

Architecting Trust: The Post-Hoc Grounding Paradigm for Enterprise LLM Agents

1. Executive Summary

The enterprise integration of Large Language Models (LLMs) has transitioned from experimental chatbots to complex, agentic systems tasked with high-stakes knowledge work—compliance reporting, financial analysis, and technical auditing. In these domains, the reliability of information is paramount, yet a critical architectural friction remains: the "attribution bottleneck." Current workflows often force agents into a "cite-then-write" paradigm, requiring them to retrieve, tag, and embed citations synchronously while generating prose. This approach fights against the stochastic nature of token generation, leading to the well-documented "bookmark problem," where agents identify sources but fail to embed them, or hallucinate citations entirely due to cognitive overload.

This report establishes that the industry standard for high-reliability citation is shifting decisively toward **Post-Hoc Grounding** (P-Cite). In this paradigm, the agent creates content first—focusing on synthesis, logic, and narrative flow—and a specialized sub-system or toolchain retroactively identifies, verifies, and embeds citations. This architecture decouples the *creative* act of writing from the *analytical* act of auditing, mirroring the rigorous editorial processes found in human publishing.

Research conducted across 2024–2026, encompassing deep analyses of production engines like Perplexity AI, Google's AGREE framework, and the open-source PaperQA2 system, indicates that post-hoc architectures significantly outperform generation-time citation. Studies show P-Cite approaches achieving up to 99% source coverage compared to the 37–65% typical of generation-time methods.¹ Furthermore, the introduction of specialized verification models like MiniCheck has reduced the cost of automated fact-checking by orders of magnitude while matching GPT-4 performance levels.³

This document provides an exhaustive implementation guide for migrating a LangGraph-based agent system to a post-hoc grounding architecture. It details the granular mechanics of claim extraction using proposition-level segmentation, the design of "audit tools" that verify claims against retrieved evidence, and the integration of emerging reasoning models like DeepSeek R1 and OpenAI o1 to plan citation structures. By treating citations not as mere text markers but as verifiable claims linked to atomic propositions, organizations can build systems where every sentence is audit-proof.

2. The Crisis of Attribution in Agentic Workflows

The user's observed "disconnect"—where agents identify useful passages but fail to insert the corresponding [N] markers—is not a trivial bug but a fundamental symptom of "Generation-Time" (G-Cite) architectures. To solve it, we must understand the cognitive and architectural pressures placed on an LLM during the writing process.

2.1 The Cognitive Dissonance of "Cite-then-Write"

In a standard G-Cite workflow, the model is asked to perform two conflicting tasks simultaneously: **synthesis** and **indexing**. Synthesis requires the model to abstract information, combine multiple sources, and generate fluid natural language. Indexing requires the model to track rigid metadata (document IDs, page numbers) and insert them at precise syntactic boundaries.

From an information-theoretic perspective, the model's attention mechanism is overloaded. When an agent reads a document and calls `cite_document()`, it retrieves a token ID (e.g.,). As the agent continues to reason and generate text, this ID competes for attention with the semantic content of the answer. By the time the agent writes the relevant claim—often several paragraphs later—the attention weight of the specific ID may have decayed or been overwritten by the need to maintain narrative coherence. The result is the "bookmark effect": the agent "knows" the source exists (it created the bookmark) but fails to surface the pointer in the final output stream.

2.2 The Economic and Latency Costs of Verification Loops

The current system attempts to mitigate this with synchronous verification, where every citation is checked in real-time. This introduces severe latency penalties. If an agent writes a 10-paragraph report and cites 20 sources, a synchronous verification loop stops the generation 20 times to perform an NLI (Natural Language Inference) check. This not only destroys the user experience (high time-to-first-token) but also increases costs significantly.

Furthermore, when the agent fails to embed the citation, the system likely enters a "correction loop," asking the agent to "please add the missing citations." These loops are notoriously inefficient. The agent, now effectively tasked with editing its own work, often hallucinates *new* citations to satisfy the constraint or simply places the original ID in the wrong location to end the interaction. This "verification theater" consumes tokens without guaranteeing accuracy.

2.3 The Architectural Solution: Decoupling Synthesis from Grounding

The solution lies in acknowledging that **writing** and **citing** are distinct cognitive operations that should be handled by distinct architectural components.

- **The Writer (Agent):** Focuses on "Post-Hoc Grounding" (P-Cite). The writer produces a draft based on its internal context and retrieved documents, utilizing "placeholders" or

even raw claims without formatting constraints.

- **The Auditor (Tool/Sub-System):** A deterministic or specialized model that scans the draft, extracts claims, performs targeted retrieval (if necessary), verifies matches, and inserts the citations.

This separation of concerns allows the Writer to maximize fluency and synthesis quality, while the Auditor maximizes precision and recall. As detailed in the following sections, this is the architecture adopted by the world's most advanced "answer engines."

3. Anatomy of Production Citation Systems

To engineer a robust solution, we examine the architectures of leading systems that have solved the post-hoc citation problem at scale. The operational logic of Perplexity AI, Google's AGREE framework, and PaperQA2 provides a blueprint for enterprise agents.

3.1 Perplexity AI: The "Answer Engine" Architecture

Perplexity AI represents the industry gold standard for citation-backed generation. Unlike a standard search engine that provides links, or a chatbot that chats, Perplexity functions as an "answer engine" where the primary product is a synthesized, cited response. Its architecture is instructive for solving the specific disconnect identified in the user's system.⁵

3.1.1 The Retrieval-First Workflow

Perplexity does not rely on the LLM to "remember" document IDs during generation in the way a naive agent might. Instead, it employs a sophisticated multi-stage pipeline⁷:

1. **Query Decomposition:** The system breaks a user prompt into multiple sub-queries, executing dozens of searches in parallel to ensure comprehensive coverage.
2. **Snippet Extraction & Indexing:** Crucially, Perplexity does not ingest full documents blindly. It performs high-quality scraping and parsing to extract "snippets"—specific paragraphs or sentences.⁸ These snippets are indexed temporarily for the session, creating a transient, high-resolution knowledge base.
3. **Constraint-Based Generation:** The LLM is instructed via system prompts to be "citation-aware." It is likely subject to a negative constraint: "Do not say anything that you cannot ground in the retrieved snippets".⁸ This effectively creates a "boundary of knowledge," preventing the model from relying on its pre-trained (and potentially hallucinated) memories.
4. **Sentence-Level Attribution:** Citations are not merely appended; they are embedded at the sentence level. The architecture likely uses a post-processing step where generated sentences are aligned with the retrieved snippets using semantic similarity or token overlap. This alignment ensures that the citation `` is physically attached to the claim it

supports, rather than floating at the end of the paragraph.

3.1.2 Solving the "Bookmark" Problem

The user's problem—agents treating citations as bookmarks—is mitigated in Perplexity by **Granularity**. By breaking documents into atomic snippets *before* the LLM sees them, the system forces the model to engage with specific data points. When the LLM generates a claim, it is statistically more likely to attend to the specific snippet that justifies it, making the linkage (the citation) easier to reconstruct. The "bookmark" is not a pointer to a 50-page PDF, but to a 3-sentence snippet, reducing the ambiguity of the reference.⁶

3.2 Google's AGREE Framework: Tuning for Self-Grounding

While Perplexity relies on architectural constraints, Google's research into the **AGREE (Adaptation for GRounding EnhancEment)** framework¹ takes a model-centric approach, modifying the LLM itself to be a better citations engine.

3.2.1 The Mechanism of Self-Grounding

AGREE demonstrates that "bolting on" citations to a standard instruction-tuned model is often insufficient because standard models are trained to be helpful, not strictly factual. AGREE fine-tunes the LLM to "self-ground" claims, meaning the model learns to recognize when it is making a factual assertion that requires support.

- **Test-Time Adaptation (TTA):** A critical innovation in AGREE is the ability to perform test-time adaptation. If the model generates a claim that lacks support during the drafting phase, it can iteratively retrieve passages to support that specific claim *before* finalizing the output.⁹

For the user's LangGraph system, this suggests that simply prompting an agent to "cite sources" may fail if the underlying model hasn't been fine-tuned for attribution. A post-hoc tool that acts as the "AGREE" layer—verifying and retrieving support for ungrounded claims—is a necessary architectural addition to emulate this capability without training a custom model.

3.3 PaperQA2 & SciRAG: The "Recall-Summarize-Generate" Loop

For workflows involving dense technical or compliance documents, **PaperQA2**¹⁰ and **SciRAG**¹² offer a rigorous alternative to standard RAG.

3.3.1 Ranking and Contextual Summarization (RCS)

PaperQA2 introduces a paradigm shift in how evidence is processed. Instead of retrieving raw chunks and feeding them to the writer, it employs a **Ranking and Contextual Summarization (RCS)** step.

- 1. **Retrieve:** Fetch the top k chunks based on the query.
- 2. **Summarize:** A separate LLM call summarizes *each* chunk specifically in the context of the user's question. This summary includes the citation metadata explicitly.
- 3. **Generate:** The writer agent receives these *summaries* (which are rich in signal and low in noise) rather than raw text. The citations are baked into the summaries (e.g., "According to [Author, 2024], X is true..."), making it trivial for the writer to carry them over to the final report.

This "summarize-first" approach is particularly effective for compliance reports, where the raw legal text might be convoluted. The summary step acts as a translation layer, converting legalese into "citable facts" that the agent can easily handle.

3.4 Comparative Architecture: Generation-Time (G-Cite) vs. Post-Hoc (P-Cite)

The research landscape ¹ clearly delineates the trade-offs between these paradigms.

Feature	G-Cite (Generation-Time)	P-Cite (Post-Hoc)
Workflow	Agent selects `` while typing "The revenue was..."	Agent writes "The revenue was \$5M"; System finds source for "\$5M".
Cognitive Load	High. Model must track prose flow + source IDs simultaneously.	Low. Model focuses on synthesis; auditing is a separate step.
Precision	Variable (12–94%). Prone to "hallucinated citations" (citing real docs that don't support the claim).	High (26–75% baseline, up to 90% with advanced tools).
Recall	Low (37–65%). Agents often forget to cite known facts.	High (74–99%). Scanners rarely miss a claim that needs citing.
Latency	Low (Single pass).	Higher (Requires verification pass), but faster than "correction loops."

Best Use Case	Chatbots requiring instant response.	Compliance reports, Audit trails, Deep Research.
----------------------	--------------------------------------	---

Conclusion for User: For a compliance report workflow where accuracy is non-negotiable and latency is secondary, **P-Cite is the only viable architecture**. The latency cost of the post-hoc step is negligible compared to the value of auditability and the risk of hallucination.

4. The Mechanics of Post-Hoc Grounding

To implement P-Cite effectively, we must break down the process into discrete engineering steps. The transformation of a raw draft into a cited report happens in the **Grounding Pipeline**, which consists of Segmentation, Retrieval/Matching, and Verification.

4.1 Step 1: Claim Segmentation Strategies

The atomic unit of a citation is not a paragraph or a sentence, but a **Claim**. A single sentence may contain multiple claims requiring different sources (e.g., "Company X revenue grew by 5% , but net income fell ").

4.1.1 The Limitations of Sentence Splitting

Simple regex splitting by period (.) is insufficient for compliance reporting. Complex sentences often mix verifiable facts with reasoning, opinion, or conditional logic. Verifying the whole sentence fails if only part of it is supported.

- *Example:* "Although the board, which met in July, approved the merger, the regulatory filing was delayed."
 - This sentence contains three potential claims: the meeting date, the approval, and the delay.

4.1.2 Proposition-Level Segmentation

Research into **Proposition-Level Segmentation** (e.g., the PropSegmEnt corpus¹⁵) advocates for decomposing prose into "atomic claims" or propositions. An atomic claim is a minimal, declarative statement that can be independently verified.

- *Tooling:* Libraries for atomic claim extraction utilize lightweight, instruction-tuned LLMs (like flan-t5 or Llama-3-8B) to rewrite complex sentences into lists of simple statements.¹⁷
- *Prompt Pattern:* "Extract all verifiable factual claims from this text as a list of independent sentences. Ignore opinions and transitions."

Recommendation: Implement a dedicated segment_claims node in LangGraph. Do not rely

on heuristics. Use a fast, small LLM to "explode" the draft into a list of propositions. This list becomes the checklist for the Auditor.

4.2 Step 2: Matching Algorithms (Connecting Claims to Sources)

Once claims are isolated, the system must find the best supporting evidence. This is where **Dense X Retrieval** and **Propositional Retrieval** represent the state of the art for 2025–2026.

4.2.1 Dense X Retrieval: Matching Granularity

Standard RAG retrieves chunks of 200–500 tokens. **Dense X Retrieval**¹⁹ argues that the retrieval unit should match the granularity of the query (the claim).

- **The Mismatch:** A 10-word claim ("Revenue increased by 5%") is difficult to match against a 500-word paragraph using vector similarity because the paragraph's vector is dominated by noise (irrelevant context).
- **The Solution:** Index individual **propositions** (atomic facts) extracted from the source documents. When the agent generates a specific claim, the retriever matches it against a database of specific facts.
- **Implementation:** Use a library like LlamaIndex's DenseXRetrievalPack or propositional-retrieval in LangChain¹⁹ to re-index your source documents. This dramatically increases **Citation Precision** by ensuring the "shape" of the evidence matches the "shape" of the claim.

4.2.2 Hybrid Retrieval Strategies

For compliance reports, **Hybrid Retrieval** is non-negotiable.⁶

- **Dense Retrieval (Embeddings):** Essential for semantic matching (e.g., matching "Revenue increased" to "Sales went up"). Use models like bge-en-large or Contriever.
- **Sparse Retrieval (BM25):** Essential for keyword precision (e.g., matching specific regulatory codes like "Section 404(b)" or distinct entity names).
- **Strategy:** Perform both retrievals, fuse the results using Reciprocal Rank Fusion (RRF), and then pass the top candidates to the verification step.

4.3 Step 3: Verification (The Judge)

The most critical step in P-Cite is verifying that the selected source *actually* supports the claim. This step guards against "hallucinated citations," where the model cites a real document that is irrelevant to the claim.

4.3.1 NLI-Based Verification (MiniCheck)

MiniCheck³ is a breakthrough in efficient fact-checking.

- **Concept:** A specialized small model (e.g., MiniCheck-FT5, 770M parameters) trained specifically on the **LLM-AggreFact** dataset to output a binary Support / Not-Support

label for a (Claim, Document) pair.

- **Performance:** It achieves GPT-4 level accuracy on fact-checking benchmarks but is **400x cheaper** to run. This cost efficiency makes it feasible to check every single claim in a long report.
- **Implementation Logic:**

Python

```
# Conceptual implementation using MiniCheck
```

```
from minicheck import MiniCheck
```

```
scorer = MiniCheck(model_name='Bespoke-MiniCheck-7B', enable_prefix_caching=True)
```

```
# Check if the retrieved chunk supports the agent's claim
```

```
# enable_prefix_caching allows reusing the document encoding for multiple claims
```

```
pred_label, raw_prob, _, _ = scorer.score(docs=[retrieved_chunk], claims=[agent_claim])
```

```
if pred_label == 1:
```

```
    insert_citation(claim_id, chunk_id)
```

```
else:
```

```
    mark_as_unverified(claim_id)
```

This specialized model approach is far superior to asking a general-purpose LLM "Does this text support this claim?", which is prone to sycophancy (saying "yes" to please the user).³

4.3.2 AlignScore

Another robust metric is **AlignScore**.²⁴ Unlike the binary output of MiniCheck, AlignScore provides a continuous score representing the factual alignment between two texts. It is particularly useful for detecting subtle hallucinations where the agent slightly twists the meaning of a regulation (e.g., "The regulation *recommends* X" vs. "The regulation *requires* X"). A high threshold on AlignScore can filter out these nuanced errors.

4.4 Handling Multi-Source Synthesis

Agents often synthesize information from multiple sources (e.g., "While US regulations require X, EU regulations mandate Y"). This requires **Multi-Source Attribution**.²⁷

- **Attribution Logic:** The system must be capable of identifying *multiple* source IDs for a single sentence.
- **Implementation:** The verification step should not stop at the first match. If a claim is complex, the segmenter might split it into two propositions ("US requires X" and "EU requires Y"). The system finds a source for each. The final re-assembly step combines these into a single sentence with two citations: "While US regulations require X, EU regulations mandate Y."

5. Agentic Workflows & Tooling Patterns

Migrating from a "cite-then-write" to a "write-then-cite" workflow requires changing the agent's cognitive architecture. We leverage **LangGraph** to build a state machine that enforces this flow, ensuring the agent focuses on one task at a time.

5.1 The "Placeholder Resolution" Pattern

This is the most practical pattern for solving the user's specific problem. It decouples the *intent* to cite from the *action* of citing.

5.1.1 Workflow Description

1. **Drafting Node:** The agent writes text freely. When it relies on a specific fact, it can optionally insert a **Placeholder** (e.g., {cite: revenue_2023} or {check: "compliance requirement X"}).
 - *Implicit Variant:* The agent doesn't even use placeholders. It just writes prose. The segmentation step (Section 4.1) treats *every* sentence as a potential placeholder that needs resolution.
2. **Resolution Node:** A "Resolver Agent" (or a deterministic tool) scans the draft.
 - It parses the placeholders or segments the text.
 - It generates queries for the retrieval tool based on the content of the claim.
 - It calls the Retrieval Tool in parallel (batch processing) for efficiency.
3. **Grounding Node:** The MiniCheck model verifies the retrieved content against the claims.
4. **Formatting Node:** The system replaces the placeholders/claims with the official [N] citation format and appends the bibliography.

5.1.2 REWOO (Reasoning WithOut Observation)

The **REWOO** pattern ²⁹ is highly relevant here.

- **Concept:** The agent generates a full *plan* with placeholders (e.g., Step 1: Get revenue. Step 2: Get headcount. Step 3: Calculate ratio).
- **Execution:** The tools run in parallel to fill these variables.
- **Synthesis:** The final LLM call sees the filled variables and writes the report.
- **Relevance:** This prevents the "forgetting to cite" issue because the citations are structurally baked into the variables *before* the final prose is written. The agent cannot "forget" the citation because the variable *is* the citation.

5.2 Tool Interface Design

The user asked: *What tool interfaces work best?* We recommend shifting from a `cite_document` tool (which implies active agent decision) to an `audit` tool (which implies

passive agent submission).

5.2.1 The "Audit Tool" Signature

Instead of `cite_document()`, provide the agent with an **Audit Tool**.³⁰

- **Function:** `audit_draft(draft_text: str)`
- **Behavior:**
 1. The agent writes a paragraph and passes it to `audit_draft`.
 2. The tool (running logic, not just an LLM) segments the text, retrieves sources, and verifies claims.
 3. **Return Value:** The tool returns the *annotated text* (with `` inserted) AND a report of "Unverified Claims."
 4. **Agent Reaction:** The agent sees "Claim X was unverified" and either rewrites the sentence to be less specific or deletes it.

5.2.2 The "Claim-to-Source" Tool

- **Function:** `find_evidence(claim: str) -> (source_id, quote, confidence_score)`
- **Usage:** Used by the **Resolver Node** (not the writer). The writer creates the claim; the Resolver uses this tool to find the backing.

5.3 LangGraph Implementation Strategy

A **LangGraph** implementation allows us to create a cyclic graph for refinement.³²

Proposed Graph Structure:

1. **Writer Node:** Generates initial draft using retrieved context (or internal knowledge).
2. **Segmenter Node:** Breaks draft into atomic claims.
3. **Verifier Node:**
 - For each claim: Retrieve -> MiniCheck -> Assign Status (Verified/Hallucinated).
4. **Editor Node (Conditional):**
 - *If all verified:* Proceed to Formatting.
 - *If hallucinations found:* Pass back to Writer with specific feedback: "The claim 'X' could not be verified. Please rewrite or remove."

This **Generate -> Verify -> Refine** loop³ ensures 100% precision. The "Editor" acts as a guardrail, preventing the system from outputting uncited text.

5.4 Data Persistence & Audit Trails

For compliance reports, **Data Provenance** is key. The system must not only generate citations but store the reasoning behind them to prevent **OLIF** (Operator-Induced Longitudinal Integrity Failure).³⁰

- **PostgreSQL Schema Recommendation:**

SQL

```
CREATE TABLE citation_audit_trail (  
  citation_id UUID PRIMARY KEY,  
  report_id UUID,  
  claim_text TEXT NOT NULL,      -- The agent's generated text  
  source_id UUID,               -- Link to the original document  
  source_verbatim_text TEXT,     -- The specific snippet used for grounding  
  retrieval_score FLOAT,        -- Dense/Sparse retrieval score  
  verification_model VARCHAR(50), -- e.g., 'MiniCheck-FT5'  
  verification_confidence FLOAT, -- Probability score from MiniCheck  
  timestamp TIMESTAMP DEFAULT NOW()  
);
```

This creates an immutable audit trail. If a report is challenged, the organization can query this table to show exactly which text chunk the agent used to justify the claim, and how confident the system was in that link.

6. Agent Prompt Engineering for Citations

While architecture does the heavy lifting, the prompt is the interface. How do we ask the agent to collaborate with this post-hoc system?

6.1 "Citation-Aware" Prompting

We need to move beyond generic "please cite sources" instructions.

6.1.1 The "Academic" System Persona

Research on Perplexity and scientific agents suggests adopting an **Academic Persona** in the system prompt.⁸

- *Prompt Instruction*: "You are a rigorous compliance auditor. You must write in an academic style. Every factual statement is a hypothesis until verified. Write clearly and precisely to facilitate verification."
- *Effect*: This primes the model to avoid flowery, untestable language and focus on concrete propositions. It reduces the "temperature" of the generation without needing to manually adjust the hyperparameter.

6.1.2 Chain-of-Thought (CoT) with Citations

Use **CoT** to plan the citations *before* writing the sentence.³⁵

- *Pattern*: Thought: I need to state the revenue. I recall seeing \$5M in document A. -> Sentence: The revenue was \$5M.
- *Refinement*: Even better, use **Structured Output** (JSON).

```
JSON
{
  "thought": "Identifying revenue figures...",
  "claim": "The revenue was $5M",
  "potential_source_keywords": ["2023 financial report", "revenue"]
}
```

This allows the *Resolver Node* to use the "potential_source_keywords" for targeted retrieval.

6.2 Handling Reasoning Models (o1, DeepSeek R1)

The new generation of reasoning models (OpenAI o1, DeepSeek R1) introduces **Reasoning Traces**.³⁶

6.2.1 Planning with Reasoning Models

These models excel at **planning** the citation structure. They are less prone to "forgetting" citations because their "thinking" process allows them to iterate on the structure before committing to the output tokens.

- *Workflow*: Use DeepSeek R1/o1 as the **Planner Node**. Ask it to outline the report and identify *what evidence is needed* for each section.
- *Execution*: Use a cheaper, faster model (GPT-4o, Sonnet 3.5) to execute the writing based on the plan and retrieved evidence.
- *Benefit*: Reasoning models are "citation-aware" in their training.³⁹ They can deduce which claims are controversial and require stronger evidence (e.g., multiple sources) versus simple facts.

6.2.2 Citation-Aware Rubric Rewards (CaRR)

For advanced users training their own agents or using Reinforcement Learning (RL), the **CaRR** framework⁴⁰ is cutting-edge (Jan 2026).

- *Concept*: Instead of binary rewards (did it answer?), use fine-grained rubrics:
 1. **Comprehensiveness**: Did it cite all relevant docs?
 2. **Connectivity**: Do the citations logically support the reasoning chain?
- *Application*: Even without training, you can use these rubrics in your **Evaluator Node** (LLM-as-a-Judge) to score drafts before showing them to users.

7. Verification and Quality Assurance (QA)

How do we measure success? The metric cannot simply be "number of citations."

7.1 The ALCE Benchmark Standards

The **ALCE (Automatic LLMs' Citation Evaluation)** benchmark ⁴² defines the industry standard metrics for this domain. Implementing these metrics in the CI/CD pipeline is essential for maintaining system integrity.

Metric	Definition	Implementation Goal
Citation Precision	The percentage of citations that <i>actually support</i> the associated statement.	Target > 95%. Use MiniCheck to measure this continuously.
Citation Recall	The percentage of verifiable statements in the answer that <i>have</i> a citation.	Target > 90%. Use an Auditor agent to scan for uncited claims.
Citation F1	Harmonic mean of Precision and Recall.	The single "North Star" metric for your system.

7.2 Measuring Hallucination

To detect hallucinated citations (citations that look real but don't exist or don't support the claim), implement a **Reference-Blind Verification**:

1. Take the agent's claim.
2. Take the cited document text.
3. Ask an NLI model (MiniCheck): "Does the text entail the claim?"
4. *Crucially*: Do **not** provide the NLI model with the agent's *reasoning*. It must verify the link purely on evidence. If the NLI model says "No Entailment," the citation is flagged as a hallucination, regardless of how confident the agent was.

8. Emerging Frontiers (2025-2026 Outlook)

The field is moving fast. Several new tools and frameworks released in late 2025 and early 2026 can accelerate this migration.

8.1 PaperQA2 and SciRAG

- **PaperQA2** ¹⁰: A library achieving "superhuman" performance on scientific literature tasks. It implements the "Recall-Summarize-Generate" loop out of the box.
Recommendation: If your compliance documents are dense PDFs, consider swapping

your custom retrieval logic for PaperQA2's pipeline.

- **SciRAG**¹²: Adds **Citation Graph Expansion**. If Document A is cited, SciRAG also checks documents *cited by* A. This is powerful for legal/compliance workflows where regulations cross-reference each other (e.g., "As defined in Section 4..."). SciRAG follows the reference chain to ensure the citation is rooted in the primary source.

8.2 Collaborative Multi-Agent Systems (AISAC)

The **AISAC (AI Scientific Assistant Core)** system⁴⁴ demonstrates a **Router-Planner-Coordinator** pattern.

- **Relevance**: It separates the "Researcher" (who finds citations) from the "Writer" (who drafts).
- **LangGraph Pattern**: Use a "Research Subgraph" in LangGraph that runs independently. The Writer creates a "Research Request," the Subgraph runs for minutes finding valid sources, and returns a "Dossier" to the Writer. This asynchronous pattern mimics human research teams.

9. Implementation Roadmap: Migrating Your System

Based on the research, we propose a three-phase plan to migrate from the current "cite-then-write" system to a robust "post-hoc grounding" architecture.

Phase 1: The "Audit Tool" (Immediate Fix)

- **Objective**: Stop the bleeding. Allow agents to write naturally and fix citations in a second pass.
- **Action**: Create a new tool `audit_and_ground(text)`.
- **Logic**:
 1. Accepts a paragraph of text *without* citations.
 2. Segments it into sentences/claims (using a small LLM).
 3. Runs retrieval for each claim (using current `cite_document` logic but automated).
 4. Verifies matches using NLI (MiniCheck or simple LLM check).
 5. Returns the text with [N] inserted.
- **Agent Change**: Instruct the agent: "Write the report naturally. Do not worry about citation IDs. After writing a section, call `audit_and_ground` to verify and cite your work."

Phase 2: The "Placeholder" Architecture (Intermediate)

- **Objective**: Improve precision and handle complex claims.
- **Action**: Modify the prompt to use placeholders `{cite: specific_claim}`.
- **Logic**:
 1. Agent writes with explicit placeholders.
 2. A `ResolutionNode` in LangGraph parses these.

3. Resolves them in parallel (batch processing).
4. Re-generates the text if a placeholder fails to resolve (loops back to Writer).

Phase 3: Full Post-Hoc Grounding Pipeline (Advanced)

- **Objective:** Enterprise-grade reliability and auditability.
- **Action:** Implement **Dense X Retrieval** and **MiniCheck**.
- **Logic:**
 1. Re-index your PostgreSQL documents using propositional indexing (atomic facts).
 2. Replace the verify step with the MiniCheck-7B model for 400x cost reduction and higher accuracy.
 3. Implement **ALCE metrics** in your CI/CD pipeline to test the agent's citation performance on a gold-standard dataset of past reports.

Conclusion

The "citation-then-write" workflow is a vestige of early RAG attempts. The industry has proven that **Post-Hoc Grounding**—where writing and citing are decoupled and mediated by a rigorous verification layer—is the only path to reliable, professional agentic outputs. By adopting the **Segment-Retrieve-Verify-Inject** pipeline, supported by tools like **LangGraph** and **MiniCheck**, your system can achieve the high coverage and precision required for compliance reporting.

10. Appendix: Technical Reference & Tool Signatures

10.1 Recommended Tool Signatures (Python/Pydantic)

Python

```
class CitationAuditRequest(BaseModel):
    """Request to audit and ground a generated draft."""
    draft_text: str = Field(..., description="The prose text generated by the agent.")
    context_scope: List[str] = Field(default=, description="Optional list of doc IDs to restrict search to.")
```

```
class VerifiedCitation(BaseModel):
    claim_text: str
    source_id: str
    quote: str
    confidence: float
    is_hallucination: bool
```

```
class CitationAuditResponse(BaseModel):
    grounded_text: str = Field(..., description="Text with [N] citations inserted.")
    citations: List[VerifiedCitation]
    unverified_claims: List[str] = Field(..., description="Claims that could not be grounded.")
```

10.2 Recommended Libraries

- **Verification:** minicheck (GitHub: Liyan06/MiniCheck).⁴
- **Retrieval:** llama-index (DenseXRetrieval) or PaperQA2.¹⁰
- **Orchestration:** langgraph (for cyclic verification loops).³²
- **Evaluation:** ragas or ALCE evaluation scripts.⁴²

Works cited

1. Effective large language model adaptation for improved grounding - Google Research, accessed February 7, 2026, <https://research.google/blog/effective-large-language-model-adaptation-for-improved-grounding/>
2. arXiv:2311.09533v3 [cs.CL] 2 Apr 2024, accessed February 7, 2026, <https://arxiv.org/pdf/2311.09533>
3. MiniCheck: Efficient Fact-Checking of LLMs on Grounding Documents - ACL Anthology, accessed February 7, 2026, <https://aclanthology.org/2024.emnlp-main.499.pdf>
4. Liyan06/MiniCheck: MiniCheck: Efficient Fact-Checking of ... - GitHub, accessed February 7, 2026, <https://github.com/Liyan06/MiniCheck>
5. Perplexity AI PDF reading: retrieval-based parsing, citation grounding, and research workflows for early 2026 - Data Studios, accessed February 7, 2026, <https://www.datastudios.org/post/perplexity-ai-pdf-reading-retrieval-based-parsing-citation-grounding-and-research-workflows-for-e>
6. Architecting and Evaluating an AI-First Search API, accessed February 7, 2026, <https://research.perplexity.ai/articles/architecting-and-evaluating-an-ai-first-search-api>
7. Introducing Perplexity Deep Research, accessed February 7, 2026, <https://www.perplexity.ai/hub/blog/introducing-perplexity-deep-research>
8. How Does Perplexity Work? A Summary from an SEO's Perspective - Ethan Lazuk, accessed February 7, 2026, <https://ethanlazuk.com/blog/how-does-perplexity-work/>
9. Effective Large Language Model Adaptation for Improved Grounding and Citation Generation - ACL Anthology, accessed February 7, 2026, <https://aclanthology.org/2024.naacl-long.346.pdf>
10. Future-House/paper-qa: High accuracy RAG for answering ... - GitHub, accessed February 7, 2026, <https://github.com/Future-House/paper-qa>
11. PaperQA2: Superhuman scientific literature search - FutureHouse, accessed February 7, 2026,

- <https://www.futurehouse.org/research-announcements/wikicrow>
12. [2511.14362] SciRAG: Adaptive, Citation-Aware, and Outline-Guided Retrieval and Synthesis for Scientific Literature - arXiv, accessed February 7, 2026, <https://arxiv.org/abs/2511.14362>
 13. yale-nlp/SciRAG - GitHub, accessed February 7, 2026, <https://github.com/yale-nlp/SciRAG>
 14. Multi-Modal Hallucination Control by Visual Information Grounding - ResearchGate, accessed February 7, 2026, https://www.researchgate.net/publication/384207360_Multi-Modal_Hallucination_Control_by_Visual_Information_Grounding
 15. PropSegmEnt: A Large-Scale Corpus for Proposition-Level Segmentation and Entailment Recognition - Semantic Scholar, accessed February 7, 2026, <https://www.semanticscholar.org/paper/PropSegmEnt%3A-A-Large-Scale-Corpus-for-Segmentation-Chen-Buthpitiya/72da4a646a31d72bcae90b916e120cd7df5f9dae>
 16. Scalable and Domain-General Abstractive Proposition Segmentation - ACL Anthology, accessed February 7, 2026, <https://aclanthology.org/2024.findings-emnlp.517.pdf>
 17. ICAT: Evaluating Completeness of Factual Information in Long-form Text Generation - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2501.03545v1>
 18. Evaluating Coverage of Diverse Factual Information in Long-form Text Generation - ACL Anthology, accessed February 7, 2026, <https://aclanthology.org/2025.findings-acl.693.pdf>
 19. Enhancing Retrieval in QA Systems with Derived Feature Association - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2410.03754v1>
 20. MixGR: Enhancing Retriever Generalization for Scientific Domain through Complementary Granularity - ACL Anthology, accessed February 7, 2026, <https://aclanthology.org/2024.emnlp-main.579.pdf>
 21. Meta-Chunking: Learning Efficient Text Segmentation via Logical Perception | OpenReview, accessed February 7, 2026, <https://openreview.net/forum?id=gh563RwuIS>
 22. GitHub - edumunozsala/llamaindex-RAG-techniques, accessed February 7, 2026, <https://github.com/edumunozsala/llamaindex-RAG-techniques>
 23. Ybakman/TruthTorchLM - GitHub, accessed February 7, 2026, <https://github.com/Ybakman/TruthTorchLM>
 24. AlignScore: Evaluating Factual Consistency with a Unified Alignment Function, accessed February 7, 2026, https://www.researchgate.net/publication/371124017_AlignScore_Evaluating_Factual_Consistency_with_a_Unified_Alignment_Function
 25. Less is More for Improving Automatic Evaluation of Factual Consistency - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2404.06579v1>
 26. yuh-zha/AlignScore: ACL2023 - AlignScore, a metric for ... - GitHub, accessed February 7, 2026, <https://github.com/yuh-zha/AlignScore>
 27. On Synthesizing Data for Context Attribution in Question Answering - ACL

- Anthology, accessed February 7, 2026,
<https://aclanthology.org/2025.acl-long.828.pdf>
28. Towards Improved Multi-Source Attribution for Long-Form Answer Generation - ACL Anthology, accessed February 7, 2026,
<https://aclanthology.org/2024.naacl-long.216.pdf>
 29. REWOO Agent Pattern — Agent Patterns 0.2.0 documentation, accessed February 7, 2026,
<https://agent-patterns.readthedocs.io/en/stable/patterns/rewoo.html>
 30. Audit-Trail Fabrication in Tool-Using LLM Agents: Operator-Induced Longitudinal Integrity Failure (OLIF), Integrity Delta (Δ), and Zero-Trust Verifiable Logging for EU AI Act Record-Keeping - ResearchGate, accessed February 7, 2026,
https://www.researchgate.net/publication/400395675_Audit-Trail_Fabrication_in_Tool-Using_LLM_Agents_Operator-Induced_Longitudinal_Integrity_Failure_OLIF_Integrity_Delta_DI_and_Zero-Trust_Verifiable_Logging_for_EU_AI_Act_Record-Keeping
 31. 15 Best AI Search Monitoring Tools (Full Guide) - Analyze - AI, accessed February 7, 2026, <https://www.tryanalyze.ai/blog/aeo-tools-optimize-for-llms>
 32. Workflows and agents - Docs by LangChain, accessed February 7, 2026,
<https://docs.langchain.com/oss/python/langgraph/workflows-agents>
 33. LangGraph 101: Let's Build A Deep Research Agent | Towards Data Science, accessed February 7, 2026,
<https://towardsdatascience.com/langgraph-101-lets-build-a-deep-research-agent/>
 34. SciRAG: Adaptive, Citation-Aware, and Outline-Guided Retrieval and Synthesis for Scientific Literature - OpenReview, accessed February 7, 2026,
<https://openreview.net/pdf/2072faa6f43ceeb0dc13cfda66e6cf49e855a845.pdf>
 35. Cite: Contextual-Aware Citation Generation for Attributed Large Language Models - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2602.00004v1>
 36. GPT-4o vs o1-preview - LLM Stats, accessed February 7, 2026,
<https://llm-stats.com/models/compare/gpt-4o-2024-05-13-vs-o1-preview>
 37. DeepSeek Reasoning for MLPerf Inference v5.1 - MLCommons, accessed February 7, 2026, <https://mlcommons.org/2025/09/deepseek-inference-5-1/>
 38. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning - arXiv, accessed February 7, 2026, <https://arxiv.org/pdf/2501.12948>
 39. Daily Papers - Hugging Face, accessed February 7, 2026,
<https://huggingface.co/papers?q=GRPO-based%20RL%20rewards>
 40. Chaining the Evidence: Robust Reinforcement Learning for Deep Search Agents with Citation-Aware Rubric Rewards - ResearchGate, accessed February 7, 2026,
https://www.researchgate.net/publication/399666591_Chaining_the_Evidence_Robust_Reinforcement_Learning_for_Deep_Search_Agents_with_Citation-Aware_Rubric_Rewards
 41. [2601.06021] Chaining the Evidence: Robust Reinforcement Learning for Deep Search Agents with Citation-Aware Rubric Rewards - arXiv, accessed February 7, 2026, <https://arxiv.org/abs/2601.06021>
 42. princeton-nlp/ALCE: [EMNLP 2023] Enabling Large ... - GitHub, accessed February

- 7, 2026, <https://github.com/princeton-nlp/ALCE>
43. Ground Every Sentence: Improving Retrieval-Augmented LLMs with Interleaved Reference-Claim Generation - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2407.01796v2>
44. AISAC: An Integrated multi-agent System for Transparent, Retrieval-Grounded Scientific Assistance - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2511.14043v1>