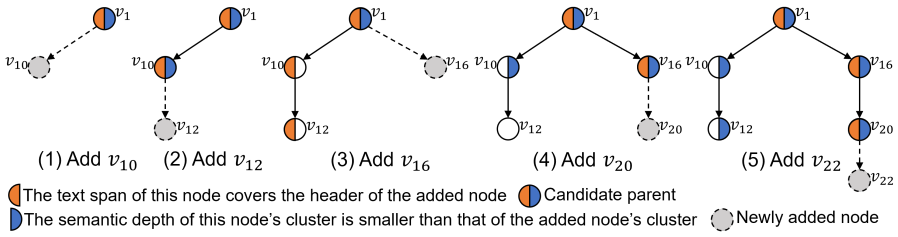
[2]As mentioned earlier, Algorithm 2 generalizes ZenDB's algorithm to work on non-well-formatted documents, and leverages the two-stage approach, as we will establish in Section 4.5.

iteration	-	1	2	3	4
scanned cluster	C_1	C_2	C_3	C_4	C_5
nodes	$\{v_1\}$	$\{v_{10}, v_{16}\}$	$\{v_{12}\}$	$\{v_{20}\}$	$\{v_{22}\}$
$ind(C)$	1	10	12	20	22
v_k	-	v_1	v_{10}	v_{16}	v_{20}
$C(v_k)$	-	C_1	C_2	C_2	C_4
$sd(C)$	0	1	2	2	3

(a) Algorithm 1

-	1	2	3	4
C_1	C_2	C_3	C_4	C_5
$\{v_1\}$	$\{v_{10}, v_{16}\}$	$\{v_{12}\}$	$\{v_{20}\}$	$\{v_{22}\}$
1	10	12	20	22
-	v_1	v_{10}	v_{12}	v_{20}
-	C_1	C_2	C_3	C_4
0	1	2	3	4

(b) Algorithm 2



(c) Algorithm 3

Figure 5: An illustration of SHT inference.

Algorithm 3 `Assemble-SHT`(V, C, sd)

1: $T \leftarrow$ an empty tree
2: **for** $v_i \in V$ in ascending order of their indices **do**
3: $\quad v_k \leftarrow \arg\max_{v_j \in T, p_i \in ts(v_j), sd(C(v_j)) < sd(C(v_i))} : j$
4: $\quad$ Add node v_i to T as the rightmost child of v_k
5: **return** T

subsection header is contained within its parent section). Since all nodes in T have indices smaller than i, we have $j < i$, and $p_i \in ts(v_j)$ further implies that v_j lies on the *rightmost path* of T, as described below, with its text span extending to the end of the document: $ts(v_j) = [p_j, \ldots, p_n]$, where $j < i < n$ and p_n denotes the final phrase of the document. The rightmost path of T starts from the root and follows the rightmost child at each node down to a leaf. In Figure 5c(5), $[v_1, v_{16}, v_{20}, v_{22}]$ is the rightmost path.

From **1)** alone, the candidate parents for v_i are the nodes on the rightmost path of T. We can therefore select v_k as the candidate with the largest index (i.e., the rightmost leaf of T), inserting v_i at the end of the rightmost path, guaranteeing that the true parent v_j remains an ancestor of v_i, and ensuring robustness.

However, this choice of v_k yields worst-case compactness: $ts(v_i)$ is added to the text span of every node on the rightmost path, contributing r times to compactness C, where r is the number of candidate parents. To reduce r and improve compactness, we introduce **2)** to further narrow the set of candidate parents.

2) Semantic Depth Constraint: $sd(C(v_j)) < sd(C(v_i))$**.** Recall that semantic depths of clusters constrain semantic relationships between nodes: the semantic depth of a parent's cluster must be smaller than that of the child's cluster, i.e., $sd(C(v_j)) < sd(C(v_i))$.

Condition **2)** uses semantic depths sd to further restrict the set of candidate parents to nodes on the rightmost path of T whose clusters have smaller semantic depths than $C(v_i)$. This set still contains the true parent v_j while reducing the number of candidate parents r. Selecting v_k as the candidate parent with the largest index preserves robustness of T, since both v_k and v_j lie on the rightmost path with $k \geq j$, implying that v_k is equal to or a descendant of v_j. This choice of v_k also improves compactness of T, as the reduced set of candidate parents leads to a smaller r than using **1)** alone.

Using the two above conditions, we can now present the algorithm to assemble an SHT T in Algorithm 3. We consider the nodes in increasing order of their indices (e.g., from v_1, v_{10} to v_{22}) (Line 2). In each iteration, we insert v_i into T as the *rightmost child* of the selected parent v_k (Line 4), so that the pre-order traversal of T remains consistent with the nodes' indexing order and thus with the document's reading order. As mentioned earlier, v_k is selected as the candidate parent of v_i with the largest index, from among all v_j in T that satisfy: **1)** $p_i \in ts(v_j)$, and **2)** $sd(C(v_j)) < sd(C(v_i))$. Note that the text span $ts(v_j)$ of each node v_j evolves as new nodes are inserted T. Therefore, in Line 3, $ts(v_j)$ denotes the text span of v_j inferred from the partial tree T at that point in the algorithm.

Example 8. Figure 5c illustrates Algorithm 3 using the nodes, clusters, and semantic depths from Figure 5a. When inserting v_{20} (step (4)), the candidate parents—nodes in T with $sd < sd(C(v_{20})) = 2$ (blue) and text spans covering p_{20} (orange)—are v_1 and v_{16}. Since v_{16} has the larger index, v_{20} is inserted as its rightmost child. □

3.5 Correctness and Compactness

We now establish that our two-stage workflow correctly generates an SHT. We also show that the resulting SHT has good compactness. Correctness and compactness hold for any document and are independent of the SD inference approach. With respect to robustness, different SD inference approaches apply to different classes of documents. We analyze these document classes in Section 4.

3.5.1 SHT Correctness. We can formally show that the tree T inferred by our two-stage workflow is an SHT:

Theorem 2. The tree T output by Algorithm 3 is an SHT.

Proof. When inserting v_i, all nodes whose text spans cover the header phrase of v_i lie on the *rightmost path* of T (e.g., in Figure 5c, all nodes highlighted in orange are on the rightmost path at each insertion). v_i is then inserted as the rightmost child of a node on this rightmost path, and therefore appears *last* in the pre-order traversal of T. By induction, after every iteration of Algorithm 3, the pre-order traversal of T remains consistent with the increasing order of the node indices. □

3.5.2 SHT Compactness. We now show that SHTs inferred by our workflow have good compactness (irrespective of algorithm).

While true compactness $\mathrm{C} = \sum_{v \in V} ts(v) \backslash ts'(v)$ is hard to quantify without the true SHT T', a suitable proxy is the depth of the inferred SHT T, defined as the number of nodes on the longest root-to-leaf path. Deeper SHTs tend to be less compact. Intuitively, consider nodes v_i and v_j, where v_j immediately precedes v_i in T (i.e., the largest index $j < i$) and lies at the end of the longest path p of T. Moving the subtree rooted at v_i to be a child of v_j preserves pre-order traversal and robustness of T, but increases the text spans of all original nodes on p prior to the extension. As a result, T becomes deeper and less compact.

Regardless of the SD inference approach used, the depth of an SHT T inferred by Algorithm 3 is bounded by the number of clusters (i.e., the number of distinct visual patterns among headers), since the path from any node to the root consists of strictly decreasing

semantic depths, terminating at zero (i.e., the semantic depth of the root's cluster). In contrast, a deep SHT T_d that links all nodes into a single chain (Figure 1d-1) has depth equal to the number of tree nodes (i.e., headers). Since the number of clusters is usually much smaller than the number of tree nodes, T is more compact than T_d. In our experiments on 592 real-world documents in Table 1 (described later), where the numbers of headers (i.e., the depths of T_d) average 112.32 with a maximum of 2,077, Algorithm 3 (with local-first) generates SHTs T with an average depth of only 5.53 and a maximum depth of only 17.

4 ROBUSTNESS ANALYSIS AND HIERARCHY

In this section, we analyze robustness of SHTs from different SD inference approaches. We first identify document classes that ensure robustness for arbitrary semantic depths in Section 4.1. These classes satisfy a property, we call *semantic depth conformance*, which depends on the inferred semantic depths; different semantic depths therefore induce different document classes where robustness is met. Next, in Section 4.2, we compare robustness guarantees of local- (Section 3.2) and global-first (Section 3.3) SD inference. We also identify a document class for which local-first guarantees robustness while global-first does not. Then, in Section 4.3, we generalize local- and global-first by combining them into an infinite family of SD inference approaches. In Section 4.4, we summarize all proposed document classes by showing hierarchical relationships among them via a Venn diagram. Finally, in Section 4.5, we illustrate the relationship between our SHT inference workflow and ZenDB.

4.1 Semantic Depth Conformance

Consider a document D, its true SHT T', and node clusters C. We now establish a property of a document under which the SHT T returned by Algorithm 3 is guaranteed to be robust for any given assignment of semantic depths sd to clusters.

To ensure robustness of T, Theorem 1 requires that every parent-child pair (v_j, v_i) in the true SHT T' remains an ancestor-descendant pair in T, which can be achieved by Algorithm 3 only if $sd(C(v_j)) < sd(C(v_i))$, a cluster-level relationship encoded by the relative ordering of semantic depths, as enforced by condition **2)** in Section 3.4. In other words, for any true parent-child pair in T', the parent's cluster must have a smaller semantic depth than the child's cluster, a document property we refer to as *semantic depth conformance* and formalize in Theorem 3, so that Algorithm 3 permits the parent to remain as an ancestor of the child in T. But, perhaps surprisingly, this property is sufficient to also guarantee robustness.

For convenience, we define $sd(v) = sd(C(v))$ for any node $v \in V$.

Theorem 3. SD Conformance Equals SHT Robustness. Consider a document D, with nodes V and clusters C. Let T be the SHT inferred by Algorithm 3, and T' be the true SHT. Given semantic depths sd of the clusters inferred by an arbitrary SD inference approach, T is robust *if and only if* $\forall\, v_i, v_j \in V$, whenever v_j is the parent of v_i in T', it holds that $sd(v_j) < sd(v_i)$.

Proof. ($\Rightarrow$). If T is robust, v_j is an ancestor of v_i in T. Algorithm 3 ensures that for any node, the semantic depth of its cluster is larger than that of the cluster containing its parent. Applying this property along the ancestor chain from v_i up to v_j in T gives $sd(v_j) < sd(v_i)$.

($\Leftarrow$). We prove by induction that each time Algorithm 3 inserts a node v_i, it places v_i as a descendant of its true parent v_j. Consequently, the tree T remains robust throughout the execution of the algorithm. The key step is to show that, at the moment v_i is inserted, *its true parent v_j is a candidate parent of v_i in Algorithm 3.* Since $j < i$ and $sd(v_j) < sd(v_i)$, it suffices to show that $ts(v_j)$ covers the header phrase of v_i at the point when v_i is inserted, which is equivalent to showing that v_j lies on the rightmost path of T. Indeed, all nodes inserted into T after v_j and before v_i are placed as descendants of v_j in T: they are descendants of v_j in the true SHT, and remain descendants of v_j in T by our induction hypothesis that T is robust before inserting v_i. Therefore, when v_i is inserted, v_j remains on the rightmost path of T. □

Note that when semantic depths are inferred using global-first, SD conformance reduces to requiring $ind(C(v_j)) < ind(C(v_i))$ for any parent-child pair (v_j, v_i) in the true SHT, because the ordering of semantic depths is consistent with the indices of the corresponding clusters.

4.2 Local vs. Global-First Depth Inference

To compare local- and global-first approaches, we first distinguish them using two concrete examples in Section 4.2.1: one where local-first infers a robust SHT while global-first fails, and vice versa. Then, in Section 4.2.2, we characterize a class of documents for which local-first guarantees robustness, whereas global-first does not.

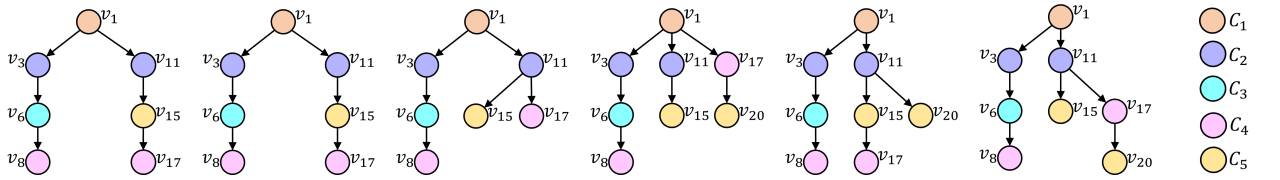
4.2.1 Illustrative Scenarios. Consider documents D_1, D_2 with true SHTs shown in Figures 6a and 6d, respectively. The clusters are indicated by colors in Figure 6g: $C_1 = \{v_1\}, C_2 = \{v_3, v_{11}\}, C_3 = \{v_6\}, C_4 = \{v_8, v_{17}\}$, and $C_5 = \{v_{15}\}$ for D_1 or $C_5 = \{v_{15}, v_{20}\}$ for D_2. For both documents, local-first assigns semantic depths $sd_1(C_{1,2,3,4,5}) = 0, 1, 2, 3, 2$, while global-first assigning $sd_2(C_{1,2,3,4,5}) = 0, 1, 2, 3, 4$.

In D_1, SD conformance required by Theorem 3 holds for sd_1, so local-first infers a robust SHT (Figure 6b), whereas global-first fails: for parent-child pair (v_{15}, v_{17}) in the true SHT in Figure 6a, since $sd_2(v_{15}) = ind(C(v_{15})) = 5$ is greater than $sd_2(v_{17}) = ind(C(v_{17})) = 4$, sd_2 violates SD conformance, and thus v_{15} is no longer an ancestor of v_{17} in the inferred SHT in Figure 6c.

In D_2, SD conformance holds for sd_2 instead, so global-first infers a robust SHT (Figure 6f), while local-first fails: for parent-child pair (v_{17}, v_{20}) in the true SHT in Figure 6d, since $sd_1(v_{17}) = 3 > sd_1(v_{20}) = 2$ violates SD conformance, v_{17} is no longer an ancestor of v_{20} in the inferred SHT in Figure 6e.

4.2.2 A Document Class Favoring Local-First Inference. We next identify a document class for which local-first guarantees robustness, while global-first does not. In this class of documents, the indexing node of each cluster must be the *leftmost child* of its parent in the true SHT (i.e., only case ① is allowed to occur in Figure 4a, while case ② is disallowed). In other words, when a new header formatting is introduced, it must be the first such header in a parent section. Under this condition, the cluster corresponding to the child indexing node is assigned a semantic depth exactly one greater than that of its true parent's cluster by Algorithm 1.

These documents further require clusters to be *level-unique*, meaning that all nodes within the same cluster are at the same



(a) True SHT (b) Algorithm 1 (c) Algorithm 2 (d) True SHT (e) Algorithm 1 (f) Algorithm 2 (g) Clusters

Figure 6: An example in which local-first generates a robust SHT while global-first does not (Figures 6a, 6b, 6c), and vice versa (Figures 6d, 6e, 6f). Colors indicate the cluster membership of each node in Figure 6g.

level in the true SHT. As a result, the semantic depth assigned by local-first matches the true depth of every node (i.e., the number of edges from the root to the node) in the true SHT, guaranteeing SD conformance and thus robustness. We refer to such documents as *depth-aligned documents*:

Definition 4. Depth-Aligned Documents. A document D, with nodes V, clusters C, and true SHT T' is *depth-aligned* if 1) $\forall\, C \in C$, $v_{ind(C)}$, whenever it is not the root, is the leftmost child of its parent in T', and 2) for any $v_i, v_j \in V$ such that $C(v_i) = C(v_j)$, v_i and v_j are at the same level in T'.

For example, Figure 6a is the true SHT of a depth-aligned document. Therefore, the SHT inferred by local-first is robust, as shown in Figure 6b. Formally, we have:

Theorem 4. Consider a document D, with nodes V, clusters C, and true SHT T'. Let sd denote the semantic depths inferred by local-first SD inference, and let T denote the SHT inferred by Algorithm 3 from sd. If D is depth-aligned, then $\forall\, v_i \in V$, $sd(v_i)$ equals the depth of v_i in T', and T is robust.

Proof. At the moment before inserting node v_i into T, assume that all nodes currently in T have semantic depths equal to their true depths in T'. If v_i is *not* the indexing node of its cluster, then by condition 2) of depth alignment, $sd(v_i)$ equals the true depth of the indexing node of $C(v_i)$, which in turn equals the true depth of v_i in T'. If v_i is the indexing node of its cluster, then by condition 1) of depth alignment, $sd(v_i) = sd(v_j) + 1$, where v_j is the true parent of v_i in T'. By our assumption that $sd(v_j)$ equals the true depth of v_j in T', it follows that $sd(v_i)$ also equals its true depth in T'. □

Although depth alignment ensures that local-first produces robust SHTs, global-first may fail on such documents. Figure 6c shows a non-robust SHT produced by global-first for a depth-aligned document whose true SHT is shown in Figure 6a.

4.3 Semantic Depth Inference Family

We can further generalize SD inference by combining local- and global-first approaches into *an infinite family of approaches*.

Recall that local-first assigns $sd_1(C)$ to be one greater than sd_1 of the cluster corresponding to the node immediately prior to its indexing node (Algorithm 1), while global-first scans all nodes prior to its indexing node and assigns $sd_2(C)$ as one greater than the largest sd_2 among their clusters (Algorithm 2).

We extend these two approaches by combining sd_1 and sd_2 linearly as $sd(C) = (K(C) \cdot sd_1 + (1 - K(C)) \cdot sd_2) + 1$, where $K(C) \in [0, 1]$ is a cluster-specific parameter. Setting $K(C) = 1(0)$ for all clusters recovers the local-first (global-first) approach. Arbitrary mappings $K : C \to [0, 1]$ define an infinite family of SD inference approaches $\mathcal{K}$, each denoted as `Infer-SD`$_K$, as shown in Algorithm 4.

Algorithm 4 `Infer-SD`$_K(V, C)$

```
sd(C_1) ← 0
for C ∈ C\{C_1} in ascending order of ind(C) do
    v_{k_1} ← arg max_{v_i ∈ V, i < ind(C)} : i
    sd_1 ← sd(C(v_{k_1}))            ▷ inferred by the local-first approach
    v_{k_2} ← arg max_{v_i ∈ V, i < ind(C)} : sd(C(v_i))
    sd_2 ← sd(C(v_{k_2}))            ▷ inferred by the global-first approach
    sd(C) ← K(C) · sd_1 + (1 − K(C)) · sd_2 + 1
return sd
```

`Infer-SD`$_K$ guarantees SHT robustness whenever the input document satisfies SD conformance, i.e., in the true SHT of D, the inferred semantic depth of each parent's cluster is smaller than that of its child's cluster (Theorem 3). Because SD conformance depends on the semantic depths sd inferred by `Infer-SD`$_K$, different Ks induce different sds and thus different conformance conditions. Therefore, the document classes that are necessary and sufficient to guarantee robustness for `Infer-SD`$_K$ vary across different parameter mappings K. In Section 4.4, we identify document class, termed *loosely-formatted* documents, which lies at the intersection of all such document classes, guaranteeing robustness for all approaches in $\mathcal{K}$ induced by varying the mappings K.

4.4 Hierarchy of Robustness

In this section, we study two additional document classes: *well-formatted documents* proposed by Lin et al. [41], and what we term *loosely-formatted documents*, which encompasses well-formatted documents and ensures SHT robustness for the family of SD inference approaches, $\mathcal{K}$, introduced in Section 4.3. We then summarize the relationships between all document classes considered thus far in the form of a hierarchy in Figure 2 and estimate their frequencies, as well as document classes that satisfy SD conformance for different approaches in $\mathcal{K}$.

Well-formatted Documents. In a well-formatted document, any pair of sibling nodes (i.e., nodes with the same parent) in the true SHT T' belongs to the same cluster. Additionally, for any parent-child node pair in T', this relationship (i.e., which node is the parent and which is the child) is consistent with the parent-child relationship observed at the first occurrence of their corresponding visual patterns[3]. Concretely, in T', if v_i is the parent of v_j, then, at the first occurrence of the visual pattern corresponding to v_j (i.e., at node v_l, the indexing node of $C(v_j)$), v_l's parent v_k has a visual pattern that matches v_i. Here, (v_k, v_l) is the first parent-child occurrence of the visual patterns corresponding to (v_i, v_j).

[3]This statement is equivalent to ZenDB's original definition of well-formattedness, which we will establish in Section 4.5.

For well-formatted documents, any approach in $\mathcal{K}$ produces robust SHTs after applying Algorithm 3. However, these documents constitute a small fraction of real-world documents, e.g., 23.39% across the 592 documents in Table 1 (described later). So, instead of restricting robustness analysis to this narrow class, we define a broader document class below, called *loosely-formatted documents*, and prove that they also guarantee robustness for all approaches in $\mathcal{K}$ in Theorem 6. Since well-formatted documents are a special case of loosely-formatted documents (see Theorem 5 below), their robustness guarantees follow immediately.

Loosely-formatted Documents. These documents relax well-formattedness by removing the requirement that siblings belong to the same cluster. And the requirement that each parent-child relationship in the true SHT matches the first *parent-child* occurrence of the corresponding visual patterns is replaced by the weaker requirement that it matches the first *ancestor-descendant* occurrence of the corresponding visual patterns. Concretely, in T', if v_i is the parent of v_j, then v_l must have an *ancestor* v_k whose visual pattern is same as that of v_i, where (v_k, v_l) is the first ancestor-descendant occurrence of the visual patterns corresponding to (v_i, v_j). For example, for the node pair $(v_i, v_j) = (v_1, v_{16})$ in Figure 1b, their corresponding visual patterns first appear as an ancestor-descendant pair at $(v_k, v_l) = (v_1, v_{10})$. This ancestor-descendant relationship correctly indicates the parent-child relationship between v_1 and v_{16}.

Definition 5. Loosely-Formatted Documents. A document D, with nodes V, clusters C, and true SHT T' is *loosely-formatted* if $\forall\ v_i, v_j \in V$, when v_i is the parent of v_j in T', there exists a pair of nodes $v_k, v_l \in V$ such that v_k is an ancestor of v_l in T' with $l = ind(C(v_j))$ and $C(v_k) = C(v_i)$.

As mentioned earlier, loosely-formatted documents relax well-formattedness, as stated below:

Theorem 5. Every well-formatted document is also loosely-formatted, but the converse does not hold.

Proof. If D is well-formatted, then $\forall\ v_i, v_j \in V$ where v_i is the parent of v_j in T', v_l's parent v_s in T' satisfies $C(v_s) = C(v_i)$, where $l = ind(C(v_j))$. Choosing $v_k = v_s$ satisfies all conditions in Definition 5, and thus D is loosely-formatted. The converse does not hold because well-formattedness does not allow parent-child relationships to be inferred from ancestor-descendant occurrences. □

We further establish that applying our two-stage workflow with any SD inference approach from $\mathcal{K}$ on loosely-formatted documents guarantees robustness:

Theorem 6. Loose-Formattedness Implies SHT Robustness. Let T be the SHT inferred by Algorithm 3 after applying any approach from $\mathcal{K}$ for SD inference on a document D. If D is loosely-formatted, then T is robust.

Proof. Let the approach from $\mathcal{K}$ be $\texttt{Infer-SD}_K$ with an arbitrary parameter mapping K. D is loosely-formatted. We prove this theorem by contradiction: suppose that the inferred SHT T is not robust. By Theorem 3, there must exist a parent-child pair (v_i, v_j) in the true SHT T' that violates SD conformance, i.e., $sd(v_i) \geq sd(v_j)$. Among all such violating pairs, choose (v_i, v_j) with the smallest child index j. v_{k_1}, v_{k_2}, sd_1 and sd_2 are inferred by Algorithm 4 while scanning the cluster $C(v_j)$ of v_j. We first establish two inequalities:
1) $sd(v_i) \leq sd_1$. If $v_{k_1} = v_i$, the claim is immediate. Otherwise, v_{k_1} is a true descendant of v_i in T'. By the minimality of j, SD conformance holds for (v_i, v_{k_1}), implying $sd(v_i) < sd(v_{k_1}) = sd_1$.
2) $sd(v_i) \leq sd_2$. This follows directly from $i < j$ and the definition of sd_2.
We now consider v_j.
Case (a): v_j is the indexing node of its cluster. By Algorithm 4, $sd(v_j) = K(C(v_j)) \cdot sd_1 + \big(1 - K(C(v_j))\big) \cdot sd_2 + 1$ for some $K(C(v_j)) \in [0, 1]$. Since $sd(v_i) \leq sd_1$ and $sd(v_i) \leq sd_2$, it follows that $sd(v_i) < sd(v_j)$.
Case (b): v_j is not the indexing node of its cluster. Let v_l be the indexing node of $C(v_j)$, with $l = ind(C(v_j)) < j$. Because D is loosely-formatted, there exists a node v_k such that v_k is an ancestor of v_l in T' and $C(v_k) = C(v_i)$. By the minimality of j, SD conformance holds for (v_k, v_l), implying $sd(v_k) < sd(v_l)$. Since $sd(v_k) = sd(v_i)$ and $sd(v_l) = sd(v_j)$, we again obtain $sd(v_i) < sd(v_j)$.
In both cases, we achieve $sd(v_i) < sd(v_j)$, a contradiction to our initial assumption that $sd(v_i) \geq sd(v_j)$. Therefore, the assumption that T is not robust must be false, and T is robust. □

Hierarchy of Robustness. We can now summarize the relationships among all document classes introduced using the Venn diagram in Figure 2. Well-formatted documents form the smallest class (yellow), and are strictly contained within loosely-formatted documents (Definition 5, orange) via Theorem 5. For any approach $\texttt{Infer-SD}_K$ from $\mathcal{K}$, loose-formattedness is a sufficient condition for SHT robustness (Theorem 6), while SD conformance is both necessary and sufficient (Theorem 3). Consequently, for each $\texttt{Infer-SD}_K$, loosely-formatted documents are strictly contained within the class of documents that satisfy SD conformance (one such conformance class is highlighted in green). We also highlight the corresponding conformance classes for local-first (blue) and global-first (purple) approaches, which are two instances of $\mathcal{K}$. Loosely-formatted documents lie in the intersection of these conformance classes. Depth-aligned documents (pink) guarantees robustness under local-first approach, so they lie within the local-first conformance class, with possible intersections with other classes.

Frequency of Each Class. Across 592 real-world documents from our evaluation spanning diverse domains (see Section 6.1) with manually annotated true SHTs, we find 60.17% are loosely-formatted, which is 2.57× the fraction of well-formatted documents (23.39%).

To further analyze class frequencies at scale, we additionally collect 3,919 real-world PDF documents [56], each with tables of contents encoded in metadata (i.e., extractable via `get_doc` in `PyMuPDF` [57]), avoiding manual true SHT construction. Among them, 52.0% are well formatted, 82.6% loosely formatted, and 32.53% depth-aligned; 85.74% and 83.39% satisfy SD conformance under the local- and global-first approaches, respectively. Since local-first dominates, we focus on it in our evaluation in Section 6.

4.5 Relationship with ZenDB

Lin et al. [41] propose an algorithm for SHT construction (Algorithm 1 in [41]). In Section 4.5.1, we prove that this algorithm is equivalent to our two-stage workflow with global-first SD inference: for each node v_i, both approaches select as their parent the node v_j with the largest index $j < i$ such that $ind(C(v_j)) < ind(C(v_i))$. Moreover, in Section 4.4, we stated that one requirement of well-formattedness as consistency between each parent-child pair and

the first occurrences of their corresponding visual patterns. We show in Section 4.5.2 that this requirement is equivalent to Lin et al.'s original well-formattedness definition (Definition 1 in [41]), which requires consistency among the ancestor visual-pattern lists of all nodes within the same cluster. Lin et al. [41] does not go beyond well-formattedness and does not consider properties such as robustness or compactness.

4.5.1 SHT Construction. ZenDB proposes Algorithm 1 in [41] to construct SHTs. We show that this algorithm is equivalent to our two-stage workflow, with global-first SD inference (Algorithm 2) followed by SHT assembly (Algorithm 3).

Theorem 7. Our two-stage workflow with global-first SD inference is equivalent to Algorithm 1 of ZenDB.

Proof. For simplicity, we use $ind(v_i)$ to denote $ind(C(v_i))$.

1) Removing the constraint on text span in Line 8 of Algorithm 1 of ZenDB. In Algorithm 1 of ZenDB, when inserting a node v_i (Line 6), the algorithm chooses the node v_j with the largest index j such that $ind(v_j) < ind(v_i)$ as the parent of v_i, where $ind(v_j) < ind(v_i)$ is implied by Line 7, and the maximality of j is implied by Line 8. At this point in Algorithm 1, $ts(v_j) = [p_j, \ldots, p_{s-1}]$, where p_s corresponds to the next non-descendant v_s of v_j in T. v_s is thus inserted before v_i, implying $ind(v_s) < ind(v_i)$. The maximality of j further implies $s > i$. Therefore, $p_i \in ts(v_j)$ holds automatically, and the constraint $ind(v_i) \in ts(v_j)$ in Line 8 of Algorithm 1 can be removed without changing the algorithm's behavior.

After removing this constraint on text span, the nested loops (Line 4-10) in Algorithm 1 of ZenDB can be reduced to the following single loop, where we additionally replace constraint $v_j \in V - V_C$ (Line 7) with $v_j \in T$:

1: **for** $v_i \in V$ in ascending order of i **do**
2: $\quad v_k \leftarrow \arg\max_{v_j \in T, ind(v_j) < ind(v_i)} : j$
3: $\quad$ Add node v_i to T as the rightmost child of v_k.

2) Equivalence to our two-stage workflow with global-first SD inference. Algorithm 3 (Line 3) selects v_k as the parent of v_i, where k is the largest index j such that $v_j \in V$, $p_i \in ts(v_j)$, and $sd(C(v_j)) < sd(C(v_i))$. Under global-first SD inference, the ordering of semantic depths is consistent with the ordering of cluster indices. Therefore, $sd(C(v_j)) < sd(C(v_i))$ is equivalent to $ind(v_j) < ind(v_i)$. As argued in *1)*, the condition $p_i \in ts(v_j)$ is also redundant for Algorithm 3 under global-first. Line 3 of Algorithm 3 under global-first can thus be reduced to the following line:

1: $v_k \leftarrow \arg\max_{v_j \in T, ind(v_j) < ind(v_i)} : j$

This line is identical to Line 2 of the single loop in *1)*. Therefore, our two-stage workflow under global-first is equivalent to Algorithm 1 of ZenDB. □

4.5.2 Well-Formattedness. ZenDB defines a document D, given its true SHT T' and clusters C, to be well-formatted via Definition 1 in [41] if it satisfies the following two conditions:
1) Any two nodes that are siblings (i.e., have the same parent) in T' must belong to the same cluster.
2) Any two nodes belonging to the same cluster must share identical *visual prefixes*, denoted $vispre$, defined as the sequences of clusters along the paths from the root to the nodes in T'.

In Section 4.4, we rephrase the second condition as a consistency requirement between any parent-child pair and the first occurrence of their corresponding visual patterns. We now show that this rephrasing (i.e., condition (2) in Theorem 8) doesn't change ZenDB's original definition (i.e., condition (1) in Theorem 8):

Theorem 8. Consider a document D with nodes V, clusters C, and true SHT T'. The following two conditions are equivalent:
(1) $\forall\ v_i, v_j \in V$, if $C(v_i) = C(v_j)$, then $vispre(v_i) = vispre(v_j)$, where $vispre(v) = [C(v_1), \ldots, C(v)]$ is the sequence of clusters along the path from the root v_1 to v in T'.
(2) $\forall\ v_i, v_j \in V$, if v_i is the parent of v_j in T', then there exists a parent-child pair (v_k, v_l) in T' such that $l = ind(C(v_j))$ and $C(v_k) = C(v_i)$.

Proof. (1) $\Rightarrow$ (2). If (1) holds for D, $vispre(v_j) = vispre(v_l)$ implies that the parents of v_j and v_l in T' belong to the same cluster, establishing (2).

(2) $\Rightarrow$ (1). If (2) holds for D, then for v_i and v_j in the same cluster C, the true parent of v_i belongs to the same cluster as the true parent of $v_{ind(C)}$, and the same holds for v_j. Thus, v_i, v_j, and $v_{ind(C)}$ all have parents in the same cluster. Applying the same argument recursively to their parents, we conclude that v_i and v_j share identical clusters at every level along their paths to the root. Hence, $vispre(v_i) = vispre(v_j)$. □

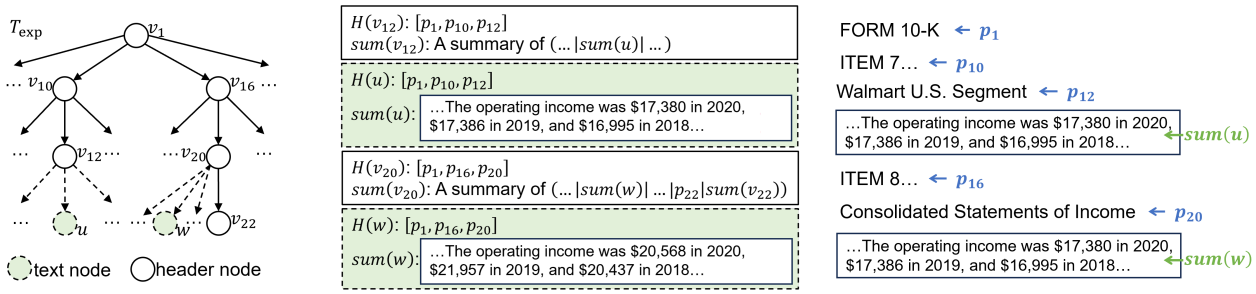
5 APPLICATION: SHT-BASED RAG

To demonstrate the usefulness of the SHTs inferred by our two-stage workflow, we present SHED-RAG, a retrieval-augmented generation (RAG) technique for documents that leverages SHTs and the hierarchical structural information they encode, information largely ignored by existing RAG techniques which only focus on content [13, 15, 18, 26, 27, 36, 86]. This technique applies independent of which SD inference approach is used.

Given a document D with an inferred SHT T, as a pre-processing step, we first expand T to include the raw text of D as part of T, in addition to the original header phrases, for fine-grained retrieval (Section 5.1). We call this expanded tree T_{exp}. Then, given a natural-language question q, we retrieve on T_{exp} the nodes most relevant to q and composes them into a final context for LLM generation, using both semantic (e.g., the text span of a node) and structural (e.g., header phrases of a node and its ancestors) information (Section 5.2). Both types of information are crucial for retrieval and generation. For example, knowing both the header phrase of v_{20} in Figure 1b and the context around *operating income* within $ts(v_{20})$ enables LLMs to generate the correct answer, whereas either piece of information alone is insufficient.

5.1 SHT Expansion

We refer to the nodes in T as *header nodes*. We expand T (which contains only header nodes, i.e., header phrases) into T_{exp} using text chunks from D, as defined below, so that, in addition to the header phrases, the rest of D is also preserved, enabling fine-grained retrieval with T_{exp}. We define a *text chunk* t to be a concatenation (denoted by $|$) of a consecutive set of phrases in D: $t = (p_i | p_{i+1} | \ldots | p_j)$. To ensure fine granularity for downstream retrieval, we set a token limit for each text chunk (e.g., 100 tokens in our implementation).



(a) SHT expansion. **(b) Node content and hierarchical context.** **(c) Context composition.**

Figure 7: SHT-based RAG.

Each text chunk is extracted from D and added to T as a new node, referred to as *text node*, as follows. For each *leaf* header node v in T, we split its text span $ts(v)$ into text chunks and attach the corresponding text nodes as children of v, in left-to-right order following D's reading order. For each *non-leaf* header node v, if there is text between its header and the header of its first child u (u is also a header node), we split this text into text chunks and attach the corresponding text nodes as children of v, positioned to the left of u in the same left-to-right order.

Example 9. Figure 7a shows the expanded SHT T_{exp} from Figure 1c. We highlight two text chunks containing "*operating income*": one in $ts(v_{12})$ and the other in $ts(v_{20})$ (Figure 1b). The chunk in $ts(v_{12})$ is attached to leaf header node v_{12} as a text node u. The other chunk in $ts(v_{20})$ is attached as text node w to non-leaf header node v_{20} and placed to the left of v_{20}'s first child v_{22}. □

For each node in T_{exp}, both semantic information (e.g., its text span) and structural information (e.g., the header phrases of its ancestors) are important for retrieval and generation. Therefore, we maintain a *node content* and a *hierarchical context* for each node to capture such semantic and structural information, respectively.

Node Content. For each node in T_{exp}, its node content, denoted $sum(\cdot)$, is associated with its text chunk if it is a text node, or with its text span if it is a header node. We first assign $sum(u)$ for each text node u, which is a leaf of T_{exp} by construction, its corresponding text chunk (≤ 100 tokens in our implementation). We then propagate $sum(\cdot)$ of header nodes upward from leaf text nodes, inspired by Raptor [60] (which didn't leverage structure): for a header node v with children $[v_i, \ldots, v_j]$ (ordered by node indices), $sum(v)$ is an LLM-generated summary of $\big(p_i|sum(v_i)|\ldots|p_j|sum(v_j)\big)$ subject to the same token limit as a text chunk, where $p_i, \ldots, p_j$ are the header phrases corresponding to the children that are header nodes; children that are text nodes don't have a header phrase.

Such hierarchical summary aggregation preserves semantic information of each header node v (i.e., $ts(v)$) in a cost-efficient manner. Conceptually, if the node contents of v's children are recursively expanded into their full text spans, $sum(v)$ collapses to $ts(v)$, thereby retaining the complete semantic information of $ts(v)$. Meanwhile, since $sum(v)$ is generated from compact node contents rather than full text spans, it requires substantially fewer LLM tokens than directly summarizing $ts(v)$, which may span multiple pages for large D.

Moreover, the summaries of all header nodes in T_{exp} provide high-level document understanding at different hierarchical levels, complementing the fine-grained details of text node contents (i.e., text chunks). Using summaries instead of full text spans also allows header node contents to match the size of text nodes, thereby preventing bias from varying content lengths when embedding different node types (i.e., header and text nodes) [33].

Hierarchical Context. Let $H(v)$ denote the hierarchical context of node v, defined as the list of header phrases corresponding to the nodes in the path from the root to v in T_{exp}, ordered by phrase indices (i.e., top-down). Note that $H(v)$ includes the header phrase of v if v is a header node, but not if v is a text node.

Example 10. Figure 7b shows the node contents and the hierarchical contexts of two header nodes v_{12}, v_{20} and two text nodes u, w. For header node v_{12}, its node content $sum(v_{12})$ is a summary of the node contents of its children (e.g., $sum(u)$), while its hierarchical context is $H(v_{12}) = [p_1, p_{10}, p_{12}]$, consisting of the header phrases of v_{12} and its ancestors v_1 and v_{10}. For text node u, $sum(u)$ is simply its text chunk, and $H(u) = H(v_{12})$ since u does not correspond to any header phrase. □

5.2 Retrieval and Context Composition

Given a natural-language question q, with $sum(\cdot)$ and $H(\cdot)$ of each node in T_{exp} computed offline, we now describe our technique of online retrieval (Section 5.2.1) and context composition (Section 5.2.2).

5.2.1 Retrieval. For each node v in T_{exp}, we compute its embedding using a text embedding model (e.g., `Sentence-BERT` [58]), denoted $E(v)$, which encodes both its node content $sum(v)$ and its hierarchical context $H(v)$, ensuring that downstream retrieval incorporates both type of information. Specifically, $E(v)$ is an embedding of the string, denoted $str(v)$, formed by concatenating the header phrases in $H(v)$, in order of phrase indices, followed by $sum(v)$. For example, as shown in Figure 7b, $E(u)$ is the embedding of $str(u) = (p_1|p_{10}|p_{12}|sum(u))$.

5.2.2 Context Composition. Given question q, we retrieve the top-k nodes from the expanded SHT T_{exp} whose embeddings are most similar to that of q. The retrieved nodes may include both header and text nodes. We then compose the retrieved nodes into the final context for LLM to generate an answer, preserving both semantic and hierarchical information. The context is initially constructed by concatenating $str(v)$ for all retrieved nodes v, following their order in the pre-order traversal of T_{exp}, where $str(v)$ denotes the concatenation of v's node content and hierarchical context as described above. We further refine the context by removing duplicate header

Table 1: Dataset statistics.

Datasets	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*
Avg. #Pages	6.79	4.04	9.79	80.09
Avg. #Tokens	2829	2050	8859	116505
#Documents	19	73	416	84
#Questions	418	669	1451	150

phrases that arise from shared ancestors among the retrieved nodes: for any two retrieved nodes v_i, v_j where v_i appears before v_j in the pre-order traversal, we remove all header phrases in $H(v_i) \cap H(v_j)$ from $str(v_j)$. This refined context is then used as input for LLM generation.

Example 11. Given the question in Figure 1b and the retrieved nodes $\{u, w\}$, Figure 7c shows the final context for LLM generation, which is $(p_1|p_{10}|p_{12}|sum(u)|p_{16}|p_{20}|sum(u))$, constructed by concatenating $str(u)$ and $str(w)$ in order, removing duplicate headers p_1 from $str(w)$ (i.e., $H(u) \cap H(w) = \{p_1\}$). □

6 EVALUATION

We describe our datasets (Section 6.1) and the implementations of SHED, our SHT inference workflow, and SHED-RAG, our SHT-based RAG technique. We evaluate SHED, focusing on robustness and compactness of the inferred SHTs (Section 6.3), and SHED-RAG in terms of question answering (QA) accuracy (Section 6.4).

6.1 Datasets

We collected four datasets of real-world PDF documents spanning diverse lengths, domains, and templates: civic agenda reports, legal contracts, scientific papers, and financial filings. We *manually* constructed the true SHTs for all 592 documents. Each dataset is paired with natural-language questions and ground-truth answers, with each question querying a single document. Dataset statistics are shown in Table 1. We use these documents to evaluate SHED and the associated QA sets to evaluate SHED-RAG. Below, we describe each dataset and its QA accuracy metric.

Civic Agenda Reports (*Civic*). This dataset, from [41], contains 19 Malibu City Council agenda reports [7], each uses a complex, nested hierarchy to describe civic projects and their attributes (e.g., status, updates, start and completion times). We curate 418 questions that ask either for specific attributes of a civic project or for sets of projects satisfying given predicates (e.g., projects starting in summer 2021). For QA accuracy, we evaluate the former class of questions using 0-1 accuracy, and the latter using F-1 score against ground-truth project sets.

Legal Contracts (*ContractNLI*). This dataset, from [32], consists of 669 questions on 73 non-disclosure agreements (NDAs) with hierarchical structures. Each question asks whether a legal hypothesis is *Entailed* or *Contradicted* by the contract. We remove questions whose ground-truth answers are *NotMentioned*. QA accuracy is measured using 0-1 accuracy.

Scientific Papers (*Qasper*). This dataset, from [12], contains questions about both detailed and high-level content in natural language processing (NLP) research papers. Here, we use its test split, which includes 416 papers and 1451 questions. QA accuracy is evaluated using LLM-as-a-judge [84], following established practice [61, 83]. We use `gpt-4o-mini` to compare the predicted answer with the ground truth and assign a similarity score from 0 to 3 (see Appendix A.2).

Financial Filings (*FinanceBench*). This dataset, from [25], includes 84 financial filings and 150 questions. The documents span 10Ks, 10Qs, 8Ks, and earnings reports from U.S. publicly traded companies between 2015 and 2023. Questions require financial analysis (e.g., whether FY2020 dividend is paid) and numerical reasoning over domain-specific metrics (e.g., FY2019 unadjusted operating income). We use LLM-as-a-judge with a rating scale of 0-3 by `gpt-4o-mini` for QA accuracy [61, 75, 78] (Appendix A.2).

6.2 Implementation of SHED and SHED-RAG

SHED. For header identification, we use small pretrained models that can be run on CPUs from `HURIDOCS` [24]. Following `HURIDOCS`' recommendations, we use a smaller model [30] for large-scale datasets (*Qasper* and *FinanceBench*), trading slightly lower accuracy for substantially faster execution than the larger model [11, 24]. For each header phrase, we extract its visual pattern using `PyMuPDF` [57]. The header phrases are treated as nodes V and clustered by their visual patterns, yielding clusters C. Finally, we use local-first SD inference, and construct SHTs T using Algorithm 3. We adopt local-over global-first since more documents satisfy SD conformance under local-first, as shown in prior statistics (Section 4.4). We implemented SHED using ~1.3k lines of Python.

SHED-RAG. SHED-RAG is implemented using ~600 lines of Python. In an expanded tree T_{exp}, we set a 100-token limit for $sum(\cdot)$ of each node, use `gpt-4o-mini`[4] to generate $sum(\cdot)$ for each header node, and use `Sentence-BERT` [58] to embed all nodes and queries.

6.3 Robustness and Compactness

We evaluate SHED and five baselines (Section 6.3.1) on SHT robustness and compactness using two metrics (Section 6.3.2).

6.3.1 SHT Inference Baselines. We have five baselines:

Deep. This approach uses the same nodes V and clusters C as SHED, and constructs an SHT by linking the header nodes into a list in index order (e.g., Figure 1d-1), guaranteeing robustness at the cost of compactness.

Wide. Using the same nodes V and clusters C, this approach selects the first node as the root and the remaining as its children (e.g., Figure 1d-2), with low robustness and high compactness.

GROBID. GROBID [17] is an off-the-shelf model trained on technical and scientific publications to infer SHTs end-to-end.

LLM-text. This approach extracts plain text from D using `PyMuPDF` and prompts an LLM (`gpt-4o-mini`) to infer an SHT from the text (see prompt in Appendix A.1). If the text exceeds the LLM input limit, it is evenly split into as few chunks as needed; SHTs inferred per chunk are merged under a virtual root.

LLM-vision. This approach prompts an LLM (`gpt-4o-mini`) with page images from D to infer SHTs. Pages are grouped sequentially into maximal-size batches within the LLM input limit (see prompt in Appendix A.1); SHTs inferred per batch are merged under a virtual root.

[4]The model version is `2024-07-18`, and the API version is `2024-12-01-preview`, with a cost of $0.15/$0.60 per 1M input/output tokens, for all `gpt-4o-mini` instances used in this paper.

Table 2: Header Identification. V and V' are the sets of extracted headers and true headers, respectively.

	Recall = $\lvert V \cap V' \rvert / \lvert V' \rvert$				Precision = $\lvert V \cap V' \rvert / \lvert V \rvert$				F-1 = $2\lvert V \cap V' \rvert / (\lvert V \rvert + \lvert V' \rvert)$			
Dataset	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*
GROBID	10.52%	38.33%	72.1%	47.24%	60.42%	21.98%	71.77%	51.51%	17.51%	23.22%	71.59%	46.8%
LLM-text	99.85%	81.44%	86.82%	26.97%	34.95%	20.18%	65.30%	35.49%	51.75%	26.59%	72.93%	24.95%
LLM-vision	98.99%	83.78%	83.71%	78.67%	32.64%	21.41%	44.56%	17.78%	48.92%	29.47%	56.03%	27.78%
SHED	97.62%	84.72%	88.91%	93.43%	91.86%	71.82%	87.19%	62.98%	94.6%	75.39%	87.91%	72.39%

Note that although we assume, for the purposes of theoretical analysis of correctness, robustness, and compactness, that the node set V used by SHED is identical to that of the true SHT V', this assumption need not hold due to errors in header identification. Table 2 reports the recall, precision, and F-1 scores of the header set V extracted per document by GROBID, LLM-text, LLM-vision, and `HURIDOCS` (used by SHED, Deep, and Wide), relative to V', averaged over all documents in each dataset, with the highest and second highest values highlighted in gray. `HURIDOCS` consistently achieves high accuracy, whereas GROBID performs poorly on datasets outside its training domain (i.e., *Civic*, *ContractNLI*, and *FinanceBench*). Both LLM-based approaches show low precision, with many extracted headers not being true headers, partly due to hallucination: averaged over all 592 documents, 6.72% and 17.16% of the headers extracted per document by LLM-text and LLM-vision, respectively, do not appear in the original document. We remove these hallucinated headers in subsequent evaluations. Thus, the performance of LLM-text and LLM-vision is an upper bound on actual performance.

6.3.2 Two Metrics. We design two metrics to evaluate robustness and compactness: *top-down*, denoted $M_\downarrow$, and *bottom-up*, denoted $M_\uparrow$. Given an inferred SHT T (with header nodes only) for a document with true SHT T', $M_\downarrow$ compares, for each node v in T', the text spans $ts(v)$ and $ts'(v)$ inferred from T and T', respectively, motivated by the definition of robustness (Definition 2), which requires $ts'(v) \subseteq ts(v)$. In contrast, $M_\uparrow$ operates in reverse: it compares hierarchical contexts $H(v)$ and $H'(v)$ inferred from T and T' (i.e., the list of ancestor headers defined in Section 5), respectively, motivated by Theorem 1, which requires $H'(v) \subseteq H(v)$.

Let the node sets of T and T' be V and V', respectively. As mentioned earlier, V may differ from V'. We now define the top-down and bottom-up metrics for T.

Top-down ($M_\downarrow$). Consider a node $v \in V'$, with true text span $ts'(v)$ inferred from T'. $M_\downarrow$ compares $ts'(v)$ with the text span $ts(v)$ inferred from T. For now we assume $v \in V$; we consider the other case later.

To measure robustness of T for v, we compute the recall of the text span (i.e., the set of words) $ts(v)$ with respect to $ts'(v)$, namely $\text{REC}(v) = \frac{|ts(v) \cap ts'(v)|}{|ts'(v)|}$, where a higher value indicates better robustness. A value of 1 implies that $ts(v)$ is a superset of $ts'(v)$, consistent with Definition 2.

To measure compactness, we compute the precision of the two text spans, $\text{PREC}(v) = \frac{|ts(v) \cap ts'(v)|}{|ts(v)|}$, where a higher value indicates a more compact text span $ts(v)$.

To jointly evaluate robustness and compactness for node v, we use the F-1 score $\text{F-1}(v) = \frac{2}{\text{REC}(v)^{-1} + \text{PREC}(v)^{-1}}$.

The overall top-down robustness and compactness of T are computed by averaging over all true nodes in V': $M_\downarrow(T) = \text{avg}_{v \in V'} f(v)$, where $f \in \{\text{REC}, \text{PREC}, \text{F-1}\}$. We average over V' to ensure that SHTs inferred from the same document by different methods, potentially with different node sets, are evaluated on a common basis.

If $v \notin V$, we select the node $u \in V$ that maximizes $f(v)$ after substituting $ts(v)$ with $ts(u)$ in the computation of $f(v)$. For example, when $f = \text{REC}$, we replace v with u whose text span $ts(u)$ covers the largest portion of $ts'(v)$ to compute $f(v)$.

Bottom-up ($M_\uparrow$). The bottom-up robustness and compactness of T are computed analogously to the top-down metrics, except that the text span of each node $v \in V'$ is replaced by its hierarchical context. Specifically, for each node $v \in V'$, we compare its hierarchical context $H(v)$ inferred from T with its true hierarchical context $H'(v)$ inferred from T'. For example, bottom-up recall is defined as $\text{REC}(v) = \frac{|H(v) \cap H'(v)|}{|H'(v)|}$, where $H(v)$ denotes the set of words appearing in all header phrases of the ancestor nodes of v as inferred from T. If $v \notin V$, $H(v)$ is replaced with $H(u)$ for some $u \in V$ that maximizes $f(v)$; in this case, we extend $H(u)$ by including the header phrase corresponding to v. Robustness and compactness are then computed using recall, precision, and F-1 score in the same manner as before.

6.3.3 Experimental Results. We show the evaluation results of $M_\downarrow$ and $M_\uparrow$, as well as LLM cost for SHT inference.

Robustness & Compactness. Table 3 and Table 4 report the evaluation results of $M_\downarrow$ and $M_\uparrow$, respectively, with $f \in \{\text{REC}, \text{PREC}, \text{F-1}\}$, averaged over all documents in each dataset. We highlight the highest and second highest values of recall, precision, and F-1 score. ***SHED achieves the highest F-1 scores across almost all datasets in both $M_\downarrow$ and $M_\uparrow$, demonstrating its advantage in simultaneously ensuring robustness and optimizing compactness.*** Across almost all datasets in both $M_\downarrow$ and $M_\uparrow$, Deep (Wide) has the highest values of recall (precision, respectively). Both approaches are closely followed by SHED with minor reductions in recall or precision, whereas the remaining baselines fail to achieve comparable performance. However, both Deep and Wide exhibit low F-1 scores for both $M_\downarrow$ and $M_\uparrow$: Deep generates deep trees, resulting in poor compactness (and therefore precision), while Wide ignores robustness (and thus has poor recall). GROBID has higher top-down and bottom-up values of recall, precision, and F-1 scores on *Qasper* than on the other datasets, as *Qasper* closely aligns with its training domain (i.e., scientific papers), showing its limited generalizability. LLM-vision outperforms LLM-text in terms of F-1 scores across almost all datasets in both $M_\downarrow$ and $M_\uparrow$, likely due to the additional visual information beyond textual semantics that helps LLMs infer structures more accurately.

LLM Cost. For SHT inference on all 592 documents, only LLM-text and LLM-vision incur LLM costs, in total $2.1 and $73.3, respectively, using `gpt-4o-mini`; SHED and other baselines incur no LLM cost.

Table 3: Top-down robustness and compactness evaluation ($M_{\downarrow}$).

f	REC				PREC				F-1			
Datasets	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*
Deep	100%	99.83%	98.54%	100%	19.89%	24.41%	19.64%	10.49%	26.72%	31.00%	25.52%	12.92%
Wide	22.19%	91.56%	91.36%	72.42%	99.95%	79.53%	98.15%	96.68%	30.28%	73.55%	91.46%	72.15%
GROBID	34.39%	81.49%	81.96%	62.09%	13.65%	37.93%	78.53%	61.22%	14.42%	40.42%	75.53%	51.36%
LLM-text	90.75%	74.10%	56.58%	80.24%	94.55%	64.35%	38.31%	33.35%	91.58%	60.63%	36.45%	27.43%
LLM-vision	89.82%	83.55%	63.17%	87.90%	95.63%	65.83%	40.89%	77.72%	90.16%	62.84%	38.87%	68.63%
SHED	98.13%	97.34%	97.87%	90.04%	97.84%	78.53%	97.32%	93.91%	96.86%	78.09%	96.81%	85.42%

Table 4: Bottom-up robustness and compactness evaluation ($M_{\uparrow}$).

f	REC				PREC				F-1			
Datasets	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*
Deep	95.64%	76.96%	83.73%	95.54%	38.13%	41.07%	39.76%	17.74%	41.76%	39.86%	39.49%	19.26%
Wide	49.59%	59.12%	72.43%	32.45%	75.94%	78.88%	91.64%	54.75%	55.36%	63.99%	77.49%	37.23%
GROBID	4.65%	28.80%	70.31%	25.97%	14.70%	48.31%	90.47%	48.55%	6.90%	27.89%	75.81%	27.09%
LLM-text	60.63%	50.86%	39.95%	41.98%	76.27%	56.29%	57.74%	52.00%	63.49%	45.76%	44.35%	39.34%
LLM-vision	72.31%	55.28%	40.08%	55.27%	81.25%	59.91%	54.69%	61.90%	73.79%	49.28%	41.60%	43.68%
SHED	92.95%	67.84%	75.94%	59.05%	82.30%	78.49%	91.36%	63.07%	85.83%	68.14%	77.29%	54.70%

6.4 RAG-based Question Answering

We demonstrate practical usefulness of our SHT inference by evaluating QA accuracy of SHED-RAG and four RAG baselines (Section 6.4.1). We further conduct ablation studies to analyze impact of different components of SHED-RAG on QA accuracy (Section 6.4.2).

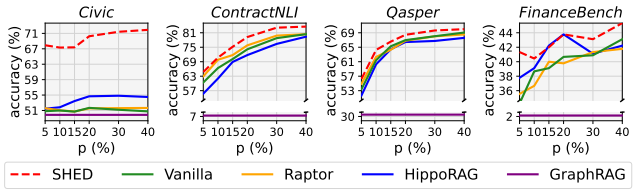


Figure 8: QA accuracy (y-axis) of SHED-RAG and baselines on four datasets across different context sizes (x-axis).

6.4.1 End-to-End Comparison. We have four baselines:

Vanilla. This text-based technique [36] divides a document into fixed-size text chunks and retrieves the chunks whose embeddings are most similar to that of the question.

Raptor. This technique [60] builds a tree of fixed depth (set to 5 based on the paper). Its leaves correspond to fixed-size text chunks, which are recursively clustered by semantic similarity and summarized into internal nodes until the depth limit is reached. All tree nodes are embedded for retrieval.

HippoRAG. This technique [19, 27] extracts entities and relationships from a document and integrates them into a knowledge graph (KG) for downstream retrieval.

GraphRAG. We use GraphRAG's global search mode [15], which builds a knowledge graph, clusters nodes into communities, and summarizes each community using its constituent nodes (i.e., entities) and induced edges (i.e., relationships). For a given question, each community summary is used to produce an intermediate answer, and all intermediate answers are then aggregated as context for LLMs to produce the final answer.

For all baselines, we limit the chunk size to 100 tokens and use `Sentence-BERT` [58] for embedding. We fix the LLM to `gpt-4o-mini` for summarization (Raptor and GraphRAG), KG construction (HippoRAG and GraphRAG), and answer generation. We limit the generation context to be at most p% of queried document's size ($p \in \{5, 10, 15, 20, 30, 40\}$) and truncate it if it exceeds `gpt-4o-mini`'s input limit of 128,000 tokens. This context size limit does not apply to GraphRAG's global search mode due to two-stage generation: intermediate answers from community summaries, followed by final aggregation. Moreover, since some questions in *FinanceBench* depend on tables within documents, to avoid noise in QA accuracy arising from LLMs' difficulty in interpreting tables serialized as plain text, we reformat each originally serialized table in generation context in Markdown [1, 66]. Specifically, each table row is represented as a Markdown section marked with ##, and each cell value `v`, together with its corresponding column header `k`, is represented as a list item "`- k: v`" under this section. This table reformatting is feasible for Vanilla, Raptor, and SHED, because their retrieved contexts can be mapped back to the raw text chunks in the original documents, whereas such mapping is impractical for the graph-based approaches HippoRAG and GraphRAG.

Experimental Results. Figure 8 shows QA accuracy (y-axis) across different context size limits p (x-axis). ***SHED-RAG achieves the highest QA accuracy across almost all datasets and p***, outperforming Vanilla by up to 21.15%, Raptor by 20.26%, HippoRAG by 17.42%, and GraphRAG by 75.94%. GraphRAG's global search mode does not support context size limits, so its accuracy appears as a horizontal line invariant to p, which is consistently the lowest among all baselines, as it relies solely on summaries and therefore misses details. Vanilla and Raptor achieve comparably low accuracy, suggesting that summaries based purely on semantic similarity, as used by Raptor, are insufficient for accurate QA. HippoRAG performs poorly on *ContractNLI* and *Qasper* and achieves slightly better accuracy on *Civic* and *FinanceBench*, indicating that while KGs capture useful relationships, they remain insufficient for accurate QA without document hierarchical structures.

The accuracy gap between SHED-RAG and the best-performing baseline is largest on *Civic* (13.95% - 22.06%), while being smaller on the other three datasets, probably due to differences in the *structural demands* of the questions: hierarchical structure is necessary to correctly answer many questions in *Civic*, while being less critical for

Table 5: Impact of SHTs on QA accuracy (relative).

Datasets	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Avg.*
Deep	-34.56%	-6.97%	-11.02%	-44.49%	-24.26%
Wide	-21.23%	-0.38%	-0.50%	-4.5%	-6.65%
GROBID	-35.4%	-6.21%	-0.83%	-8.48%	-12.73%
LLM-text	-3.41%	-5.83%	-0.13%	-14.99%	-6.09%
LLM-vision	-2.11%	-0.94%	-3.03%	-6.48%	-3.14%
SHED	1.52%	-0.19%	0.18%	-1.49%	0.01%

questions in other datasets. We formalize a *structurally demanding question* q as follows. Let P' denote the minimal set of phrases needed for LLMs to correctly answer q. If P' contains two non-sequential phrases p_i and p_j (i.e., $|j-i| \neq 1$) that correspond to an ancestor-descendant pair in the true SHT, we label q as *structurally demanding*. When p_j is a non-header phrase, we further require p_i to be a *non-parent ancestor* of p_j, thereby excluding questions that rely only on trivial hierarchical structures, which can be captured even by a wide SHT: retrieving p_j on a wide SHT automatically retrieves its parent header p_i as its hierarchical context. *Civic* contains approximately 60% structurally demanding questions, compared to 12% in *ContractNLI*, 0.4% in *Qasper*, and none in *FinanceBench*, which explains the varying accuracy gaps across these datasets. Future work should explore the construction of benchmarks that are more representative (i.e., benefit from structure and content).

LLM Cost. For LLM cost (excluding generation) using `gpt-4o-mini`, ***SHED-RAG spends $6.9 in total over 592 documents comprising 14M tokens***, only 0.53× and 0.07× of HippoRAG and GraphRAG, respectively, while being 1.61× of Raptor, thanks to Raptor's depth limitation (i.e., 5) during tree construction. Both SHED-RAG and Raptor use LLMs to generate a summary for each tree node from the summaries of its children, with each summary capped at 100 tokens. As a result, their costs is related to the number of nodes in their respective trees. For a document D, the number of leaves (e.g., text nodes for SHED-RAG) is the same in both trees, as they correspond to the text chunks of D. However, the number of internal nodes (e.g., header nodes for SHED-RAG) differs: in SHED-RAG, it depends on the header phrases in D (longer documents generally contain more headers), whereas in Raptor, it remains relatively constant due to the fixed tree depth. Across the 592 documents, SHED-RAG's trees contain 1.27× as many nodes as Raptor's, leading to a retrieval cost that is 1.61× higher. Breaking its cost down by dataset, SHED-RAG spends $0.16 on *Civic* (54K tokens), $0.15 on *ContractNLI* (150K tokens), $2.28 on *Qasper* (4M tokens), and $4.33 on *FinanceBench* (10M tokens), demonstrating its low cost and good scalability.

6.4.2 Ablation Studies. We study the impact of different components of SHED-RAG on QA accuracy, fixing $p\% = 20\%$.

Impact of SHTs. In our implementation of SHED-RAG, SHTs are inferred by SHED. Here, we replace these SHTs with alternatives from the approaches evaluated in Section 6.3 (Deep, Wide, GROBID, LLM-text, and LLM-vision), and report QA accuracy relative to using the true SHTs in Table 5. Specifically, Table 5 reports the accuracy change Δ of SHED-RAG when using T inferred by any approach (acc_T), relative to the true SHT T' ($acc_{T'}$): $\Delta = \frac{acc_T - acc_{T'}}{acc_{T'}}$. The results reveal a correlation between QA accuracy and SHT robustness and compactness: ***SHTs inferred by SHED achieve the highest QA accuracy and perform on par with the true SHTs***, consistent with their high top-down and bottom-up F-1 scores (Tables 3 and 4). Deep has the greatest accuracy reduction due to poor compactness (i.e., low top-down and bottom-up precision), which introduces redundant text spans and hierarchical context that contaminate node embeddings and retrieved contexts. GROBID exhibits significant accuracy drops on datasets outside its training domain (i.e., *Civic*, *ContractNLI*, and *FinanceBench*). Wide, with poor robustness (i.e., low top-down and bottom-up recall), has large accuracy reduction on *Civic* which contains many structurally demanding questions (Section 6.4.1), while its impact on the other three datasets with fewer structurally demanding questions is smaller. We further evaluate Wide on the 85 structurally demanding questions in *ContractNLI*, and observe rapid drop in relative accuracy (from 0.38% to 14.08%). The two LLM-based SHT inference approaches experience larger accuracy reductions on *FinanceBench* than on other datasets, likely due to hallucinations in large documents, consistent with their low top-down and bottom-up F-1 scores on *FinanceBench*.

Table 6: Impact of SHT expansion on QA accuracy (relative). `w/o SHT expansion` means excluding text nodes from retrieval *and* context composition.

Datasets	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Avg.*
`w/o SHT expansion`	-3.2%	-5.28%	-9.98%	-21.33%	-9.95%

Table 7: Impact of hierarchical context in QA accuracy (relative). `embedding`, `context`, and `both` means excluding hierarchical context from node embeddings, final context, and both, respectively.

Datasets	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Avg.*
`embedding`	-1.69%	0.0%	0.19%	-7.63%	-2.28%
`context`	-16.92%	-2.64%	-1.83%	-6.6%	-7.0%
`both`	-22.2%	-3.02%	-0.67%	-8.63%	-8.63 %

Impact of SHT Expansion. Table 6 reports the relative QA accuracy change $\Delta = \frac{acc' - acc}{acc} \times 100\%$, where acc' denotes the QA accuracy of SHED-RAG without SHT expansion (i.e., excluding text nodes in T_{exp} from retrieval and context composition), and acc corresponds to the original SHED-RAG. ***Removing SHT expansion reduces QA accuracy drops across all datasets***, as text nodes store the original text chunks that provide concrete and reliable evidence in the retrieved contexts for QA.

Impact of Hierarchical Context. We ablate hierarchical context by removing them from node embeddings (i.e., embedding only $sum(v)$ for each node v), from final context (i.e., concatenating $sum(v)$ of the retrieved nodes in decreasing order of embedding similarity to the question until reaching the context token limit), or from both components. Table 7 reports the resulting QA accuracy changes $\Delta = \frac{acc' - acc}{acc} \times 100\%$. ***Removing hierarchical context from either node embeddings or final retrieved contexts consistently reduces QA accuracy.*** On average, accuracy drops more sharply when hierarchical contexts are removed from the retrieved contexts than when they are removed from node embeddings, indicating that providing LLMs with hierarchical contexts during QA is more impactful than incorporating hierarchical contexts solely into node embeddings. *Civic* is most severely affected by the removal of hierarchical contexts from the retrieved contexts, consistent with its high proportion of structurally demanding questions. *ContractNLI* and *Qasper* exhibit minimal sensitivity to removing hierarchical

contexts from node embeddings, likely due to their low proportions of structurally demanding questions; in these cases, adding hierarchical contexts to embeddings may even introduce noise that degrades retrieval accuracy. Although *FinanceBench* contains no structurally demanding questions, it still experiences non-negligible accuracy drops, suggesting that hierarchical structure aids both accurate retrieval and generation in long-document settings.

Impact of Embedding Models. We replicate end-to-end comparison in Section 6.4.1 using three additional embedding models: `BM25` [59], `DPR` [29], and `text-embedding-3-small` [50], covering both sparse and dense (`Sentence-BERT`, `DPR` and `text-embedding-3-small`) schemes. GraphRAG is excluded from this experiment because its global search mode does not use embeddings. Table 8 reports QA accuracy under each embedding model, in which ***SHED-RAG achieves the highest QA accuracy across almost all embedding models and datasets***, demonstrating its consistent advantage for QA in RAG invariant to the embedding model.

7 RELATED WORK

We survey related work on processing and analyzing documents.

Systems for Document Analytics. A number of recent systems target the processing of documents across various domains to support analytical tasks. Some systems focus on table extraction, for the specific setting where documents are predominantly tabular or structured [3, 40, 76], which could complement our work. Other systems provide declarative interfaces for general-purpose document analytics [28, 42, 51, 62, 67, 68, 71] by treating documents as linear sequences of text, ignoring their inherent structures; our approach can be integrated to enhance their analytical capabilities. ZenDB [41] is the only document analytics system that considers structures, focusing on the setting where many documents in a collection share the same template. However, ZenDB assumes rigid, templatized document structures, targeting well-formatted documents and the extraction of true structures. In contrast, our approach substantially broadens the classes of applicable documents by introducing looser structures that are robust and compact.

Document Structure Inference. A long line of research has focused on inferring document structures. Some approaches target documents with explicit structures, such as HTML [47], which are not applicable to PDF and Word documents. For these documents, existing approaches can be broadly categorized into three classes. *Rule-based approaches* rely on heuristics [16, 52, 53, 57, 64], which often fail to cover heterogeneous layouts and provide no guarantees—and are dominated by LLM-based approaches. Our approach instead provides theoretical guarantees for robustness, which are sufficiently effective for real-world applications. *Learning-based approaches* [5, 31, 46, 73] train models to infer structures but generalize poorly. We choose GROBID [17], trained primarily on scientific papers, as a representative in our experiments, as it is the most widely adopted. Our approach differs by leveraging learning-based approaches only to header identification, a simpler and more generalizable task [37, 55, 85], where models achieve high accuracy [11, 24, 30]. *LLM-based approaches* can infer structures from plain text [39, 74] or page images [10, 45]. As we see in our experiments, these approaches are costly and prone to hallucination [6, 22, 44], and provide poor robustness and compactness.

Retrieval-Augmented Generation (RAG). RAG retrieves context most relevant to a given query for generation, efficiently compensating for LLM limits on large documents. Most techniques simply treat a document as a collection of chunks [13, 26, 36, 70, 79], ignoring hierarchical relationships among chunks that are crucial for accuracy, which our RAG technique addresses. Raptor [60] organizes chunks into a tree via recursive summarization, while GraphRAG [15] and HippoRAG [19, 27] build a knowledge graph. However, these techniques do not capture hierarchical structure such as section-subsection relationships, thus suffer from similar limitations. Our technique improves accuracy by up to 75.94% compares to these schemes. BookRAG [74], an extension of graph-based techniques, includes hierarchical structure into a knowledge graph, but incurs high construction costs, a common limitation of graph-based techniques. Our technique instead leverages hierarchical structure in a lightweight manner, reducing cost by up to 93% compared to graph-based techniques. Our technique can be used to provide the structure for BookRAG. Our technique could also be used to provide a structure for StructRAG's underlying representation [39] or for PageIndex's hierarchical navigation [81]. Agentic RAG [65] equips agents with tools for planning and reasoning for RAG; our SHTs could be used by agents to reason about structure.

8 CONCLUSION

We introduced SHED, a two-stage workflow for SHT inference that guarantees robustness while minimizing compactness. We proposed local- or global-first SD inference for its first stage, and generalized them into an infinite family of approaches. We further theoretically characterized the space of documents into hierarchical family of classes for which SHED can generate robust SHTs under different approaches. We developed SHED-RAG as an application of SHED for RAG, achieving up to 75.94% QA accuracy improvement over four RAG baselines across four datasets. While inferring the true SHT is difficult, our work shows that inferring robust and compact variants is both feasible and practically useful.

REFERENCES

[1] Improving Agents. 2025. Which Table Format Do LLMs Understand Best? https://www.improvingagents.com/blog/best-input-data-format-for-llms/

[2] Ashkan Alinejad, Krtin Kumar, and Ali Vahdat. 2024. Evaluating the retrieval component in llm-based question answering systems. *arXiv preprint arXiv:2406.06458* (2024).

[3] 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. *arXiv preprint arXiv:2304.09433* (2023).

[4] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*. 3119–3137.

[5] Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. 2023. Nougat: Neural optical understanding for academic documents. *arXiv preprint arXiv:2308.13418* (2023).

[6] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: early experiments with GPT-4 (2023). *arXiv preprint arXiv:2303.12712* 1 (2023).

[7] Malibu Agenda Center. 2024. https://www.malibucity.org/AgendaCenter

[8] Tian-Yi Che, Xian-Ling Mao, Tian Lan, and Heyan Huang. 2024. A hierarchical context augmentation method to improve retrieval-augmented llms on scientific papers. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 243–254.

Table 8: Impact of embedding models.

Models	BM25					DPR					text-embedding-3-small				
Datasets	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Avg.*	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Avg.*	*Civic*	*ContractNLI*	*Qasper*	*FinanceBench*	*Avg.*
Vanilla	51.73%	77.73%	61.04%	29.56%	55.02%	51.96%	70.7%	59.13%	31.11%	53.22%	53.85%	79.97%	67.03%	36.44%	59.32%
Raptor	53.59%	79.82%	61.73%	34.44%	57.39%	52.58%	72.2%	60.79%	32.44%	54.5%	51.92%	81.17%	66.94%	40.44%	60.12%
HippoRAG	53.11%	76.98%	61.15%	31.33%	55.64%	54.81%	71.15%	60.37%	33.78%	55.03%	54.26%	76.53%	66.67%	40.44%	59.47%
SHED-RAG	67.28%	82.21%	63.98%	40.89%	63.59%	63.55%	70.55%	62.35%	34.67%	57.78%	68.21%	84.31%	69.24%	39.78%	65.38%

[9] Jenny SL Chung. 2000. Signals and reading comprehension—Theory and practice. *System* 28, 2 (2000), 247–259.

[10] ContextGem. 2025. https://github.com/shcherbak-ai/contextgem

[11] Cheng Da, Chuwei Luo, Qi Zheng, and Cong Yao. 2023. Vision grid transformer for document layout analysis. In *Proceedings of the IEEE/CVF international conference on computer vision*. 19462–19472.

[12] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011* (2021).

[13] Kuicai Dong, Derrick Goh Xin Deik, Yi Quan Lee, Hao Zhang, Xiangyang Li, Cong Zhang, and Yong Liu. 2024. MC-indexing: Effective long document retrieval via multi-view content-aware indexing. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 2673–2691.

[14] Tom Ebbers, Rudolf B Kool, Ludi E Smeele, Richard Dirven, Chrisje A den Besten, Luc HE Karssemakers, Tim Verhoeven, Jasmijn M Herruer, Guido B van den Broek, and Robert P Takes. 2022. The impact of structured and standardized documentation on documentation quality; a multicenter, retrospective study. *Journal of Medical Systems* 46, 7 (2022), 46.

[15] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[16] Mathieu Fenniak, Matthew Stamy, pubpub zz, Martin Thoma, Matthew Peveler, exiledkingcc, and pypdf Contributors. 2024. The pypdf library. https://pypi.org/project/pypdf/

[17] GROBID. 2008–2025. https://github.com/kermitt2/grobid. arXiv:1:dir:dab86b296e3c3216e2241968f0d63b68e8209d3c

[18] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779* (2024).

[19] Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802* (2025).

[20] Tam Harbert. 2021. https://mitsloan.mit.edu/ideas-made-to-matter/tapping-power-unstructured-data

[21] Conghui He, Wei Li, Zhenjiang Jin, Chao Xu, Bin Wang, and Dahua Lin. 2024. Opendatalab: Empowering general artificial intelligence with open datasets. *arXiv preprint arXiv:2407.13773* (2024).

[22] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* 43, 2 (2025), 1–55.

[23] Yulong Hui, Yao Lu, and Huanchen Zhang. 2024. Uda: A benchmark suite for retrieval augmented generation in real-world document analysis. *Advances in Neural Information Processing Systems* 37 (2024), 67200–67217.

[24] HURIDOCS. 2025. https://github.com/huridocs/pdf-document-layout-analysis

[25] Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. 2023. Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944* (2023).

[26] Gautier Izacard and Edouard Grave. 2020. Distilling knowledge from reader to retriever for question answering. *arXiv preprint arXiv:2012.04584* (2020).

[27] Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems* 37 (2024), 59532–59569.

[28] Saehan Jo and Immanuel Trummer. 2023. Demonstration of thalamusdb: answering complex sql queries with natural language predicates on multi-modal data. In *Companion of the 2023 International Conference on Management of Data*. 179–182.

[29] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering.. In *EMNLP (1)*. 6769–6781.

[30] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[31] Yuta Koreeda and Christopher D Manning. 2021. Capturing logical structure of visually structured documents with multimodal transition parser. *arXiv preprint arXiv:2105.00150* (2021).

[32] Yuta Koreeda and Christopher D Manning. 2021. ContractNLI: A dataset for document-level natural language inference for contracts. *arXiv preprint arXiv:2110.01799* (2021).

[33] Reagan J Lee, Samarth Goel, and Kannan Ramchandran. 2024. Quantifying positional biases in text embedding models. *arXiv preprint arXiv:2412.15241* (2024).

[34] Yibin Lei, Liang Ding, Yu Cao, Changtong Zan, Andrew Yates, and Dacheng Tao. 2023. Unsupervised dense retrieval with relevance-aware contrastive pre-training. *arXiv preprint arXiv:2306.03166* (2023).

[35] Julie Lemarié, Robert F Lorch Jr, and Marie-Paule Péry-Woodley. 2012. Understanding how headings influence text processing. *Discours. Revue de linguistique, psycholinguistique et informatique. A journal of linguistics, psycholinguistics and computational linguistics* 10 (2012).

[36] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[37] Minghao Li, Yiheng Xu, Lei Cui, Shaohan Huang, Furu Wei, Zhoujun Li, and Ming Zhou. 2020. Docbank: A benchmark dataset for document layout analysis. *arXiv preprint arXiv:2006.01038* (2020).

[38] Xiangci Li and Jessica Ouyang. 2024. How Does Knowledge Selection Help Retrieval Augmented Generation? *arXiv preprint arXiv:2410.13258* (2024).

[39] Zhuoqun Li, Xuanang Chen, Haiyang Yu, Hongyu Lin, Yaojie Lu, Qiaoyu Tang, Fei Huang, Xianpei Han, Le Sun, and Yongbin Li. 2024. Structrag: Boosting knowledge intensive reasoning of llms via inference-time hybrid information structurization. *arXiv preprint arXiv:2410.08815* (2024).

[40] Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G Parameswaran. 2025. Twix: Automatically reconstructing structured data from templatized documents. *arXiv preprint arXiv:2501.06659* (2025).

[41] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeighami, Aditya G Parameswaran, and Eugene Wu. 2025. Querying templatized document collections with large language models. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 2422–2435.

[42] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A declarative system for optimizing ai workloads. *arXiv preprint arXiv:2405.14696* (2024).

[43] Jerry Liu. 2022. LlamaIndex. https://doi.org/10.5281/zenodo.1234

[44] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics* 12 (2024), 157–173.

[45] LlamaParse. 2025. https://developers.llamaindex.ai/python/cloud/llamaparse/

[46] Jiefeng Ma, Jun Du, Pengfei Hu, Zhenrong Zhang, Jianshu Zhang, Huihui Zhu, and Cong Liu. 2023. Hrdoc: Dataset and baseline method toward hierarchical reconstruction of document structures. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 1870–1877.

[47] Tomohiro Manabe and Keishi Tajima. 2015. Extracting logical hierarchical structure of HTML documents based on headings. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1606–1617.

[48] Steve McDowell. 2023. https://www.forbes.com/sites/stevemcdowell/2023/03/09/komprise-unleashes-fresh-insights-about-your-unstructured-data/?sh=5f444c474aa9

[49] Junbo Niu, Zheng Liu, Zhuangcheng Gu, Bin Wang, Linke Ouyang, Zhiyuan Zhao, Tao Chu, Tianyao He, Fan Wu, Qintong Zhang, et al. 2025. Mineru2. 5: A decoupled vision-language model for efficient high-resolution document parsing. *arXiv preprint arXiv:2509.22186* (2025).

[50] OpenAI. 2024. text-embeddig-3-small. https://platform.openai.com/docs/guides/embeddings/

[51] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4171–4184.

[52] pdfium. 2025. https://github.com/chromium/pdfium

[53] pdfminer.six. 2019. https://pdfminersix.readthedocs.io/en/latest/

[54] Lee Petherbridge and Christopher A Cotropia. 2016. Should Your Law Review Article Have an Abstract and Table of Contents: An Empirical Analysis. *Miss. LJ*

85 (2016), 295.
[55] Birgit Pfitzmann, Christoph Auer, Michele Dolfi, Ahmed S Nassar, and Peter Staar. 2022. Doclaynet: A large human-annotated dataset for document-layout segmentation. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*. 3743–3751.
[56] Chris Mattmann (PI), Wayne Burke, Dustin Graf, Tim Allison, Ryan Stonebraker, Mike Milano, Philip Southam, Anastasia Menshikova, and Peter Wyatt. 2021. SAFEDOCS (CC-MAIN-2021-31-PDF-UNTRUNCATED). https://digitalcorpora.org/corpora/file-corpora/cc-main-2021-31-pdf-untruncated/
[57] PyMuPDF. 2025. https://github.com/pymupdf/PyMuPDF
[58] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).
[59] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and trends® in information retrieval* 3, 4 (2009), 333–389.
[60] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*.
[61] Tobias Schimanski, Imene Kolli, Jingwei Ni, Yu Fan, Ario Saeid Vaghefi, Elliott Ash, and Markus Leippold. 2026. pdfQA: Diverse, Challenging, and Realistic Question Answering over PDFs. *arXiv preprint arXiv:2601.02285* (2026).
[62] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G Parameswaran, and Eugene Wu. 2024. Docetl: Agentic query rewriting and evaluation for complex document processing. *arXiv preprint arXiv:2410.12189* (2024).
[63] Zejiang Shen, Kyle Lo, Lucy Lu Wang, Bailey Kuehl, Daniel S Weld, and Doug Downey. 2022. VILA: Improving structured content extraction from scientific PDFs using visual layout groups. *Transactions of the Association for Computational Linguistics* 10 (2022), 376–392.
[64] Jeremy Singer-Vine and The pdfplumber contributors. 2025. *pdfplumber*. https://github.com/jsvine/pdfplumber
[65] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136* (2025).
[66] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 645–654.
[67] Ji Sun, Guoliang Li, Peiyao Zhou, Yihui Ma, Jingzhe Xu, and Yuan Li. 2025. Agenticdata: An agentic data analytics system for heterogeneous data. *arXiv preprint arXiv:2508.05002* (2025).
[68] Zhaoze Sun, Qiyan Deng, Chengliang Chai, Kaisen Jin, Xinyu Guo, Han Han, Ye Yuan, Guoren Wang, and Lei Cao. 2025. Quest: Query optimization in unstructured document analysis. *arXiv preprint arXiv:2507.06515* (2025).
[69] Chelse Swoopes, Ziwei Gu, and Elena L Glassman. 2025. Interface Design to Support Legal Reading and Writing: Insights from Interviews with Legal Experts. *arXiv preprint arXiv:2509.24854* (2025).
[70] Unstructured. 2025. https://unstructured.io/
[71] Matthias Urban and Carsten Binnig. 2024. Demonstrating CAESURA: language models as multi-modal query planners. In *Companion of the 2024 International Conference on Management of Data*. 472–475.
[72] Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, Zhiyuan Zhao, Rui Xu, Kaiwen Liu, Yuan Qu, Fukai Shang, et al. 2024. Mineru: An open-source solution for precise document content extraction. *arXiv preprint arXiv:2409.18839* (2024).
[73] Jiawei Wang, Kai Hu, Zhuoyao Zhong, Lei Sun, and Qiang Huo. 2024. Detect-order-construct: A tree construction based approach for hierarchical document structure analysis. *Pattern Recognition* 156 (2024), 110836.
[74] Shu Wang, Yingli Zhou, and Yixiang Fang. 2025. BookRAG: A Hierarchical Structure-aware Index-based Approach for Retrieval-Augmented Generation on Complex Documents. *arXiv preprint arXiv:2512.03413* (2025).
[75] Xinyu Wang, Jijun Chi, Zhenghan Tai, Tung Sum Thomas Kwok, Hailin He, Zhuhong Li, Yuchen Hua, Muzhi Li, Peng Lu, Suyucheng Wang, et al. 2025. Finsage: A multi-aspect rag system for financial filings question answering. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. 6144–6152.
[76] 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)*. 2518–2533.
[77] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 1192–1200.
[78] Antonio Jimeno Yepes, Yao You, Jan Milczek, Sebastian Laverde, and Renyu Li. 2024. Financial report chunking for effective retrieval augmented generation. *arXiv preprint arXiv:2402.05131* (2024).
[79] Tan Yu, Anbang Xu, and Rama Akkiraju. 2024. In defense of rag in the era of long-context language models. *arXiv preprint arXiv:2409.01666* (2024).
[80] Junyuan Zhang, Qintong Zhang, Bin Wang, Linke Ouyang, Zichen Wen, Ying Li, Ka-Ho Chow, Conghui He, and Wentao Zhang. 2025. Ocr hinders rag: Evaluating the cascading impact of ocr on retrieval-augmented generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 17443–17453.
[81] Mingtian Zhang, Yu Tang, and PageIndex Team. 2025. PageIndex: Next-Generation Vectorless, Reasoning-based RAG. *PageIndex Blog* (September 2025). https://pageindex.ai/blog/pageindex-intro.
[82] Q Zhang, VSJ Huang, B Wang, J Zhang, Z Wang, H Liang, S Wang, M Lin, C He, and W Zhang. [n.d.]. Document parsing unveiled: Techniques, challenges, and prospects for structured information extraction. arXiv 2024. *arXiv preprint arXiv:2410.21169* ([n. d.]).
[83] Yibo Zhao, Jiapeng Zhu, Zichen Ding, and Xiang Li. 2026. GRACE: Reinforcement Learning for Grounded Response and Abstention under Contextual Evidence. *arXiv preprint arXiv:2601.04525* (2026).
[84] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.
[85] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. 2019. Publaynet: largest dataset ever for document layout analysis. In *2019 International conference on document analysis and recognition (ICDAR)*. IEEE, 1015–1022.
[86] Zijie Zhong, Hanwen Liu, Xiaoya Cui, Xiaofan Zhang, and Zengchang Qin. 2025. Mix-of-granularity: Optimize the chunking granularity for retrieval-augmented generation. In *Proceedings of the 31st International Conference on Computational Linguistics*. 5756–5774.

A PROMPTS

A.1 Prompts for LLM-based SHT Construction

LLM-text SHT Construction Prompt

Task:
Generate a detailed and structured Table of Contents (ToC) for the given document. The ToC must accurately capture **all headers** (including the document title, section headers, subsection headers, sub-subsection headers, etc.) and clearly represent their **hierarchical structure**.

Instructions:
1. Do not modify any headers.
- Use the exact text of each header as it appears in the document.
2. Use the following Markdown format to show hierarchy:

```markdown
# Title
## Section Header
### Subsection Header
#### Sub-subsection Header
...
```

- The number of '#' symbols indicates the hierarchy level.
- Include one space between the '#' symbols and the header text.
- Preserve the order in which the headers appear in the document.

3. Output only the Table of Contents, with nothing else.

You are given the following document:
[Begin of the document]
{doc}

[End of the document]

Your Table of Contents:
<your ToC here>

The prompt used by LLM-vision is almost the same, except it replaces the text in placeholder `{doc}` with a batch of page images following the document's reading order.

A.2 Prompts for LLM-as-a-Judge

LLM-as-a-Judge for *Qasper*

Please act as an impartial judge and evaluate the quality of the answer provided by an AI assistant to a question. You will be given the question, a reference answer, and the assistant's answer. Compare the assistant's answer with the reference answer and grade it using the rubric below. Provide a grade from 0 to 3 and an explanation for your grade. Your response should be formatted as "Rating: [[your rating]]. Explanation: [[your explanation]]." Example ratings for comparison are provided below.

Grading Rubric:
- Score 0: The answer is completely incorrect or irrelevant.
- Score 1: The answer is partially correct, addressing one aspect of the question.
- Score 2: The answer mostly covers the question but misses or incorrectly addresses a critical point.
- Score 3: The answer fully matches the reference answer, covering all key points.

Example Question:
What human evaluation metrics were used in the paper?

Example Reference Answer:
Rating questions on a scale of 1-5 based on fluency of language used and relevance of the question to the context.

Example Assistant Answers and Ratings:
Assistant Answer 1: "I don't know the answer."
Rating: [[0]]. Explanation: The answer does not address the question.

Assistant Answer 2: "The human evalution metrics had a rating scale of 1-5."
Rating: [[1]]. Explanation: The answer is relevant to the topic but does not fully address the question.

Assistant Answer 3: "Fluency and relevance."
Rating: [[2]]. Explanation: The answer covers the two metrics but lacks specific details, and the answer misses the rating scale of the metrics.

Assistant Answer 4: "Fluency of language used and relevance of the query to the context, rated on a scale of 1-5."
Rating: [[3]]. Explanation: The answer fully matches the reference, covering all key aspects.

Now, evaluate the following assistant's answer:

Question:
{query}

Reference Answer:
{reference}

Assistant's Answer:
{answer}

Your rating: "Rating: [[0/1/2/3]]. Explanation: [[your explanation]]."

LLM-as-a-Judge for *FinanceBench*

Please act as an impartial judge and evaluate the quality of the answer provided by an AI assistant to a question. You will be given the question, a reference answer, and the assistant's answer. Compare the assistant's answer with the reference answer and grade it using the rubric below. Provide a grade from 0 to 3 and an explanation for your grade. Your response should be formatted as "Rating: [[your rating]]. Explanation: [[your explanation]]." Example ratings for comparison are provided below.

Grading Rubric:
- Score 0: The answer is completely incorrect or irrelevant.
- Score 1: The answer is partially correct, addressing one aspect of the question.
- Score 2: The answer mostly covers the question but misses or incorrectly addresses a critical point.
- Score 3: The answer fully matches the reference answer, covering all key points.

Example Question:
What drove operating margin change as of FY2022 for 3M? If operating margin is not a useful metric for a company like this, then please state that and explain why.

Example Reference Answer:
Operating Margin for 3M in FY2022 has decreased by 1.7% primarily due to:
-Decrease in gross Margin
-mostly one-off charges including Combat Arms Earplugs litigation, impairment related to exiting PFAS manufacturing, costs related to exiting Russia and divestiture-related restructuring charges

Example Assistant Answers and Ratings:
Assistant Answer 1: "I don't know the answer."
Rating: [[0]]. Explanation: The answer does not address the question.

Assistant Answer 2: "Litigation charges."
Rating: [[1]]. Explanation: The answer is relevant to the topic

but does not fully address the question.

Assistant Answer 3: “Decrease in gross Margin and Combat Arms Earplugs litigation charges.”
Rating: [[2]]. Explanation: The answer covers two driven causes but misses others.

Assistant Answer 4: “Gross Margin Decrease, Combat Arms Earplugs litigation, impairment realted to exiting PFAS manufacturing, costs related to exiting Russia, and divestiture-related restructuring charges.”
Rating: [[3]]. Explanation: The answer fully matches the reference, covering all key aspects.

Now, evaluate the following assistant’s answer:

Question:
{query}

Reference Answer:
{reference}

Assistant’s Answer:
{answer}

Your rating: “Rating: [[0/1/2/3]]. Explanation: [[your explanation]]”