applied sciences

MDPI

Article

SelfAudit: Heterogeneous Multi-Dimensional Reflection for Complex Document Auditing

Fei Chen [1], Chenyu Zhang [1,*], Meng Jiang [2] and Yun Chen [1]

[1] School of Control and Computer Engineering, North China Electric Power University, Beijing 102200, China; chenfei@ncepu.edu.cn (F.C.); yunc9862@163.com (Y.C.)
[2] Department of Computer Science, Hong Kong Baptist University, Hong Kong, China; 24253030@life.hkbu.edu.hk
* Correspondence: zcy335100@163.com

Abstract

Complex document auditing requires the integration of information across chapters, verification of multiple rules, and fine-grained reasoning over long contexts. Existing LLM-based methods often struggle with task decomposition, distinguishing different types of errors, and maintaining the quality of intermediate results. We propose SelfAudit, an end-to-end multi-agent framework built around Heterogeneous Multi-Dimensional Confidence Decomposition with Dependency-Aware Error Attribution (HCD-EA). HCD-EA estimates a three-dimensional confidence vector consisting of Retrieval Fidelity, Evidence Support, and Rule Compliance, and uses the minimum of these dimensions to identify the weakest part of the auditing process. If the confidence falls below a predefined threshold, HCD-EA uses the predefined dependency graph, locates the earliest failing node in topological order, and re-executes only the affected sub-pipeline. A heterogeneous verifier provides an independent confidence estimate, reducing the risk of circular self-evaluation. Experiments on ContractNLI, CUAD, and LegalBench-RAG show that SelfAudit outperforms 13 baselines across five categories, achieving a weighted F1 of 0.844 on ContractNLI. Ablation studies demonstrate the contribution of each component, while selective re-execution reduces LLM calls by 17% compared with blind retry.

Keywords: document auditing; multi-agent framework; confidence decomposition; error attribution; heterogeneous verification; large language models

1. Introduction

With the ongoing digital transformation of enterprises and institutions, the volume of complex documents, including financial contracts, legal agreements, and corporate reports, has grown rapidly, increasing the demand for automated auditing systems. These documents are often lengthy, structurally complex, and subject to strict regulatory requirements. Auditing systems must therefore integrate information across multiple sections, perform multi-step reasoning, and conduct fine-grained rule verification over long contexts. Traditional manual auditing is inefficient at scale, difficult to maintain consistently, and costly in terms of labor, as illustrated in Figure 1.

Recent advances in large language models (LLMs) [1] have demonstrated strong capabilities in natural language understanding and reasoning, creating new opportunities for automated document auditing. Through prompt-based interaction, LLMs can assist with tasks such as clause extraction, semantic comparison, and risk identification, thereby reducing part of the manual workload. However, auditing complex documents involves

Check for updates

Academic Editor: Andrea Prati

Received: 14 September 2026
Revised: 23 October 2026
Accepted: 23 October 2026
Published: 27 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

more than classification or question answering. It requires long-context understanding, cross-section information integration, and multi-step rule verification within a coherent reasoning process. When applied directly to such scenarios, LLMs remain susceptible to limited context coverage [2], fragmented reasoning, and cumulative error propagation, making it difficult to meet the stringent requirements for accuracy and interpretability in financial and legal applications.

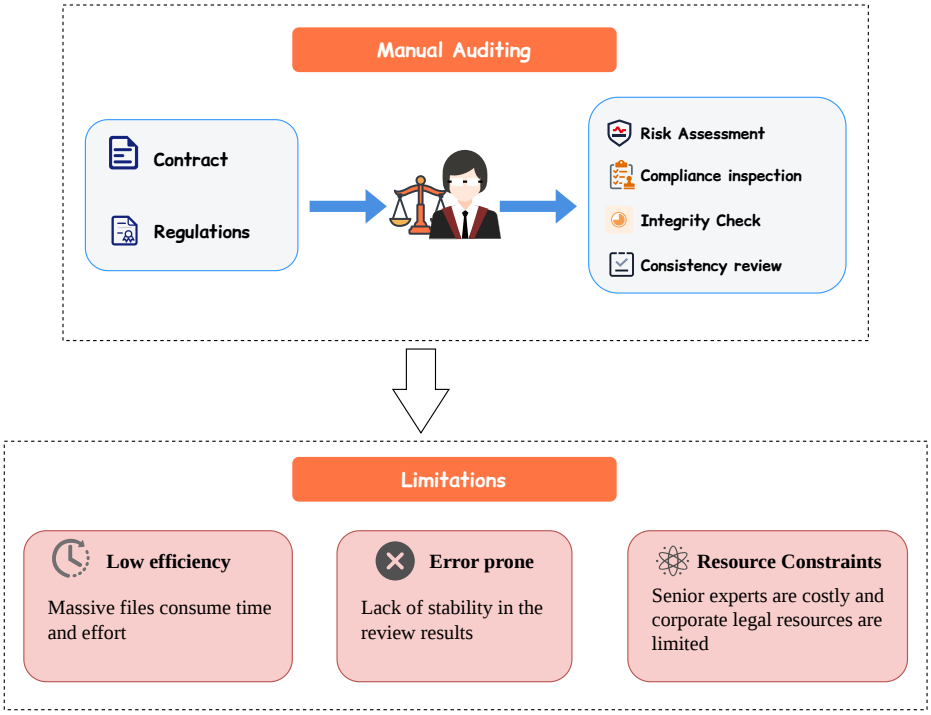

Figure 1. Manual contract review process and defects. Blue arrows show the flow from contract and regulatory inputs through manual review to the review activities. The downward outlined arrow introduces the limitations of manual auditing.

Single-agent approaches can improve reasoning through explicit reasoning strategies, but they remain insufficient for addressing these challenges. Chain-of-Thought (CoT) [3], for instance, improves reasoning coherence but lacks an explicit mechanism for decomposing complex auditing tasks, which can lead to missed auditing points. Generating multiple reasoning paths and aggregating consistent answers has also been shown to improve reliability [4]. ReAct [5] incorporates tool interaction into the reasoning process, but may still struggle to locate relevant evidence across sections of long documents, with errors potentially accumulating over successive reasoning steps. Domain-specific models such as LegalBERT [6], ChatLaw [7], and SaulLM [8] have advanced legal-domain pre-training, yet they similarly treat auditing as end-to-end classification without explicitly modeling the verification process. More importantly, these approaches generally treat auditing as a single-model reasoning problem and provide limited supervision or correction of intermediate results, allowing errors to propagate throughout the process.

RAG-based [9] methods can improve evidence coverage by retrieving relevant text fragments, reducing information omission. Subsequent work has refined this paradigm through structure-aware retrieval [10,11] and self-reflective mechanisms that assess retrieved evidence and the reasoning based on it [12–17]. However, these reflection mechanisms operate primarily at the token or output level and cannot distinguish retrieval failures from errors in rule interpretation [18]. Moreover, using the same model for both generation and evaluation introduces circularity into self-supervision. Expanding the retrieval scope can improve

recall, but it may also introduce substantial noise, making it difficult to balance evidence coverage and retrieval precision.

Multi-agent frameworks such as AutoGen [19], MetaGPT [20], CAMEL [21], and Generative Agents [22] have demonstrated that agent coordination can improve task execution consistency. In the legal domain, PAKTON [23] combined multi-agent orchestration with RAG for long-form legal agreement processing. Despite these advances, existing multi-agent systems lack explicit quality control over intermediate outputs, allowing errors to propagate across agents without systematic detection or correction. These gaps motivate an auditing framework that integrates structured task decomposition, dimension-level reflection, and process-level error correction within a unified closed loop.

To address these limitations, we propose SelfAudit, a reflection-driven multi-agent framework for complex document auditing. The framework consists of four specialized agents that collaboratively perform the auditing process. The Task Planning Agent (TPA) decomposes complex auditing tasks into structured subtasks, aligns them with the corresponding auditing rules, and represents each subtask as a triple (objective, rule, audit-point), specifying the task objective, applicable rule, and auditing point. The Evidence Retrieval Agent (ERA) performs structural document segmentation using a hierarchical semantic tree, enabling cross-section evidence localization and the construction of audit evidence chains. The Inspector Agent (IA) evaluates documents from multiple perspectives, including numerical correctness, content completeness, semantic consistency, and risk, and then generates the final audit report. The Self-Reflection Agent (SRA) evaluates intermediate outputs at the stages of task planning, evidence retrieval, and auditing, and performs corrections when necessary. Based on these confidence estimates, the framework controls the execution process and selectively reactivates relevant stages when low-confidence outputs are detected, enabling iterative refinement of the auditing workflow.

The main contributions of this work are threefold. We propose a structured multi-agent auditing framework that decomposes complex audit queries into executable (objective, rule, audit-point) triples, improving task coverage and execution control. We introduce a hierarchical semantic tree for document segmentation and an evidence retrieval method that combines hybrid indexing with fused reranking, preserving structural relationships across chapters while supporting reliable cross-chapter evidence retrieval. We further propose a rubric-guided stage-level reflection mechanism with confidence-driven process control, enabling iterative detection and correction of errors during the auditing process.

2. Materials and Methods

2.1. Overview

SelfAudit addresses the task of auditing a document D against a set of regulatory requirements. Given an audit query Q and a document D consisting of n sections $\{s_1, \ldots, s_n\}$, the system produces a verdict $v \in \{\text{Entailment}, \text{Contradiction}, \text{Not Mentioned}\}$, together with an evidence chain E and a confidence score c. The auditing process is formulated as:

$$\text{SelfAudit}(Q, D, M) \rightarrow (v, E, c)$$

where $M = \{m_1, \ldots, m_k\}$ denotes the predefined rule set. Four specialized agents execute the auditing process. TPA decomposes audit queries into structured subtasks, ERA retrieves task-relevant evidence, IA performs multi-dimensional verification, and SRA monitors and refines intermediate outputs through HCD-EA. By providing feedback throughout the pipeline, SRA enables iterative refinement of the auditing process. Figure 2 illustrates the overall architecture and interactions among the four agents.

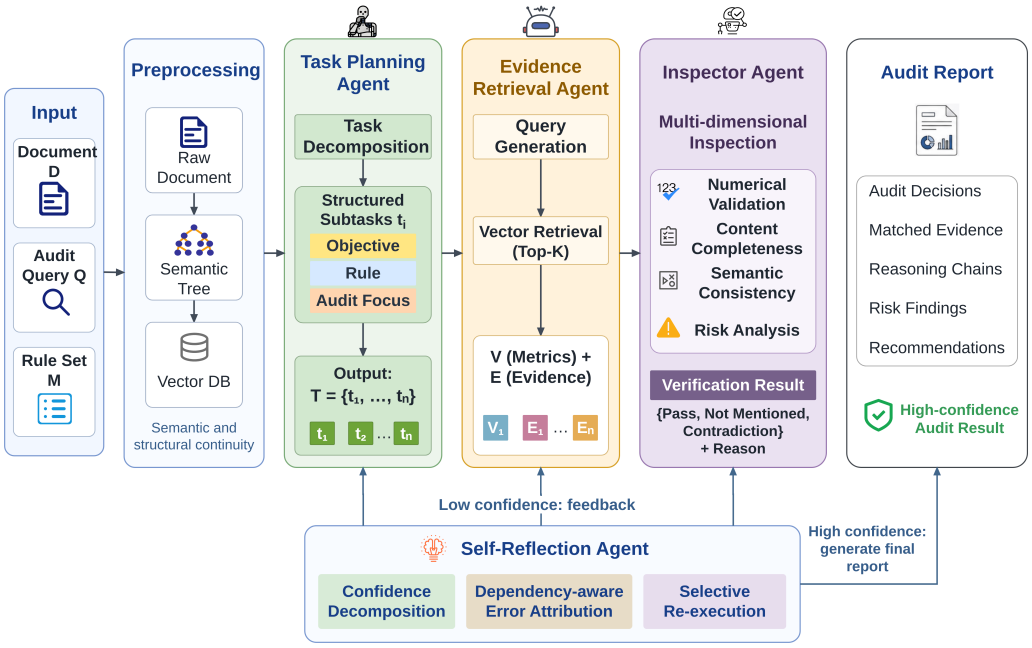

Figure 2. Overview of the SelfAudit framework. The auditing pipeline consists of four specialized agents: TPA decomposes audit queries into triples; ERA retrieves cross-chapter evidence through hierarchical semantic tree-based vector retrieval; IA performs multi-dimensional verification across D1–D4; and SRA, equipped with HCD-EA, monitors intermediate outputs at each stage and triggers feedback-driven selective re-execution when confidence falls below the threshold. Main-flow arrows indicate processing order; feedback arrows indicate selective re-execution, and the high-confidence path leads to the final report. Colors distinguish modules and field types rather than numerical confidence. Ellipses denote additional subtasks or evidence entries of the same type.

2.2. Document Preprocessing

The preprocessing module extracts textual and structural information from input documents. For native PDF documents, layout analysis identifies text elements, heading hierarchies, and paragraph structures. For scanned documents, OCR is used to recover text from page images. Headers, footers, redundant symbols, and duplicated characters are removed to reduce noise during subsequent retrieval.

To support semantic retrieval, the system organizes the preprocessed document as a hierarchical tree consisting of a document root, section nodes, subsection nodes, and paragraph blocks. Section titles and their levels are obtained from document preprocessing. The section structure is constructed recursively: each section supplies its title, text, page information, and nested subsections, and each subsection is attached to its parent section.

Within each section, blank-line separators (`\n\n`) define paragraph boundaries, and empty or whitespace-only paragraphs are discarded. Each paragraph block remains associated with its section and is stored with an identifier, text, page number, and section title. When section information is unavailable, the full text is split into paragraph blocks and attached directly to the document root. A pre-trained embedding model maps each block to a dense vector, which is stored in the vector database together with its structural metadata. This organization preserves the section context of the retrieved evidence.

2.3. Task Planning Agent

TPA decomposes complex auditing queries into executable subtasks and assigns the corresponding auditing rules. Its inputs include the user query Q, the preprocessed document collection D, the predefined rule set M, and reflection feedback from SRA. The decomposition is formalized as:

$$T = \mathrm{TPA}(Q, D, M, \mathrm{Reflection})$$

Each subtask t_i in T is represented as a domain-specific triple:

$$t_i = (\mathrm{obj}_i, \mathrm{rule}_i, \mathrm{point}_i)$$

where obj_i denotes the auditing objective, or what needs to be checked; rule_i denotes the applicable rule, or the criterion used to evaluate the objective; and point_i denotes the specific auditing focus, or the items to be examined. This representation explicitly connects each subtask with its objective, evaluation criterion, and auditing focus, providing a structured basis for subsequent retrieval and verification.

2.4. Evidence Retrieval Agent

The Evidence Retrieval Agent (ERA) performs semantic retrieval over the document collection D based on each subtask and reflection feedback from the Self-Reflection Agent (SRA). For each subtask, ERA first maps the task description to a semantic retrieval query q_i, computes the cosine similarity between the query vector and document vectors, and selects the Top-k text chunks as candidate retrieval regions. The retrieval process is formalized as:

$$q_i = \mathrm{Embed}(t_i), \quad d_j = \mathrm{Embed}(s_j)$$

$$\mathrm{Candidates}(t_i) = \mathrm{TopK}_{d_j \in D}[\cos(q_i, d_j)]$$

ERA then applies a hybrid indexing and fusion reranking mechanism that combines semantic and keyword-based retrieval to improve candidate evidence recall. Fusion reranking further refines the candidates according to their relevance to the subtask objective and applicable audit rules. From the final retrieval results, ERA extracts the required metric information to construct V. It also preserves structural metadata in E, including source document identifiers, section locations, and adjacent text chunks, to retain the context required for subsequent auditing. For each subtask t_i, the output of ERA is formulated as:

$$V, E = \mathrm{ERA}(t_i, D, \mathrm{Reflection})$$

2.5. Inspector Agent

IA performs auditing based on the subtask list generated by TPA, combining the predefined rules with the evidence retrieved by ERA. It conducts verification along four dimensions. Numerical Correctness (D1) checks whether numerical information satisfies the relevant computational rules. Content Completeness (D2) determines whether required clauses or information are missing. Semantic Consistency (D3) examines whether conflicting expressions occur across different sections. Risk Analysis (D4) identifies anomalous or high-risk expressions that may introduce compliance risks. For each subtask, the verification process is formalized as:

$$\mathrm{Result}_i = \mathrm{IA}(t_i, V_i, E_i, \mathrm{Reflection})$$

2.6. Self-Reflection Agent with HCD-EA

SRA is the core component of SelfAudit. Unlike existing reflection mechanisms that mainly evaluate or rewrite model outputs, SRA acts as a diagnostic module that identifies not only whether an output is problematic, but also where the problem arises and what causes it. This design addresses three limitations of existing reflection mechanisms: self-supervision circularity, where the same LLM generates and evaluates an output; single-scalar blindness, where a single confidence score cannot distinguish retrieval failure, insufficient evidence,

and rule misinterpretation; and blind re-execution, where methods such as Reflexion and Self-Refine may rerun an entire stage or rewrite the full output when confidence is low, increasing computational cost while potentially reproducing the same error. Huang et al. [18] further showed that LLMs cannot reliably self-correct reasoning errors, underscoring the limitations of self-supervised evaluation. Figure 3 illustrates the HCD-EA mechanism, which combines multi-dimensional confidence decomposition, dependency-aware error attribution, and heterogeneous verification.

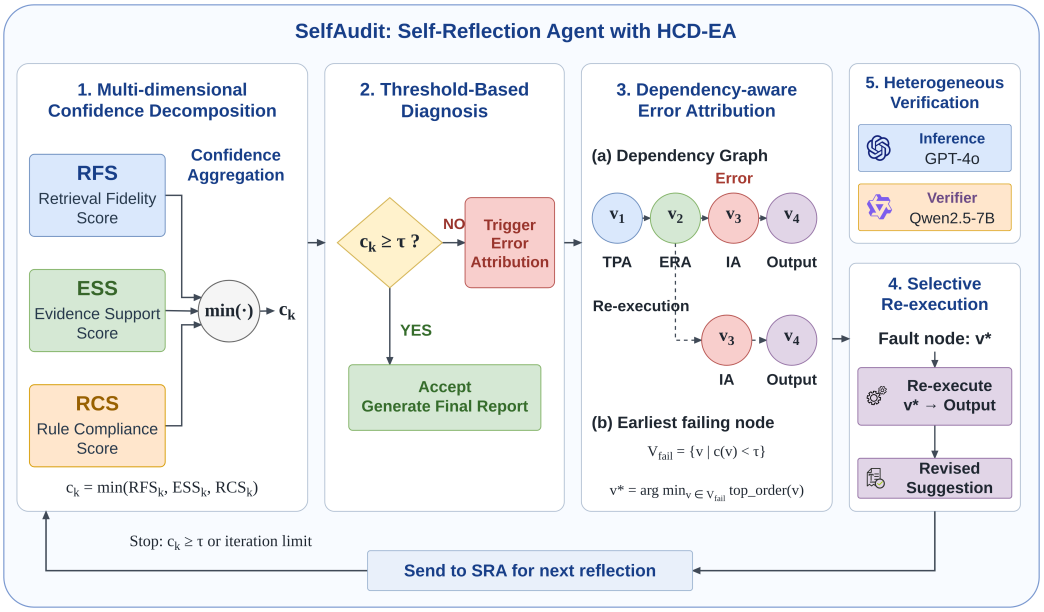

Figure 3. The HCD-EA reflection mechanism. Multi-dimensional confidence decomposition: SRA estimates a three-dimensional confidence vector (RFS, ESS, RCS) and aggregates it as $c_k = \min(\text{RFS}, \text{ESS}, \text{RCS})$, where the weakest dimension determines the overall confidence. Dependency-aware error attribution: when $c_k < \tau$, SRA identifies all nodes with aggregate confidence below the threshold, determines the topologically earliest failing node in the dependency graph $G = (V, E)$, and selectively re-executes the sub-pipeline from that node to the output, targeting the likely source of cascading errors. The symbol v^* denotes the topologically earliest failing node, written as v_{earliest} in the text. Arrows indicate process flow or dependencies; the dashed path illustrates selective re-execution from the selected node, and the bottom return path indicates the next reflection iteration.

2.6.1. Multi-Dimensional Confidence Decomposition and Error Attribution

SRA estimates a separate three-dimensional confidence vector for each node corresponding to the TPA, ERA, IA, and final-verdict outputs. Each assessment uses the input context and output of that node:

$$c_k = (\text{RFS}_k, \text{ESS}_k, \text{RCS}_k)$$

The three dimensions capture distinct aspects of audit quality. The Retrieval Fidelity Score (RFS) $\in [0, 1]$ measures the semantic alignment between the retrieved evidence and the task objective. It is computed as a weighted combination of embedding cosine similarity and an LLM-based relevance judgment:

$$\text{RFS}_k = \alpha \cdot \cos(\text{Embed}(q_k), \text{Embed}(E_k)) + (1 - \alpha) \cdot \text{LLM_judge}(q_k, E_k)$$

where $\alpha \in [0, 1]$ controls the relative contributions of embedding similarity and the LLM-based relevance assessment.

The Evidence-Support Score (ESS) $\in [0, 1]$ evaluates whether the retrieved evidence logically supports the current verification conclusion. Its inputs are the evidence and the

conclusion. The scoring rubric uses 1.0, 0.5, and 0.0 as reference points for full support, partial support with an evidential gap, and contradictory or irrelevant evidence, respectively. For this assessment, the first three retrieved evidence passages are concatenated, and the evidence text supplied to the prompt is limited to 2000 characters.

The Rule-Compliance Score (RCS) $\in [0, 1]$ evaluates the current verification conclusion against the applicable audit rules. Its inputs are the rules and the conclusion. The corresponding rubric uses 1.0 for full compliance, 0.5 for partial compliance with a gap, and 0.0 for a rule violation. Qwen2.5-7B-Instruct (Qwen team, Alibaba Cloud, Hangzhou, China) performs both assessments at temperature 0. The returned JSON contains the fields `support_score` and `compliance_score` for ESS and RCS, respectively, together with `reasoning`. The numerical fields are parsed as the corresponding scores. If a valid verification result is unavailable or a model call or response parsing fails, the affected score is set to 0.5.

The aggregate confidence c_k, a scalar distinct from the vector $\mathbf{c}_k$, is defined as the minimum of the three dimensions:

$$c_k = \min(\mathrm{RFS}_k, \mathrm{ESS}_k, \mathrm{RCS}_k)$$

This formulation reflects the auditing principle that a conclusion is only as reliable as its weakest component. For example, if RFS = 0.91 and ESS = 0.85 but RCS = 0.62, the aggregate confidence is $c_k = 0.62$, indicating that rule verification is the limiting factor. In this case, SRA re-executes only the rule verification stage rather than the entire pipeline.

Minimum aggregation requires all three dimensions to meet the acceptance threshold: $\min(\mathrm{RFS}, \mathrm{ESS}, \mathrm{RCS}) \geq \tau$ if and only if RFS $\geq \tau$, ESS $\geq \tau$, and RCS $\geq \tau$. A high score in one dimension therefore cannot compensate for insufficient evidence support or rule compliance in another. In the example above, the arithmetic mean is approximately 0.793 and would exceed $\tau = 0.75$, whereas the minimum retains the low rule-compliance score of 0.62 and triggers reflection.

When the aggregate confidence c_k falls below the threshold $\tau = 0.75$, calibrated on a held-out validation set of 200 samples, SRA performs error attribution using an explicit dependency graph $G = (V, E)$. Each node $v \in V$ represents an intermediate output, and each edge $(u \to v) \in E$ denotes a dependency between two outputs. Each node is associated with its decomposed confidence vector. The aggregate confidence of each node is computed from its own RFS, ESS, and RCS scores, and these node-specific aggregates are used to form the failure set. Because errors can propagate downstream, a failure at an upstream stage may reduce the confidence of subsequent stages. Re-executing from the earliest failing node in topological order therefore targets the likely source of the error and allows downstream outputs to be recomputed.

The dependency graph is derived from the fixed agent topology of the SelfAudit pipeline. The node set V contains four nodes corresponding to the outputs of TPA, ERA, IA, and the final verdict. The graph is constructed once during initialization and remains fixed throughout the auditing process.

Formally, SRA first constructs the failure set containing all nodes whose aggregate confidence falls below the threshold:

$$V_{\text{fail}} = \{v \in V \mid c(v) < \tau\}$$

SRA then identifies the topologically earliest node in the failure set:

$$v_{\text{earliest}} = \arg\min_{v \in V_{\text{fail}}} \text{top_order}(v)$$

and re-executes the sub-pipeline from v_{earliest} to the output. When $|V_{\text{fail}}| = 1$, the failure is isolated, and re-execution starts from that node and updates its downstream outputs.

When $|V_{\text{fail}}| > 1$, the pattern of downstream failures suggests that the earliest failing node is the likely root cause, allowing the system to correct the upstream error without separately addressing each downstream symptom.

Process control follows a threshold-based selective re-execution strategy. When $c_k < \tau$, SRA locates the earliest failing node in the dependency graph and re-executes the affected sub-pipeline. The maximum number of iterations is set to $K = 3$, based on iterative reflection experiments showing that $K = 3$ achieves the best overall performance, while larger values provide diminishing returns due to redundant adjustments. Results that remain unresolved after K iterations are flagged for manual review.

2.6.2. HeterogeneousVerification

To mitigate self-supervision circularity, SRA employs a heterogeneous verifier. The main audit pipeline, including TPA, ERA, and IA, runs on a high-capability model (GPT-4o; OpenAI, San Francisco, CA, USA [24]), while SRA estimates confidence using a lightweight model with a different architecture and training data (Qwen2.5-7B-Instruct [25]). This separation introduces an independent judgment into the audit process, analogous to the distinction between auditor and reviewer roles in human auditing. The lightweight verifier adds limited computational overhead while providing an assessment based on different model characteristics.

For each of the four nodes, Qwen2.5-7B-Instruct receives the task objective, retrieved evidence, applicable rules, and the corresponding stage output. It returns a relevance judgment, an evidence-support score, a rule-compliance score, and an explanation. The relevance judgment is combined with embedding cosine similarity using $\alpha = 0.5$ to obtain RFS, while the support and compliance scores provide ESS and RCS. The resulting vector is stored under the corresponding node identifier and supplied to the dependency-aware attribution procedure.

2.7. Experimental Setup

2.7.1. Datasets

We evaluate SelfAudit on three benchmarks covering complementary capabilities required for document auditing: document-level classification, information extraction, and evidence retrieval. ContractNLI [26] contains 2091 test samples, each consisting of a full non-disclosure agreement (NDA), a legal hypothesis, and one of three labels: Entailment, Contradiction, and Not Mentioned. We report accuracy (Acc), weighted F1 (F1[W]), and class-level F1 scores for the three labels.

CUAD [27] provides expert annotations covering 41 contract review categories. We formulate each category as a binary span extraction task and report the area under the precision–recall curve (AUPR), micro-averaged F1, precision, and recall.

LegalBench-RAG [28] is used to independently evaluate the evidence retrieval component. The benchmark covers four legal domains, including NDAs, M&A agreements, commercial contracts, and consumer-facing privacy policies. We evaluate retrieval at $k \in \{1, 2, 4, 8, 16, 32, 64\}$ using Precision@k and Recall@k. Representative retrieval results are presented in the Results section.

2.7.2. Baselines

We compare SelfAudit with 13 baseline configurations, with GPT-4o used as the backbone model for the main ContractNLI and CUAD comparisons. The baselines cover prompting, reasoning, retrieval augmentation, reflection, and multi-agent approaches.

The prompting baselines include zero-shot (ZS) and few-shot (FS) prompting. Reasoning-based methods include Chain-of-Thought (CoT) [3], Self-Consistency [4],

and ReAct [5]. Retrieval-based methods include Standard RAG [9] and Self-RAG [12]. Reflection-based methods include Reflexion [13], Self-Refine [14], and CRITIC [15]. Multi-agent baselines include AutoGen [19], MetaGPT [20], and PAKTON [23]. For methods involving retrieval or iterative reasoning, the corresponding retrieval component is enabled under the same experimental setting.

2.7.3. Implementation and Metrics

GPT-4o is used for the Task Planning Agent (TPA), Evidence Retrieval Agent (ERA), and Inspector Agent (IA), while Qwen2.5-7B-Instruct [25] serves as the heterogeneous verifier in the Self-Reflection Agent (SRA). We use `text-embedding-3-large` (OpenAI, San Francisco, CA, USA) with 3072-dimensional embeddings for semantic retrieval. The confidence threshold is calibrated on 200 held-out samples from the ContractNLI training split. Candidate thresholds are 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, and 0.95. With the maximum number of reflection iterations fixed at $K = 3$ and the RFS weighting factor fixed at $\alpha = 0.5$, the threshold yielding the highest weighted F1 on the calibration set is selected, giving $\tau = 0.75$. The same threshold is used without further tuning in the other experiments that invoke reflection; the threshold-sensitivity experiment varies τ separately. ERA retrieves the top $k = 10$ text blocks. All experiments use a temperature of 0 and are conducted on four NVIDIA A100 80 GB GPUs (NVIDIA Corporation, Santa Clara, CA, USA).

For ContractNLI, we report accuracy (Acc), weighted F1 (F1[W]), and F1 for Entailment (F1[E]), Contradiction (F1[C]), and Not Mentioned (F1[N]). Weighted F1 is calculated as the support-weighted average of the class-level F1 scores, using the 1046 Entailment, 209 Contradiction, and 836 Not Mentioned test samples:

$$F1_W = \frac{1046F1_E + 209F1_C + 836F1_N}{2091}.$$

CUAD evaluation uses complete evidence-chain output and confidence-threshold scanning. Each contract–category pair is treated as an evaluation sample. The pipeline wrapper retains the evidence chain from the final `Verdict.evidence_chain` and returns it through the `evidence_chain` field read by the span-extraction module. Each prediction record contains the contract identifier, category, predicted text, character positions in the original contract, and extraction confidence, together with the associated gold spans. Positions use half-open character intervals $[\text{start}, \text{end})$. A missing evidence-chain field is not treated as an empty prediction.

Each sample uses the confidence returned by the pipeline wrapper as its extraction-result score. All candidate spans within sample i share this score, so that $s_{ij} = c_i$ for every candidate span j in that sample. The extraction-output threshold is set to 0.50 for precision, recall, and micro-F1, separately from the HCD-EA reflection threshold of 0.75.

For each category, all distinct confidence scores in the saved predictions are scanned in descending order. At each threshold, candidate spans whose confidence is no lower than the threshold are retained, and candidates with equal confidence are included together. Within each sample, the character positions covered by the retained predictions and those covered by the gold spans are each combined into a union to avoid counting duplicate or overlapping spans more than once.

At each threshold, correctly covered, incorrectly covered, and missed character positions are pooled across all samples in the category as TP, FP, and FN, respectively. Precision is calculated as $TP/(TP + FP)$, and recall as $TP/(TP + FN)$. The precision–recall curve is formed from these threshold-dependent points, starting at recall 0 and precision 1 when no predictions are selected. Trapezoidal integration gives the AUPR for each category, and

the reported AUPR is the arithmetic mean across the 41 categories. Precision–recall pairs computed separately for individual samples are not integrated to obtain AUPR.

Micro-precision, micro-recall, and micro-F1 are calculated at the fixed extraction-output threshold of 0.50 by pooling TP, FP, and FN across all test samples and categories. Micro-F1 is calculated as $2TP/(2TP + FP + FN)$. The AUPR calculation scans all distinct saved confidence scores rather than using only this fixed threshold.

Appendix A details the evaluation interface, span records, and a worked example of confidence-threshold scoring.

For LegalBench-RAG, we report Precision@k and Recall@k to evaluate retrieval independently. Computational cost is measured by the number of LLM inference calls and the average numbers of input and output tokens per sample. In the efficiency comparison, these quantities are normalized relative to Standard RAG.

3. Results

3.1. Overall Performance

3.1.1. ContractNLI Results

Table 1 presents the results on the ContractNLI test set. SelfAudit achieves an accuracy of 0.841 and a weighted F1 of 0.844. Compared with PAKTON, accuracy increases from 0.797 to 0.841, a gain of 4.4 percentage points, while weighted F1 increases from 0.797 to 0.844, a gain of 4.7 percentage points.

Table 1. Comparison of SelfAudit with baseline configurations on the ContractNLI test set. All methods use GPT-4o as the backbone LLM. E, C, and N denote Entailment, Contradiction, and Not Mentioned, with 1046, 209, and 836 test samples, respectively. F1[W] is the support-weighted average of the three class-level F1 scores. The best result in each column is shown in **bold**.

Category	Method	Acc	F1[W]	F1[E]	F1[C]	F1[N]
Prompting	ZS	0.612	0.644	0.749	0.416	0.570
Prompting	FS	0.664	0.683	0.737	0.473	0.667
Reasoning	CoT	0.682	0.703	0.762	0.488	0.682
Reasoning	Self-Consistency	0.712	0.718	0.778	0.502	0.698
Reasoning	ReAct	0.713	0.728	0.785	0.516	0.710
Retrieval	Standard RAG	0.738	0.751	0.812	0.548	0.726
Retrieval	Self-RAG	0.769	0.767	0.826	0.562	0.745
Reflection	Reflexion	0.748	0.761	0.820	0.556	0.738
Reflection	Self-Refine	0.774	0.774	0.832	0.570	0.752
Reflection	CRITIC	0.773	0.784	0.841	0.583	0.764
Multi-agent	AutoGen	0.780	0.780	0.838	0.578	0.758
Multi-agent	MetaGPT	0.779	0.791	0.847	0.592	0.770
Multi-agent	PAKTON	0.797	0.797	0.796	0.759	0.807
Ours	SelfAudit	**0.841**	**0.844**	**0.872**	**0.785**	**0.823**

At the class level, SelfAudit achieves F1 scores of 0.872, 0.785, and 0.823 for Entailment, Contradiction, and Not Mentioned, respectively. These scores exceed the corresponding PAKTON results by 7.6, 2.6, and 1.6 percentage points.

Among the prompting and reasoning baselines, CoT increases accuracy from 0.664 for FS to 0.682. Weighted F1 increases from 0.683 to 0.703, a gain of 2.0 percentage points. The corresponding improvements in Entailment, Contradiction, and Not Mentioned F1 are 2.5, 1.5, and 1.5 percentage points, respectively.

3.1.2. CUAD Results

Table 2 reports the results on CUAD. SelfAudit achieves an AUPR of 0.634 and a micro-averaged F1 of 0.704, exceeding the corresponding PAKTON results by 5.1 and 5.2 percentage points, respectively. SelfAudit also achieves a precision of 0.722 and a recall of 0.687.

The reported precision and recall yield a micro-averaged F1 of 0.704, consistent with the harmonic mean of the two values. The gains therefore extend beyond document-level classification to span-level information extraction, indicating that SelfAudit can improve extraction performance across the 41 contract review categories.

Table 2. Comparison of SelfAudit with selected baselines on CUAD. All methods use GPT-4o as the backbone LLM. The best result in each column is shown in **bold**. AUPR is macro-averaged over 41 categories using threshold-dependent character coverage. Precision, recall, and F1 are micro-averaged at an extraction-output threshold of 0.50.

Method	AUPR	Micro-F1	Precision	Recall
GPT-4o ZS	0.482	0.556	0.562	0.550
Standard RAG	0.524	0.598	0.610	0.587
Self-RAG	0.548	0.622	0.638	0.607
PAKTON	0.583	0.652	0.668	0.637
SelfAudit	**0.634**	**0.704**	**0.722**	**0.687**

3.2. Component and Robustness Results

3.2.1. Agent Component Ablation

Table 3 reports the ablation results for the four agents. Removing TPA decreases accuracy from 0.841 to 0.769 and weighted F1 from 0.844 to 0.774. Replacing ERA with keyword matching yields an accuracy of 0.773 and a weighted F1 of 0.768. Removing IA reduces weighted F1 to 0.792, while removing SRA reduces it to 0.804.

Table 3. Agent-level ablation results on ContractNLI.

Variant	Acc	F1[W]	F1[E]	F1[C]
w/o TPA	0.769	0.774	0.816	0.714
w/o ERA	0.773	0.768	0.808	0.705
w/o IA	0.787	0.792	0.835	0.738
w/o SRA	0.810	0.804	0.848	0.750
SelfAudit	**0.841**	**0.844**	**0.872**	**0.785**

Note: Bold values indicate the best performance in each column.

Among the four variants, replacing ERA with keyword matching produces the largest decrease in weighted F1, corresponding to 7.6 percentage points relative to the complete system. The decreases associated with TPA, IA, and SRA are 7.0, 5.2, and 4.0 percentage points, respectively.

These results indicate that retrieval and task planning have the largest effects on final decision quality among the four components under the current experimental setting. The smaller but consistent decrease caused by removing SRA suggests that reflection provides additional quality control after planning, retrieval, and inspection.

3.2.2. HCD-EA Component Ablation

Table 4 evaluates the three components of HCD-EA. The complete configuration uses node-specific three-dimensional confidence estimates from Qwen2.5-7B-Instruct and dependency-aware attribution. In the w/o Multi-Dim variant, each node receives a single scalar confidence score instead of a three-dimensional vector, reducing weighted F1 from

0.844 to 0.818. In the w/o Dep-Attr variant, low-confidence results trigger blind re-execution instead of selecting the starting node through dependency-aware attribution, yielding a weighted F1 of 0.822. In the w/o Hetero-V variant, GPT-4o replaces Qwen2.5-7B-Instruct for the same node-level scoring tasks, yielding a weighted F1 of 0.810.

Table 4. Ablation results for the three HCD-EA components on ContractNLI.

Variant	Acc	F1[W]	F1[E]	F1[C]
w/o Multi-Dim	0.813	0.818	0.856	0.762
w/o Dep-Attr	0.828	0.822	0.860	0.768
w/o Hetero-V	0.804	0.810	0.848	0.752
SelfAudit	**0.841**	**0.844**	**0.872**	**0.785**

Note: Bold values indicate the best performance in each column.

The corresponding decreases in weighted F1 are 2.6, 2.2, and 3.4 percentage points, respectively. Among these variants, the largest decrease is observed when the heterogeneous verifier is replaced. However, because the heterogeneous verifier uses Qwen2.5-7B-Instruct whereas the alternative uses GPT-4o, this comparison reflects differences in verifier diversity, model architecture, scale, and training. Therefore, the result should be interpreted as evidence for the effectiveness of the tested heterogeneous configuration rather than as an isolated estimate of the causal contribution of verifier heterogeneity.

3.2.3. Cross-Model Robustness

Table 5 compares SelfAudit with zero-shot prompting across four backbone models: Qwen2.5-72B, Llama-3.3-70B, DeepSeek-V3, and GPT-4o. SelfAudit improves weighted F1 across all four backbones. The average weighted F1 increases from 0.718 with zero-shot prompting to 0.831 with SelfAudit, corresponding to an average gain of 11.3 percentage points.

Table 5. Cross-model comparison between zero-shot prompting and SelfAudit. CV denotes the coefficient of variation across the four backbone models.

Backbone	Method	Acc	F1[W]
Qwen2.5-72B	ZS	0.773	0.770
	SelfAudit	0.825	0.831
Llama-3.3-70B	ZS	0.677	0.672
	SelfAudit	0.815	0.808
DeepSeek-V3	ZS	0.789	0.787
	SelfAudit	0.836	0.842
GPT-4o	ZS	0.612	0.644
	SelfAudit	0.841	0.844
Average, ZS		0.713	0.718
Average, SelfAudit		0.829	0.831
CV, ZS		11.7%	9.9%
CV, SelfAudit		1.4%	2.0%

The coefficient of variation (CV), computed as the standard deviation divided by the mean, decreases from 11.7% to 1.4% for accuracy and from 9.9% to 2.0% for weighted F1. The lower variation across backbones indicates that SelfAudit is less sensitive to the choice of backbone model under the tested settings.

3.3. Retrieval and Reflection Results

3.3.1. Retrieval Performance

Table 6 reports the retrieval results on LegalBench-RAG. SelfAudit's ERA achieves a Precision@1 of 26.18% and a Recall@1 of 30.84%, compared with 22.34% and 26.77% for PAKTON. These results correspond to relative improvements of 17.2% in Precision@1 and 15.2% in Recall@1, respectively.

Table 6. Retrieval performance on LegalBench-RAG. All retrieval scores are reported as percentages.

Method	Chunking	P@1	P@8	P@64	R@1	R@8	R@64
Standard RAG	Naive	2.40	4.33	1.29	3.37	34.51	76.39
Standard RAG	RCTS + Cohere	6.13	5.02	1.81	3.98	24.34	61.06
PAKTON	–	22.34	7.63	3.52	26.77	56.66	89.58
SelfAudit	–	**26.18**	**9.15**	**3.84**	**30.84**	**62.34**	**91.23**

Note: Bold values indicate the best performance in each column.

At $k = 8$, ERA achieves a precision of 9.15% and a recall of 62.34%. At $k = 64$, precision decreases to 3.84%, while recall increases to 91.23%. This pattern reflects the expected precision–recall trade-off as the retrieval depth increases: expanding the candidate set increases the likelihood of retrieving relevant evidence while also introducing more irrelevant items.

3.3.2. Confidence Decomposition

Table 7 summarizes the three confidence dimensions produced by HCD-EA. RFS has a mean of 0.82 with a standard deviation of 0.15, ESS has a mean of 0.78 with a standard deviation of 0.18, and RCS has a mean of 0.74 with a standard deviation of 0.20. Among the cases that trigger re-execution, RFS, ESS, and RCS are the lowest-confidence dimensions in 22.4%, 31.8%, and 45.8% of cases, respectively.

Table 7. Confidence decomposition produced by HCD-EA. Bottleneck denotes the proportion of re-execution cases in which the corresponding confidence dimension is the lowest among RFS, ESS, and RCS.

Dimension	Mean ± Std.	Bottleneck
RFS	0.82 ± 0.15	22.4%
ESS	0.78 ± 0.18	31.8%
RCS	0.74 ± 0.20	45.8%

RCS is therefore the most frequent bottleneck in the current experiments, although the three dimensions exhibit distinct distributions. This result suggests that audit uncertainty arises from multiple sources rather than being concentrated in a single quality dimension.

3.3.3. Error Attribution

We evaluate dependency-aware error attribution on 200 randomly sampled cases from the ContractNLI test set that triggered reflection under the predefined confidence criterion. The human reference diagnosis is based on the original audit task, relevant document text, and the outputs of TPA, ERA, IA, and the final-verdict stage. Node confidence scores, HCD-EA attribution results, and baseline selections are hidden during annotation. The reference node is assigned before comparison with either method, and the supporting document text and diagnostic rationale are recorded.

A TPA error is defined as incorrect task decomposition, omission of a required audit item, or assignment of an inappropriate rule. An ERA error is assigned when the task

is correctly specified but essential evidence is omitted, irrelevant material is retrieved, or evidence is incorrectly located. An IA error is assigned when the task and available evidence are sufficient but the reasoning or rule application is incorrect. A final-verdict error is assigned when the preceding stage results are correct but their aggregation or label mapping is incorrect. When errors occur at multiple stages, the reference identifies the earliest stage at which an actual error arises, rather than the node selected by a confidence-based rule.

Against these human reference labels, the dependency-aware method correctly identifies the faulty source node in 87.0% of cases (174/200). A baseline that selects the node with the lowest aggregate confidence achieves an accuracy of 42.5% (85/200), giving a difference of 44.5 percentage points.

In one annotated case, the task is to determine whether the recipient may disclose confidential information to legal advisers who are bound by confidentiality obligations. The document contains both a general prohibition on third-party disclosure and an explicit exception for such legal advisers. The recorded stage outputs are as follows: TPA identifies the general restriction, the legal-adviser exception, and the confidentiality condition; ERA retrieves only the general prohibition; IA consequently concludes that disclosure to legal advisers is prohibited; and the final-verdict stage outputs Contradiction. The human reference label is ERA because omission of the explicit exception is the first actual error. HCD-EA selects ERA, whereas the lowest-confidence baseline selects the final-verdict node.

This result indicates that the dependency structure provides useful information for locating the likely origin of an error. In contrast, selecting the lowest-confidence node does not necessarily identify the earliest source of failure because downstream stages may inherit uncertainty from an upstream error.

3.3.4. Evaluation of Heterogeneous Verification

We evaluate the verifier described in Section 2.6.2 by comparing the heterogeneous Qwen2.5-7B-Instruct verifier with a same-model GPT-4o verifier on 300 manually annotated verification decisions. The heterogeneous configuration achieves a Cohen's κ of 0.81 with human judgments, compared with 0.72 for the same-model configuration. The heterogeneous configuration increases total token consumption by approximately 4.2%.

The higher agreement observed for the heterogeneous configuration suggests that a verifier with different training and model characteristics can provide an additional source of judgment in the reflection stage. However, because the two verifiers differ in both model identity and capacity, the result does not isolate the effect of model heterogeneity from differences in model capability.

3.3.5. Parameter Sensitivity

Figure 4 shows the sensitivity of weighted F1 to the confidence threshold τ and the maximum number of reflection iterations K.

For the confidence threshold, F1[W] is 0.818 at $\tau = 0.55$, reaches 0.844 at $\tau = 0.75$, and decreases to 0.822 at $\tau = 0.95$. For the maximum number of iterations, F1[W] increases from 0.818 at $K = 1$ to 0.844 at $K = 3$, followed by values of 0.839 at both $K = 4$ and $K = 5$.

These results indicate that moderate reflection is preferable to either insufficient or excessive re-execution under the current setting. A threshold that is too low may allow uncertain outputs to pass without further verification, whereas a threshold that is too high may trigger additional iterations without corresponding performance gains. Similarly, performance peaks at $K = 3$ and remains slightly lower for larger iteration limits.

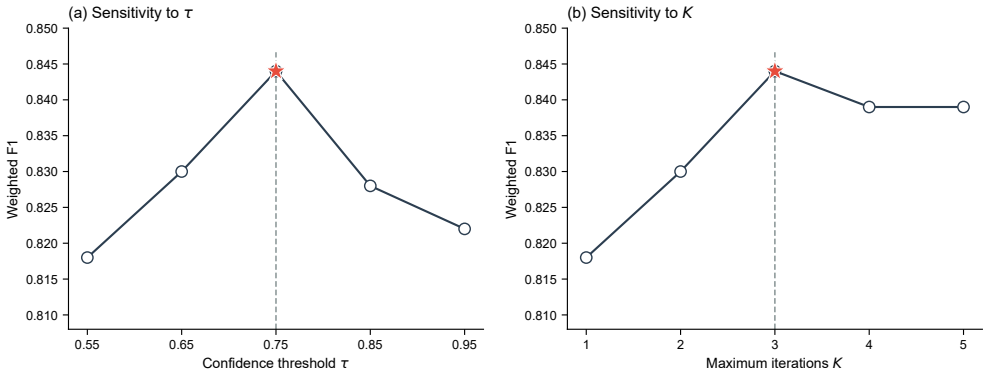

Figure 4. Sensitivity of weighted F1 to the confidence threshold τ and the maximum number of reflection iterations K. Red stars mark the selected best-performing settings, $\tau = 0.75$ and $K = 3$; vertical dashed lines indicate their parameter values. Stars do not denote statistical significance.

3.4. Computational Efficiency

Table 8 compares the computational cost of SelfAudit with representative baselines. All values are normalized to Standard RAG, which is assigned a relative cost of 1.0×. The complete SelfAudit framework requires 7.8× as many LLM inference calls as Standard RAG, with 7.1× input tokens and 6.4× output tokens. The version without HCD-EA, which uses blind retry, requires 9.4× calls, 8.6× input tokens, and 7.8× output tokens.

Compared with the blind-retry variant, the full HCD-EA configuration reduces the number of LLM calls from 9.4× to 7.8×, corresponding to a reduction of approximately 17.0%. This result indicates that diagnostic reflection can reduce part of the redundant computation introduced by iterative auditing. Nevertheless, SelfAudit remains substantially more expensive than conventional retrieval-based inference, indicating a clear performance–efficiency trade-off.

Table 8. Computational cost normalized to Standard RAG. Values represent relative LLM inference calls, input tokens, and output tokens per sample.

Method	LLM Calls	Input Tokens	Output Tokens
Standard RAG	1.0×	1.0×	1.0×
Self-RAG	2.3×	2.1×	1.8×
Reflexion	4.5×	4.0×	3.6×
Self-Refine	3.8×	3.4×	3.2×
CRITIC	5.2×	4.7×	4.3×
PAKTON	6.4×	5.8×	5.2×
SelfAudit w/o HCD-EA	9.4×	8.6×	7.8×
SelfAudit	7.8×	7.1×	6.4×

4. Discussion

4.1. Overall Findings

The results suggest that SelfAudit benefits from treating document auditing as a structured process rather than a single prediction task. Its performance across ContractNLI and CUAD indicates that the framework supports both document-level decision making and fine-grained information extraction, while the LegalBench-RAG results show that its retrieval component can provide relevant evidence for downstream auditing. The progressive improvement from prompting and reasoning-based methods to retrieval-augmented and multi-agent approaches further indicates that complex document auditing depends not only on the reasoning capability of the underlying LLM, but also on access to relevant

evidence and explicit process organization. SelfAudit extends this process with diagnostic reflection, which evaluates audit outputs and selectively revisits uncertain results rather than relying solely on repeated generation.

4.2. Role of the Audit Pipeline and HCD-EA

The ablation results indicate that the four agents play complementary roles in the auditing pipeline. Task planning and evidence retrieval have the largest effects under the current experimental setting, consistent with their roles in defining audit objectives and providing supporting evidence. The contribution of SRA is smaller but remains consistent, reflecting its role as a quality-control mechanism rather than a substitute for planning, retrieval, or inspection. The HCD-EA results further suggest that reflection benefits from distinguishing retrieval fidelity, evidence support, and rule compliance, as uncertainty can arise at different stages of the pipeline. Dependency-aware attribution provides additional information for locating the likely origin of an error, while heterogeneous verification provides an independent source of judgment during reflection. However, the heterogeneous verifier and same-model verifier differ in both model identity and capacity, so the observed improvement should not be interpreted as a controlled estimate of the causal effect of verifier diversity.

4.3. Performance–Efficiency Trade-Off

SelfAudit introduces additional inference cost because it performs task planning, inspection, confidence estimation, and selective re-execution. The results nevertheless show that HCD-EA can reduce part of the redundant computation associated with blind retry by directing additional execution toward uncertain stages. This design reflects a deliberate trade-off between audit reliability and computational cost: the framework prioritizes evidence-supported decisions rather than minimizing the number of LLM calls. Although this trade-off is reasonable for applications where audit reliability is more important than inference cost, the current implementation remains substantially more expensive than conventional RAG, motivating further optimization of confidence estimation and verification.

4.4. Limitations and Future Work

Several limitations remain. First, the heterogeneous verification experiment does not fully isolate the contribution of verifier diversity because the compared models differ in architecture and capacity. Future work could compare same-model and heterogeneous verification using verifiers with similar capacity and task capability, while keeping the input evidence, prompts, temperature, and inference budget fixed. Repeating this comparison across several model pairs would help distinguish the effect of verifier heterogeneity from the characteristics of a particular model pair. Second, the inference overhead of SelfAudit limits its direct use in cost- or latency-sensitive settings, motivating lightweight verification, caching, and early-exit strategies. Third, the current evaluation focuses mainly on legal and contract-oriented benchmarks. Broader experiments on financial, regulatory, procurement, and corporate compliance documents are needed to assess the generalization of the framework to other document-auditing settings. Finally, the diagnostic analyses rely partly on manually annotated subsets. Larger evaluations with multiple independent annotators would provide stronger evidence for the reliability of error attribution and verification.

5. Conclusions

This paper presented SelfAudit, a reflection-driven multi-agent framework for complex document auditing. The framework integrates task planning, evidence retrieval, inspection, and self-reflection into a structured auditing pipeline. HCD-EA decomposes audit confidence into retrieval fidelity, evidence support, and rule compliance, enabling diagnostic error attribution and selective re-execution. Experiments on ContractNLI, CUAD, and LegalBench-RAG demonstrate consistent improvements over representative baselines in document-level decision making, information extraction, and evidence retrieval. Ablation studies further confirm that task planning, structured retrieval, inspection, and diagnostic reflection each contribute to the overall performance of the framework.

SelfAudit also introduces additional inference overhead, and the current heterogeneous verification experiment does not fully disentangle the effect of verifier diversity from that of the verifier model itself. Future work will explore lightweight confidence estimation and early-exit strategies to reduce verification overhead, extend evaluation to a broader range of document domains, and retain validated reflection outcomes as reusable audit knowledge. These directions could further improve the scalability and adaptability of reflection-driven multi-agent approaches to document auditing.

Author Contributions: Conceptualization, C.Z. and F.C.; methodology, C.Z.; software, C.Z.; validation, C.Z., Y.C. and M.J.; formal analysis, C.Z.; investigation, C.Z.; resources, F.C. and Y.C.; data curation, C.Z.; writing—original draft preparation, C.Z.; writing—review and editing, C.Z., F.C., Y.C. and M.J.; visualization, C.Z.; supervision, F.C.; project administration, F.C.; funding acquisition, F.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The code and data that support the findings of this study are openly available in the SelfAudit GitHub repository at https://github.com/zcy2466/selfAudit (accessed on 22 October 2026).

Acknowledgments: During the preparation of this work the authors used ChatGPT (GPT6.1sol; OpenAI, San Francisco, CA, USA) in order to improve language. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication. The authors thank the anonymous reviewers for their valuable comments.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. CUAD Evaluation Interface, Span Records, and Confidence-Threshold Scoring

Appendix A.1. Evaluation Implementation and Evidence-Chain Export

This appendix describes the evaluation interface, span records, and confidence-threshold scoring procedure for the CUAD results in Table 2. Each contract–category pair is an evaluation sample. GPT-4o ZS, Standard RAG, Self-RAG, PAKTON, and SelfAudit use the same test samples and gold annotations.

The final SelfAudit graph output contains a `Verdict` object. The evaluation wrapper exports its complete `evidence_chain` through the field of the same name consumed by the span-extraction module. It also retains `final_label`, `confidence`, `confidence_3d`, and `iteration`, corresponding to the final label, sample-level confidence, three-dimensional scores, and iteration count. Table A1 summarizes the interface and scoring differences relative to public revision `81fb656`.

Table A1. Interface and scoring behavior of the CUAD evaluation implementation.

Component	Public Revision `81fb656`	Evaluation Implementation
Wrapper output	Returns the label, confidence, three-dimensional scores, and iteration count; the evidence-chain field is absent	Retains these fields and explicitly exports the final evidence chain
Span-extraction input	Requests an evidence-chain field absent from the wrapper output	Reads the complete evidence chain exported by the wrapper
Missing field and empty prediction	An empty-list default combines a missing field with an empty prediction	Treats a missing field as a record error and an explicit empty list as no candidate evidence
PR-curve construction	Combines precision–recall pairs from different examples	Sweeps confidence thresholds within each category and pools character-level TP, FP, and FN at each threshold

Algorithm A1 summarizes the export operation. It organizes the graph output described in Section 2.6 into evaluation records while retaining the Qwen verifier, node-level three-dimensional scores, dependency graph, and selective re-execution mechanism.

Algorithm A1 (Evidence-chain export and span preparation)

Input: final graph result R and original contract text D.
Output: evaluation result E.
1 Read the final Verdict object V from R.
2 Check the presence of `evidence_chain` and a confidence $c \in [0, 1]$.
3 Copy all entries of the evidence chain to E.
4 Export the final label, confidence, three-dimensional scores, and iteration count.
5 For each evidence entry, determine its original-text interval $[a, b)$, check $0 \leq a < b \leq \text{len}(D)$ and exact agreement with `D[a:b]`, and export `start`, `end`, `text`, and `score` $= c$.
6 For an explicitly empty evidence chain, export `predicted_spans` as an empty list.
7 Return E with its evidence chain and prepared spans.

Appendix A.2. Span Records and Confidence Assignment

Each record retains the contract identifier, category, original text, gold spans, and the confidences and candidate spans produced by each method. Table A2 defines the fields.

Table A2. Fields retained for a contract–category evaluation sample.

Field	Definition
`sample_id`	Unique identifier for a contract–category pair
`document_id`	Identifier for the original contract
`category`	Contract-review category evaluated for this sample
`document_text`	Original text used to interpret character offsets
`gold_spans`	Gold intervals, each with `start`, `end`, and exact `text`
`predictions`	Outputs indexed by the method names in Table 2
`confidence`	Saved extraction-result confidence for the method and sample
`predicted_spans`	Candidate intervals for the method and sample
`start`, `end`	Candidate start and end positions in the original text
`text`	Exact text covered by the interval
`score`	Candidate score, equal to the sample-level confidence

Offsets are zero-based Unicode character positions in the original contract and use half-open intervals [start, end). A span covers positions from `start` through `end`−1, has length `end`−`start`, and satisfies `document_text[start:end] == text`. Spaces, punctuation, and line breaks retain their original positions and count when covered by an interval.

For method m and sample i, let $c_i^{(m)}$ be the saved sample-level confidence. Every candidate span j in that method–sample record receives the same score:

$$s_{ij}^{(m)} = c_i^{(m)}. \tag{A1}$$

For SelfAudit, this value is the confidence returned by the final wrapper. Each method uses its own saved confidence; sharing applies to the candidate spans within one method and one sample.

The records cover all evaluated samples, including those without gold answers, those without candidates, and those with confidence below 0.50. Empty gold and candidate lists are recorded explicitly. Candidates are saved before applying the fixed output threshold so that all distinct confidences remain available for the PR sweep.

Appendix A.3. Original-Text Alignment and Character Counts

Recorded original-text offsets are checked against the contract used for evaluation. Text-only candidates are located by exact matching when their text occurs once. Repeated passages are resolved using their original locations.

Let G_i be the set of character positions covered by the union of the gold intervals for sample i. For method m at threshold t, the predicted coverage is

$$P_i^{(m)}(t) = \bigcup_{j:\, s_{ij}^{(m)} \geq t} \{a_{ij}, a_{ij}+1, \ldots, b_{ij}-1\}, \tag{A2}$$

where $[a_{ij}, b_{ij})$ is candidate j. Gold intervals are combined in the same way. Duplicate intervals and overlapping portions count once within a sample; positions in different samples are counted separately. The sample-level counts are

$$\begin{aligned} TP_i^{(m)}(t) &= |P_i^{(m)}(t) \cap G_i|, \\ FP_i^{(m)}(t) &= |P_i^{(m)}(t)| - TP_i^{(m)}(t), \\ FN_i^{(m)}(t) &= |G_i| - TP_i^{(m)}(t). \end{aligned} \tag{A3}$$

An empty prediction retains the missed gold characters as FN. A prediction for a sample without a gold answer contributes its covered characters as FP. Both cases enter the category and global totals.

Appendix A.4. Worked Example of Span Export and Threshold Scoring

Four constructed short contracts illustrate the record format and scoring arithmetic for one assignment-related category k^*. The example covers a complete match, partial coverage, a false-positive extraction, and a missed answer. Each original string below is a single line ending with a period.

C1 (75 characters). `The Supplier shall not assign this Agreement without prior written consent.`

C2 (71 characters). `The Customer may assign this Agreement to an Affiliate without consent.`

C3 (48 characters). `This Agreement shall continue for twelve months.`

C4 (56 characters). `Neither party may assign this Agreement without consent.`

The gold answers cover the complete sentences in C1, C2, and C4. C3 concerns the agreement term and has no gold answer for this example category. Table A3 gives all intervals and confidences.

Table A3. Complete inputs for the scoring example. Interval lengths are measured in characters.

Sample	Gold Intervals	Candidate Intervals	Confidence	Gold Length	Union Length
C1	$[0, 75)$	$[0, 75)$	0.90	75	75
C2	$[0, 71)$	$[0, 23), [17, 38)$	0.80	71	38
C3	$\emptyset$	$[0, 48)$	0.80	0	48
C4	$[0, 56)$	$\emptyset$	0.30	56	0

For C2, $[0, 23)$ contains `The Customer may assign`, and $[17, 38)$ contains `assign this Agreement`. Their lengths are 23 and 21, and their overlap $[17, 23)$ contains the six characters of `assign`. The union is $[0, 38)$, with length $23 + 21 - 6 = 38$. Both candidates have score 0.80. Record A1 shows the corresponding SelfAudit entry; the five methods use the fields in Table A2 for their respective outputs.

Record A1. Structured span-record example for C2.

```
{
  "sample_id": "Example-C2-kstar",
  "document_id": "Example-C2",
  "category": "assignment scoring example",
  "document_text":
    "The Customer may assign this Agreement to an Affiliate without consent.",
  "gold_spans": [
    {"start": 0, "end": 71, "text":
      "The Customer may assign this Agreement to an Affiliate without consent."}
  ],
  "predictions": {
    "SelfAudit": {
      "confidence": 0.80,
      "predicted_spans": [
        {"start": 0, "end": 23,
          "text": "The Customer may assign", "score": 0.80},
        {"start": 17, "end": 38,
          "text": "assign this Agreement", "score": 0.80}
      ]
    }
  }
}
```

The total gold coverage is $75 + 71 + 0 + 56 = 202$ characters. The distinct saved confidences are 0.90, 0.80, and 0.30. The sweep starts with no selected candidate, then visits these thresholds in descending order. At 0.80, C2 and C3 enter together. At 0.30, C4 contributes no predicted coverage because its candidate list is empty. The resulting counts and PR points appear in Table A4.

Table A4. Threshold-dependent character counts and PR points for the example.

Threshold	Selected Candidates	TP	FP	FN	Precision	Recall
>0.90	None (initial point)	0	0	202	1.000000	0.000000
0.90	C1	75	0	127	1.000000	0.371287
0.80	C1, C2, C3	113	48	89	0.701863	0.559406
0.30	C1, C2, C3	113	48	89	0.701863	0.559406

The fixed output threshold 0.50 lies between 0.80 and 0.30 and retains the candidates from C1, C2, and C3. Table A5 gives the per-sample counts.

Table A5. Per-sample counts at the fixed output threshold of 0.50.

Sample	TP	FP	FN	Count Interpretation
C1	75	0	0	Complete coverage of the 75-character gold answer
C2	38	0	33	Coverage of 38 of the 71 gold characters after merging overlaps
C3	0	48	0	All 48 predicted characters are false positives
C4	0	0	56	All 56 gold characters are missed
Total	113	48	89	Counts pooled across the four samples

The example micro-averaged metrics are therefore

$$\begin{aligned} \text{Precision}_{\text{ex}} &= \frac{113}{113+48} = \frac{113}{161} \approx 0.701863354, \\ \text{Recall}_{\text{ex}} &= \frac{113}{113+89} = \frac{113}{202} \approx 0.559405941, \\ \text{Micro-F1}_{\text{ex}} &= \frac{2\times 113}{2\times 113+48+89} = \frac{226}{363} \approx 0.622589532. \end{aligned} \tag{A4}$$

Trapezoidal integration of the points in Table A4 gives

$$\begin{aligned} \text{AUPR}_{k^*} &= \left(\frac{75}{202}-0\right)\frac{1+1}{2} \\ &\quad + \left(\frac{113}{202}-\frac{75}{202}\right)\frac{1+\frac{113}{161}}{2} \\ &\quad + \left(\frac{113}{202}-\frac{113}{202}\right)\frac{\frac{113}{161}+\frac{113}{161}}{2} \\ &\approx 0.371287129 + 0.160076256 + 0 \\ &\approx 0.531363385. \end{aligned} \tag{A5}$$

The repeated point at 0.30 adds zero area. Integration ends at the achieved maximum recall of 113/202. This example gives the area for one category; the CUAD summary averages the 41 category areas.

Appendix A.5. Complete Confidence-Threshold Scoring Procedure

For method m and category k, let I_k contain all evaluated samples in that category. At threshold t, the category counts are

$$\begin{aligned} TP_k^{(m)}(t) &= \sum_{i\in I_k} TP_i^{(m)}(t), \\ FP_k^{(m)}(t) &= \sum_{i\in I_k} FP_i^{(m)}(t), \\ FN_k^{(m)}(t) &= \sum_{i\in I_k} FN_i^{(m)}(t). \end{aligned} \tag{A6}$$

Precision and recall are calculated from these pooled counts:

$$\begin{aligned} p_k^{(m)}(t) &= \frac{TP_k^{(m)}(t)}{TP_k^{(m)}(t)+FP_k^{(m)}(t)}, \\ r_k^{(m)}(t) &= \frac{TP_k^{(m)}(t)}{TP_k^{(m)}(t)+FN_k^{(m)}(t)}. \end{aligned} \tag{A7}$$

Each method is evaluated separately within each category. All distinct saved confidences are scanned in descending order, and tied candidates enter together before the next PR point is computed. The initial point is $(r_0, p_0) = (0, 1)$. PR-curve precision is 1 while

no predicted character is selected. Empty-candidate samples retain their confidence as a threshold value and contribute zero predicted coverage.

For L threshold points, the category area is

$$\mathrm{AUPR}_k^{(m)} = \sum_{\ell=1}^{L} (r_\ell - r_{\ell-1}) \frac{p_\ell + p_{\ell-1}}{2}. \tag{A8}$$

Integration uses the original threshold points and ends at the achieved maximum recall. A category with zero gold-positive characters has area 0 and remains in the 41-category mean:

$$\mathrm{AUPR}^{(m)} = \frac{1}{41} \sum_{k=1}^{41} \mathrm{AUPR}_k^{(m)}. \tag{A9}$$

The precision, recall, and micro-F1 columns in Table 2 use the fixed output threshold $t_{\mathrm{out}} = 0.50$. Counts are first pooled across all samples and categories:

$$\begin{aligned} TP^{(m)} &= \sum_{k=1}^{41} TP_k^{(m)}(0.50), \\ FP^{(m)} &= \sum_{k=1}^{41} FP_k^{(m)}(0.50), \\ FN^{(m)} &= \sum_{k=1}^{41} FN_k^{(m)}(0.50). \end{aligned} \tag{A10}$$

The reported metrics are then

$$\begin{aligned} \mathrm{Precision}^{(m)} &= \frac{TP^{(m)}}{TP^{(m)} + FP^{(m)}}, \\ \mathrm{Recall}^{(m)} &= \frac{TP^{(m)}}{TP^{(m)} + FN^{(m)}}, \\ \text{Micro-F1}^{(m)} &= \frac{2TP^{(m)}}{2TP^{(m)} + FP^{(m)} + FN^{(m)}}. \end{aligned} \tag{A11}$$

A fixed-threshold metric is set to zero when its denominator is zero. The HCD-EA reflection threshold $\tau = 0.75$ controls selective re-execution, while the extraction-output threshold 0.50 selects candidate spans. Algorithm A2 summarizes the scoring process.

Algorithm A2 (Category AUPR and global micro metrics)

Input:	saved candidate spans and confidences for all methods and samples, gold spans, and 41 categories.
Output:	AUPR, micro-precision, micro-recall, and micro-F1 for each method.
1	For each method, process every category separately.
2	Build each sample's gold character union and sort all distinct saved confidences in descending order.
3	Initialize the curve at recall 0 and precision 1.
4	At each threshold t, retain candidates with score $\geq t$, include ties together, build each sample's predicted union, pool category counts, and append the resulting precision and recall.
5	Integrate the threshold-ordered points with trapezoids and average the areas of the 41 categories.
6	Retain candidates at threshold 0.50, pool counts over all samples and categories, and calculate the three micro-averaged metrics.

Scoring reads the saved spans and confidences directly and requires no additional model inference. Each PR point is derived from category-level counts at one confidence threshold.

Appendix A.6. Correspondence with Table 2

Table A6 lists the five methods and values reported in Table 2. AUPR is the mean of the 41 category areas; the other columns use global character counts at threshold 0.50.

Table A6. CUAD results and their scoring correspondence. AUPR averages 41 category areas; micro-F1, precision, and recall use the extraction-output threshold of 0.50.

Method	AUPR	Micro-F1	Precision	Recall
GPT-4o ZS	0.482	0.556	0.562	0.550
Standard RAG	0.524	0.598	0.610	0.587
Self-RAG	0.548	0.622	0.638	0.607
PAKTON	0.583	0.652	0.668	0.637
SelfAudit	**0.634**	**0.704**	**0.722**	**0.687**

Note: Bold values indicate the best performance in each column.

For each method, all four metrics refer to the same samples, gold annotations, candidate spans, and saved confidences. AUPR averages category areas, while the micro metrics use globally pooled counts. Calculations retain full precision, and reported values are rounded to three decimal places. Relative to PAKTON, the SelfAudit gains are $0.634 - 0.583 = 0.051$ in AUPR and $0.704 - 0.652 = 0.052$ in micro-F1, or 5.1 and 5.2 percentage points, respectively.

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.U.; Polosukhin, I. Attention is All you Need. In *Proceedings of the Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
2. Liu, N.F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; Liang, P. Lost in the middle: How language models use long contexts. *Trans. Assoc. Comput. Linguist.* **2024**, *12*, 157–173. [CrossRef]
3. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.V.; Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Proceedings of the Advances in Neural Information Processing Systems*; Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2022; Volume 35, pp. 24824–24837. [CrossRef]
4. Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv* **2022**, arXiv:2203.11171.
5. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. React: Synergizing reasoning and acting in language models. *arXiv* **2022**, arXiv:2210.03629.
6. Chalkidis, I.; Fergadiotis, M.; Malakasiotis, P.; Aletras, N.; Androutsopoulos, I. LEGAL-BERT: The Muppets straight out of Law School. In *Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020*; Cohn, T., He, Y., Liu, Y., Eds.; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 2898–2904. [CrossRef]
7. Cui, J.; Ning, M.; Li, Z.; Chen, B.; Yan, Y.; Li, H.; Ling, B.; Tian, Y.; Yuan, L. Chatlaw: A multi-agent collaborative legal assistant with knowledge graph enhanced mixture-of-experts large language model. *arXiv* **2023**, arXiv:2306.16092.
8. Colombo, P.; Pires, T.; Boudiaf, M.; Melo, R.; Culver, D.; Malaboeuf, E.; Hautreux, G.; Charpentier, J.; Desa, M. SaulLM-54B & SaulLM-141B: Scaling Up Domain Adaptation for the Legal Domain. In *Proceedings of the Advances in Neural Information Processing Systems*; Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2024; Volume 37, pp. 129672–129695. [CrossRef]
9. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.T.; Rocktäschel, T.; et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 9459–9474.
10. Li, Y.; Ke, W.; Liu, J.; Wang, P.; Liu, J.; He, Y. Towards evidence-aware retrieval-augmented generation via self-corrective chain-of-thought. *Inf. Process. Manag.* **2026**, *63*, 104369. [CrossRef]
11. Sarthi, P.; Abdullah, S.; Tuli, A.; Khanna, S.; Goldie, A.; Manning, C. Raptor: Recursive abstractive processing for tree-organized retrieval. In *Proceedings of the International Conference on Learning Representations*; ICLR: Rio de Janeiro, Brazil, 2024; Volume 2024, pp. 32628–32649.

12. Asai, A.; Wu, Z.; Wang, Y.; Sil, A.; Hajishirzi, H. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of the International Conference on Learning Representations*; ICLR: Rio de Janeiro, Brazil, 2024; Volume 2024, pp. 9112–9141.
13. Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 8634–8652. [CrossRef]
14. Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. Self-refine: Iterative refinement with self-feedback. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 46534–46594. [CrossRef]
15. Gou, Z.; Shao, Z.; Gong, Y.; Yang, Y.; Duan, N.; Chen, W. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *Proceedings of the International Conference on Learning Representations*; Kim, B., Yue, Y., Chaudhuri, S., Fragkiadaki, K., Khan, M., Sun, Y., Eds.; ICLR: Rio de Janeiro, Brazil, 2024; Volume 2024, pp. 57734–57811.
16. Trivedi, H.; Balasubramanian, N.; Khot, T.; Sabharwal, A. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Rogers, A., Boyd-Graber, J., Okazaki, N., Eds.; Association for Computational Linguistics: Toronto, OT, Canada, 2023; pp. 10014–10037. [CrossRef]
17. Dhuliawala, S.; Komeili, M.; Xu, J.; Raileanu, R.; Li, X.; Celikyilmaz, A.; Weston, J. Chain-of-verification reduces hallucination in large language models. In Proceedings of the Findings of the Association for Computational linguistics: ACL 2024, Bangkok, Thailand, 11–16 August 2024; pp. 3563–3578.
18. Huang, J.; Chen, X.; Mishra, S.; Zheng, H.S.; Yu, A.; Song, X.; Zhou, D. Large Language Models Cannot Self-Correct Reasoning Yet. In *Proceedings of the International Conference on Learning Representations*; Kim, B., Yue, Y., Chaudhuri, S., Fragkiadaki, K., Khan, M., Sun, Y., Eds.; ICLR: Rio de Janeiro, Brazil, 2024; Volume 2024, pp. 32808–32824.
19. Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv* **2023**, arXiv:2308.08155.
20. Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Wang, J.; Zhang, C.; Yau, S.; Lin, Z.; Zhou, L.; et al. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *Proceedings of the International Conference on Learning Representations*; Kim, B., Yue, Y., Chaudhuri, S., Fragkiadaki, K., Khan, M., Sun, Y., Eds.; ICLR: Rio de Janeiro, Brazil, 2024; Volume 2024, pp. 23247–23275.
21. Li, G.; Hammoud, H.; Itani, H.; Khizbullin, D.; Ghanem, B. Camel: Communicative agents for "mind" exploration of large language model society. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 51991–52008. [CrossRef]
22. Park, J.S.; O'Brien, J.; Cai, C.J.; Morris, M.R.; Liang, P.; Bernstein, M.S. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*; ACM: New York, NY, USA, 2023; pp. 1–22.
23. Raptopoulos, P.; Filandrianos, G.; Lymperaiou, M.; Stamou, G. PAKTON: A Multi-Agent Framework for Question Answering in Long Legal Agreements. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*; Christodoulopoulos, C., Chakraborty, T., Rose, C., Peng, V., Eds.; Association for Computational Linguistics: Suzhou, China, 2025; pp. 7948–7984. [CrossRef]
24. Jaech, A.; Kalai, A.; Lerer, A.; Richardson, A.; El-Kishky, A.; Low, A.; Helyar, A.; Madry, A.; Beutel, A.; Carney, A.; et al. Openai o1 system card. *arXiv* **2024**, arXiv:2412.16720.
25. Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Lu, K.; et al. Qwen2. 5-coder technical report. *arXiv* **2024**, arXiv:2409.12186.
26. Koreeda, Y.; Manning, C. ContractNLI: A Dataset for Document-level Natural Language Inference for Contracts. In *Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2021*; Moens, M.F., Huang, X., Specia, L., Yih, S.W.T., Eds.; Association for Computational Linguistics: Punta Cana, Dominican Republic, 2021; pp. 1907–1919. [CrossRef]
27. Hendrycks, D.; Burns, C.; Chen, A.; Ball, S. Cuad: An expert-annotated nlp dataset for legal contract review. *arXiv* **2021**, arXiv:2103.06268.
28. Pipitone, N.; Alami, G.H. Legalbench-rag: A benchmark for retrieval-augmented generation in the legal domain. *arXiv* **2024**, arXiv:2408.10343.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.