PageIndex as RAG Alternative: A Comparative Evaluation of Vectorless Document Retrieval

Nathan King

January 24, 2026

Contents

Abstract

Retrieval-Augmented Generation (RAG) pipelines require practitioners to manage document chunking, embedding model selection, vector infrastructure, and retrieval parameter tuning—engineering overhead that compounds as corpus size and query complexity grow. PageIndex, developed by VectifyAI, proposes an alternative: replace vector embeddings with a hierarchical tree built from document structure, and replace similarity search with LLM-based reasoning to traverse that tree at query time. This paper presents a controlled empirical comparison between a traditional RAG pipeline (PyMuPDF4LLM parsing, Voyage 3 Large embeddings, FAISS retrieval, Claude Sonnet 4 generation) and the PageIndex Cloud API on a corpus of financial documents. Both pipelines answer identical question sets spanning five question types and three difficulty levels. Evaluation covers answer correctness (tiered exact, fuzzy, and semantic matching), retrieval precision and recall, hallucination rate, citation fidelity, per-query cost, and response latency. Results characterize the tradeoff space between the two approaches, identifying conditions under which vectorless retrieval matches, exceeds, or falls short of traditional RAG, and providing practical guidance for practitioners evaluating PageIndex for document intelligence workflows.

I. Introduction

Retrieval-Augmented Generation has become the default architecture for grounding large language models in private document collections. The standard pipeline is well-understood: parse documents into chunks, generate vector embeddings, store them in a vector index, retrieve the top-k most similar chunks at query time, and pass them as context to a generative model. This approach works, but it carries substantial engineering overhead. Practitioners must choose chunk sizes, manage embedding model selection and versioning, tune retrieval parameters, and maintain vector infrastructure—all before addressing the quality problems that arise when chunk boundaries split critical information or when embedding similarity fails to capture the reasoning needed to answer a question.

PageIndex, developed by VectifyAI, proposes a fundamentally different retrieval mechanism. Rather than converting document content into vectors and relying on geometric similarity, PageIndex constructs a hierarchical tree structure from the document and uses LLM reasoning to traverse that tree at query time. The system navigates from broad document sections down to specific pages based on the semantic relationship between the query and the document's organizational structure. There are no embeddings, no vector indices, and no top-k hyperparameters to tune.

The claim is compelling: eliminate the vector pipeline entirely while preserving or improving retrieval quality. A tree-based traversal strategy could plausibly outperform vector similarity on

questions requiring multi-step reasoning or synthesis across non-adjacent sections—cases where the "right" chunks may not be the most geometrically similar ones. Conversely, it could underperform on simple fact extraction where a direct embedding match would suffice. The cost and latency profile also differs: PageIndex replaces upfront embedding computation with per-query reasoning steps, shifting costs in ways that may favor one workload pattern over another.

This paper presents a controlled empirical comparison between a traditional RAG pipeline and PageIndex on a corpus of financial documents. Both pipelines answer the same set of questions spanning five question types and three difficulty levels. Evaluation is multi-dimensional: answer correctness (via tiered exact, fuzzy, and semantic matching), retrieval precision and recall, hallucination rate, citation fidelity, cost per query, and response latency. An optional blinded human evaluation provides qualitative validation of the automated scores.

The goal is not to declare a winner, but to characterize the tradeoff space. Under what conditions does vectorless retrieval match or exceed traditional RAG? Where does it fall short? And what does the cost–quality curve look like for each approach? The answers should help practitioners decide whether PageIndex merits integration into their document intelligence workflows, or whether the traditional pipeline remains the pragmatic choice.

II. Background

Traditional RAG Architecture

The standard RAG pipeline decomposes into three stages: document preparation, index construction, and retrieval-augmented generation. Each stage introduces design decisions that compound into a non-trivial engineering surface.

Document preparation begins with parsing source documents—typically PDFs—into machine-readable text. Tools like PyMuPDF4LLM extract content while preserving structural elements such as headers, tables, and section boundaries. The extracted text is then split into chunks, usually via recursive character splitting with a target size (commonly 500–1500 tokens) and an overlap window between adjacent chunks to preserve context across boundaries. Each chunk becomes an independent unit for downstream processing.

Index construction transforms chunks into dense vector representations using an embedding model. Models like Voyage 3 Large or OpenAI's text-embedding-3 map text into high-dimensional spaces (typically 1024–3072 dimensions) where semantic similarity corresponds to geometric proximity. These vectors are stored in an index—FAISS, Pinecone, Weaviate, or similar—that supports efficient nearest-neighbor search across the corpus.

Retrieval and generation operate at query time. The user's question is embedded using the same model, and the index returns the top-k most similar chunks. These chunks are concatenated into a context window and passed to a generative model alongside the question. The model produces an answer grounded in the retrieved context.

This architecture has known failure modes. Chunk boundary splits occur when the information needed to answer a question spans two or more chunks, but neither chunk alone contains enough context for the model to reason correctly. Semantic drift in embeddings arises when the geometric similarity between a query and the relevant passage is low despite their semantic relationship—common when the answer requires inference rather than lexical overlap. Top-k sensitivity manifests when the correct chunks fall just outside the retrieval window, or when irrelevant but superficially similar chunks dilute the context. These failure modes are well-documented but difficult to eliminate through parameter tuning alone, because they stem from the fundamental mismatch between geometric similarity and the reasoning required to answer complex questions.

PageIndex Architecture

PageIndex takes a structurally different approach to the retrieval problem. Rather than fragmenting documents into chunks and relying on vector similarity to locate relevant passages, it preserves the document's organizational hierarchy and uses LLM reasoning to navigate it.

Tree construction is the indexing phase. PageIndex analyzes the document's structure—section headings, page boundaries, content summaries—and builds a hierarchical tree that represents the document's organization. Each node in the tree contains a summary of the content within its scope, from broad top-level sections down to individual pages. This tree serves as a navigational map of the document rather than a bag of searchable fragments.

Reasoning-based traversal replaces vector similarity at query time. When a question arrives, the system starts at the root of the tree and uses an LLM to reason about which branches are most likely to contain relevant information. It descends through the hierarchy, making branching decisions at each level based on the relationship between the query's intent and each node's content summary. The traversal terminates at specific pages, which are then passed as context to the generation model. This approach treats retrieval as a reasoning task rather than a search task—the system decides where to look based on understanding what the question asks and how the document is organized.

The Cloud API exposes this functionality through a straightforward interface. Documents are submitted for indexing, which produces the tree structure at a cost of $0.01 per page. Queries are submitted against indexed documents and return generated answers along with token usage metadata. The Python SDK (`pageindex` package) wraps the API with a `PageIndexClient` class

providing methods for document submission, tree generation, and chat completions. Token usage is reported in chat completion responses, enabling direct cost tracking per query.

The architectural differences have direct implications for the failure modes described above. Tree traversal is inherently structure-aware, so it should not suffer from chunk boundary splits—the system retrieves whole pages in context, guided by document organization rather than arbitrary token windows. Reasoning-based navigation could potentially handle queries requiring multi-step inference better than vector similarity, since the traversal decision at each node is itself an act of reasoning about relevance. However, this reasoning adds latency: each traversal step involves an LLM call, making per-query costs dependent on tree depth rather than being a fixed embedding lookup. For simple fact-extraction queries where the relevant passage would score highly on embedding similarity, the additional reasoning steps may add cost and latency without improving retrieval quality.

Related Approaches

PageIndex is not the only system to challenge the vector-embedding orthodoxy. Several alternative and hybrid retrieval strategies have emerged, each addressing different failure modes of the standard pipeline.

Graph-based retrieval systems like GraphRAG construct knowledge graphs from documents and traverse entity relationships to locate relevant context. This approach excels at multi-hop reasoning but requires upfront entity extraction and relationship modeling, adding pipeline complexity comparable to or exceeding traditional RAG.

Late-interaction models such as ColBERT retain per-token embeddings rather than compressing entire passages into single vectors, enabling finer-grained matching. These preserve some structural information but still operate within the embedding paradigm and require specialized index infrastructure.

Structured retrieval approaches use document metadata, section headers, or table-of-contents information to pre-filter candidate passages before applying similarity search. These are lightweight hybrids that reduce the search space but do not eliminate the embedding step.

Long-context models sidestep retrieval entirely by fitting full documents into expanded context windows (100K+ tokens). This eliminates chunking and retrieval errors but scales poorly with corpus size and incurs high per-query token costs.

Keyword-based retrieval (BM25) remains a zero-cost baseline that outperforms embedding search on certain query types, particularly those with distinctive terminology. Hybrid approaches combining BM25 with dense retrieval often exceed either alone.

PageIndex occupies a distinct position in this space: it eliminates embeddings entirely, relies on document structure as its primary organizing principle, and delegates retrieval decisions to LLM reasoning. The closest analog is structured retrieval, but where structured retrieval uses metadata to narrow a vector search, PageIndex uses metadata—encoded as a tree—as the sole retrieval mechanism. Whether this architectural bet pays off in practice is the empirical question this evaluation aims to answer.

III. Methodology

Pipeline Definitions

The evaluation compares two end-to-end pipelines that share a generation model but differ in their retrieval mechanisms.

Traditional pipeline: PDF → PyMuPDF4LLM extraction → recursive character splitting (1000 chars / 200 overlap) → Voyage 3 Large embeddings (1024 dims) → FAISS IndexFlatL2 top-5 → Claude Sonnet 4 generation

PageIndex pipeline: PDF → PageIndex Cloud API (hierarchical tree generation) → reasoning-based tree traversal → Claude generation (server-side)

Traditional Pipeline

The traditional pipeline follows the standard RAG architecture described in Section II, instantiated with specific component choices.

Document parsing uses PyMuPDF4LLM to extract each PDF page as markdown text, preserving headers, tables, and structural formatting. Pages are extracted individually with character count metadata, maintaining page provenance through subsequent processing.

Chunking applies recursive character splitting with a hierarchical separator preference: paragraph boundaries (`\n\n`) first, then line breaks (`\n`), then word boundaries ( ), with a hard-split fallback at 1100 characters. The target chunk size is 1000 characters with 200 characters of overlap between consecutive chunks. Each chunk retains a `source_pages` list tracking which document pages contributed to its content. Character-based splitting (rather than token-based) preserves markdown structure and enables accurate page tracking without requiring a tokenizer.

Embedding and indexing transforms chunks into 1024-dimensional vectors using Voyage 3 Large, processing in batches of 128 for API efficiency. Vectors are stored in a FAISS `IndexFlatL2` index—exact Euclidean nearest-neighbor search, which is fast enough for the corpus scale (sub-millisecond per query) and avoids approximation errors that could confound the evaluation. Index files and

chunk metadata persist to disk for reuse across evaluation runs.

Query execution embeds the question using the same Voyage model, retrieves the top-5 most similar chunks, and formats them as context with page references (`[Pages N, M]\n{chunk_text}`). The context and question are passed to Claude Sonnet 4 (`claude-sonnet-4-20250514`) with temperature 0.0 and a system prompt emphasizing citation of page numbers in the response. Maximum generation length is 1024 tokens.

PageIndex Pipeline

The PageIndex pipeline delegates both indexing and retrieval to the PageIndex Cloud API, accessed via the Python SDK (`pageindex` package).

Document indexing submits each PDF to the API, which constructs a hierarchical tree representing the document's organizational structure. The client polls at 2-second intervals (up to a 30-second timeout) until the API confirms the tree is ready for retrieval. The resulting document identifier is cached locally to avoid re-indexing on subsequent runs.

Query execution proceeds in two phases. First, the retrieval phase submits the question and polls until the tree traversal completes, yielding a set of retrieved nodes with physical page indices. Second, the generation phase calls the chat completions endpoint, which produces an answer grounded in the traversed pages. Citations are extracted from both the structural retrieval response (page indices from `retrieved_nodes`) and from the answer text itself (regex pattern matching for page references), then merged and deduplicated. Token usage (`prompt_tokens` and `completion_tokens`) is reported in the API response for cost tracking.

Controlled Variables

The experimental design isolates the retrieval mechanism as the independent variable by holding other factors constant where possible.

Generation model. Both pipelines use Claude Sonnet 4 for answer generation. The traditional pipeline calls the Anthropic API directly; the PageIndex pipeline uses Claude server-side through the PageIndex API. While the PageIndex API's internal model configuration is not fully transparent, VectifyAI's documentation indicates Claude is the default generation backend.

Temperature. Set to 0.0 for both pipelines to produce deterministic outputs, enabling meaningful comparison across runs.

Question set. Both pipelines answer identical questions from the same question set, evaluated against the same ground truth answers and scoring thresholds.

Prompt structure. The traditional pipeline's system prompt instructs the model to answer based on provided context and cite page numbers. The PageIndex pipeline's prompt behavior is controlled server-side, but the user message (the question) is identical across both pipelines.

Test Corpus

The test corpus consists of financial documents representative of commercial lending workflows, selected to span document types, lengths, and formatting characteristics.

Document types include five categories drawn from commercial credit analysis:

- **Credit memoranda** (10 documents, 15–60 pages): approval memos, renewal analyses, and modification requests containing narrative assessments, financial summaries, and risk ratings
- **Audited financial statements** (10 documents, 30–150 pages): complete audit packages with balance sheets, income statements, cash flow statements, and footnote disclosures
- **Covenant compliance certificates** (10 documents, 5–20 pages): periodic borrower certifications with calculated financial ratios and covenant threshold comparisons
- **Appraisal reports** (5 documents, 50–200 pages): real estate valuations with comparable sales analyses, income approach calculations, and property descriptions
- **Loan agreements** (5 documents, 80–300 pages): credit facilities with defined terms, conditions precedent, financial covenants, and default provisions

Formatting diversity spans three complexity levels:

- **Clean digital**: PDFs with embedded text layers, requiring no OCR—typically system-generated financial statements or word-processed memos
- **Scanned**: Document images requiring optical character recognition, introducing potential extraction noise—commonly older agreements or signed certificates
- **Mixed**: Documents combining digital and scanned pages—such as a digitally-prepared memo with scanned signature pages or attached hand-annotated exhibits

The target corpus size is approximately 40 documents totaling ~3,000 pages, yielding roughly 6,000 chunks under the traditional pipeline's chunking parameters.

Question Set Design

The question set targets five question types across three difficulty levels, designed to stress different retrieval and reasoning capabilities.

Question types:

1. **Single-fact extraction**: Questions answerable from a single location in the document, re-

quiring direct lookup rather than inference. Example: "What year was the Ellis House built?"

2. **Multi-location synthesis**: Questions requiring information from multiple document sections, testing whether retrieval surfaces all relevant passages. Example: "How does the debt service coverage ratio compare to the covenant requirement?"
3. **Tabular reasoning**: Questions requiring interpretation of data presented in tables, testing whether parsing preserves tabular structure and whether retrieval locates the correct table.
4. **Negative/absence**: Questions whose correct answer is that information is not present in the document, testing whether the system avoids hallucinating unsupported claims. Example: "Is family information mentioned for Dr. Hawthorne?"
5. **Conditional logic**: Questions involving hypothetical scenarios or conditional reasoning over document content. Example: "If William bid $70, who wins the appetizer?"

Difficulty levels reflect the complexity of reasoning required:

- **Level 1** (easy): Direct extraction from a single clearly-identified location
- **Level 2** (medium): Synthesis across multiple locations or moderate inference
- **Level 3** (hard): Conditional reasoning, counterfactual scenarios, or complex multi-step interpretation

Ground truth specification for each question includes:

- A canonical correct answer (normalized, lowercased)
- A list of acceptable variants capturing alternative phrasings (e.g., "$25M", "25 million", "$25,000,000")
- Source page numbers (1-indexed) identifying all pages containing relevant information
- Source section names for structural reference
- A reasoning annotation describing the inference steps required
- Optional notes on ambiguity or edge cases

The target question set contains at least 100 questions distributed across types and difficulty levels, with a minimum of 15 questions per type and at least 20 questions at each difficulty level. Two human evaluators score each question independently to enable inter-rater agreement measurement.

IV. Scoring Framework

Evaluation uses a multi-dimensional scoring framework that combines automated metrics with optional human judgment. Each pipeline's response to each question produces scores across four automated dimensions—answer correctness, retrieval quality, hallucination rate, and citation fidelity—plus optional blinded human scores.

Answer Correctness

Answer correctness uses a tiered cascade that applies progressively looser matching criteria, stopping at the first tier that produces a match. This design balances precision (exact matches are unambiguous) with recall (semantically correct answers phrased differently from the ground truth should still receive credit).

Tier 1: Exact match. The generated answer and ground truth are both normalized (lowercased, stripped of leading/trailing whitespace) and compared directly. If they match, the answer scores `correct` with confidence 1.0. If the ground truth specifies acceptable variants, each variant is checked with the same normalization. A variant match also scores `correct` with confidence 1.0. This tier handles cases where the answer is precisely right but may use a known alternative phrasing—"$25M" matching a ground truth of "25 million dollars", for instance, when both are listed as acceptable variants.

Tier 2: Fuzzy match. If exact matching fails, the answer is scored using rapidfuzz's `token_set_ratio`, which tokenizes both strings, computes the ratio of shared tokens regardless of order, and returns a score between 0 and 100 (normalized to 0.0–1.0). This handles token reordering, minor phrasing differences, and partial overlaps. Thresholds:

- Score >= 0.90: `correct` (confidence = fuzzy score)
- Score >= 0.60: `partial` (confidence = fuzzy score)
- Score < 0.60: proceed to Tier 3

Token set ratio is particularly effective for financial answers where the same information may be expressed with different formatting ("$25,000,000" vs. "25 million" vs. "$25M") after normalization brings them into comparable token sets.

Tier 3: Semantic similarity. If fuzzy matching does not reach the correct threshold, both the generated answer and ground truth are embedded using Voyage 3 Large (the same model used for document indexing) and compared via cosine similarity. This captures semantic equivalence that surface-level token matching misses—for example, "the loan matures in 2027" matching a ground truth of "maturity date is December 31, 2027." Thresholds:

- Cosine similarity >= 0.85: `correct` (confidence = similarity score)
- Cosine similarity >= 0.65: `partial` (confidence = similarity score)
- Similarity < 0.65: `incorrect` (confidence = max of fuzzy score and 0.0)

The cascade terminates at the first tier that produces a `correct` or `partial` result. If all three tiers fall below their thresholds, the answer is scored `incorrect`. Each result records both the final score (`correct`, `partial`, `incorrect`) and the method that produced it (`exact`, `variant_exact`, `fuzzy`, `semantic`, `below_threshold`), enabling post-hoc analysis of which scoring tiers are doing

the most work.

Retrieval Quality

Retrieval quality measures whether the pipeline cited the correct source pages, independent of answer correctness. Both pipelines are instructed to cite page numbers, and the evaluation compares cited pages against the ground truth's `source_pages` list.

Precision measures the fraction of cited pages that are actually relevant:

$$\text{Precision} = \frac{|\text{cited\_pages} \cap \text{source\_pages}|}{|\text{cited\_pages}|}$$

A precision of 1.0 means every cited page contains relevant information; lower precision indicates the pipeline cited irrelevant pages, potentially diluting context or misleading the reader.

Recall measures the fraction of relevant pages that were cited:

$$\text{Recall} = \frac{|\text{cited\_pages} \cap \text{source\_pages}|}{|\text{source\_pages}|}$$

A recall of 1.0 means the pipeline found all relevant pages; lower recall indicates missed sources. For questions where `source_pages` is empty (edge cases in the question set), recall defaults to 1.0.

These metrics specifically evaluate retrieval rather than generation: a pipeline might generate a correct answer from incomplete retrieval (lucky inference) or an incorrect answer despite perfect retrieval (generation failure). Separating retrieval quality from answer correctness helps diagnose whether errors originate in the retrieval or generation stage.

Hallucination Rate

Hallucination rate measures the fraction of claims in the generated answer that are not supported by the retrieved context. This metric uses an LLM-as-judge approach: Claude is prompted to decompose the generated answer into individual claims, then evaluate each claim against the context that was available to the generation model.

For each claim, the judge determines whether it is supported by the retrieved passages. The hallucination rate is:

$$\text{Hallucination Rate} = \frac{|\text{unsupported claims}|}{|\text{total claims}|}$$

A hallucination rate of 0.0 means every claim in the answer is grounded in retrieved context. Higher rates indicate the model is generating information beyond what the retrieval step provided—a particularly concerning failure mode for financial document analysis where unsupported claims could lead to incorrect business decisions.

The individual unsupported claims are recorded alongside the aggregate rate, enabling qualitative analysis of hallucination patterns (e.g., whether hallucinations tend to involve numerical values, dates, or entity names).

Citation Fidelity

Citation fidelity measures the fraction of claims that are traceable to a specific cited page, distinct from hallucination rate in that it evaluates attribution rather than factual support. A claim might be supported by the retrieved context but not explicitly attributed to a page number in the response.

$$\text{Citation Fidelity} = \frac{|\text{claims with page citations}|}{|\text{total claims}|}$$

High citation fidelity indicates the pipeline produces verifiable answers—a reader can check each claim against the cited source page. Low citation fidelity, even with low hallucination rate, suggests the answer is likely correct but difficult to verify, reducing its utility in workflows that require audit trails.

Human Evaluation

An optional blinded human evaluation provides qualitative validation of the automated scores. Two evaluators independently score each answer on three dimensions using a 0–1–2 scale:

Correctness (0/1/2):

- 0: The answer is incorrect or substantially wrong
- 1: The answer is partially correct—contains the right information but is incomplete or includes minor errors
- 2: The answer is fully correct and complete

Completeness (0/1/2):

- 0: The answer addresses little or none of the question's scope
- 1: The answer addresses the question partially, missing significant relevant information
- 2: The answer fully addresses all aspects of the question

Citation quality (0/1/2):

- 0: No citations provided, or citations are incorrect
- 1: Some citations provided but incomplete or partially incorrect
- 2: All claims are accurately cited with correct page references

Evaluators see the question, the generated answer, and the source document, but not the pipeline that produced the answer (blinding). Each question receives scores from two evaluators, enabling inter-rater agreement calculation (Cohen's kappa) as a check on scoring consistency. Disagreements exceeding one point on any dimension are flagged for adjudication.

V. Experimental Setup

Implementation Stack

The evaluation harness is implemented in Python 3.11+ as a CLI application using Typer with Rich for formatted terminal output. The project uses a src-layout (`src/vectify/`) with a `pyproject.toml` build configuration (hatchling backend), and manages dependencies including `pymupdf4llm`, `voyageai`, `anthropic`, `pageindex`, `faiss-cpu`, `rapidfuzz`, and `pydantic` (v2).

The CLI exposes four subcommands:

- `vectify index`: Indexes all corpus documents through both pipelines, recording indexing metrics
- `vectify evaluate`: Runs the question set against both pipelines, applying the full scoring framework
- `vectify report`: Aggregates results and generates comparison tables by pipeline, question type, and difficulty level
- `vectify human-eval`: Presents blinded answers for human scoring

Configuration is loaded from TOML files (`configs/default.toml`) via `tomllib`, with API keys sourced from environment variables. All pipeline parameters—chunk size, overlap, top-k, embedding dimensions, scoring thresholds—are captured in a single configuration object and snapshotted with each evaluation run for reproducibility.

Cost Tracking

Cost is tracked per API call for both pipelines, using token counts and page counts reported in API responses rather than estimates.

PageIndex costs have two components:

- *Indexing*: $0.01 per page, calculated from the document's page count at submission time

- *Querying*: $0.03 per query as a base cost, plus token-level costs derived from the `usage` field in chat completion responses (`prompt_tokens` and `completion_tokens`)

Traditional pipeline costs have two components:

- *Embedding*: Voyage AI charges per token embedded. The SDK returns `total_tokens` in each embedding response, priced at $0.06 per million tokens. Embedding costs are incurred both at indexing time (chunked document text) and at query time (question embedding)
- *Generation*: Anthropic charges per input and output token. The SDK returns `usage.input_tokens` and `usage.output_tokens` in each response, priced at $3.00 per million input tokens and $15.00 per million output tokens for Claude Sonnet 4

All costs are recorded per-call and aggregated at two levels: per-document (indexing costs) and per-question (query costs). The final report presents both the total cost of each pipeline across the full evaluation and the marginal cost per query, enabling practitioners to compare the two approaches under different usage patterns—high indexing volume with few queries versus small corpora with high query throughput.

Latency Measurement

Latency is measured as wall-clock time using Python's `time.perf_counter()`, which provides sub-millisecond precision independent of system clock adjustments. Each API call is individually timed, capturing the full round-trip from request initiation to response completion.

For the **traditional pipeline**, latency includes:

- *Indexing*: Voyage embedding API calls (batched, so total time reflects batch count $\times$ per-batch latency)
- *Query*: Voyage question embedding (single call) + FAISS search (sub-millisecond, negligible) + Anthropic generation call

For the **PageIndex pipeline**, latency includes:

- *Indexing*: Document submission + polling wait until tree generation completes (includes server-side processing time)
- *Query*: Query submission + retrieval polling (tree traversal completes server-side) + chat completions call

Wall-clock time is the user-experienced metric and is directly comparable across pipelines despite their different internal architectures. Server-side processing time is not separately observable for either pipeline's cloud API calls, but wall-clock measurement captures the total latency a production system would experience. All timing values are recorded in milliseconds per question, with aggregate

statistics (mean, median, p95) computed at report time.

Resumability

The evaluation harness supports interruption and resumption without data loss or duplicate work. Results are persisted in append-mode JSONL files (one JSON record per line, flushed after each write), ensuring that partial runs produce valid, parseable output files even if the process is terminated mid-execution.

On restart, the harness reads the existing results file, extracts the set of completed `(question_id, pipeline)` pairs, and skips those combinations during the current run. This deduplication-on-restart approach means that:

- A network timeout or API error on question 47 does not require re-running questions 1–46
- Evaluation can be paused and resumed across sessions without configuration changes
- Partial results are immediately available for interim analysis while the full run continues

Each evaluation run writes to a dedicated directory (`data/results/{run_id}/`) containing the run configuration snapshot, indexing metrics, answer results, and human scores as separate JSONL files. The run ID is timestamp-based (`YYYYMMDD_HHMMSS`), providing natural chronological ordering across runs.

VI. Results

Results are reported from the primary evaluation run (36 questions across 2 difficulty levels and 3 question types), with comparisons to an initial 16-question pilot run where relevant. Of the 100 questions in the full question set, 36 were evaluable based on successfully indexed documents. The remaining questions targeted documents that were not indexed in either or both pipelines at the time of evaluation.

Answer Correctness

The traditional pipeline substantially outperforms PageIndex on answer correctness.

Pipeline	Correct	Partial	Incorrect
Traditional	47.2% (17/36)	50.0% (18/36)	2.8% (1/36)
PageIndex	22.2% (8/36)	52.8% (19/36)	25.0% (9/36)

The traditional pipeline produces a correct answer nearly half the time and an incorrect answer only once across 36 questions. Its failure mode is imprecision rather than fabrication: when it does

not match ground truth exactly, it typically lands in the partial range (fuzzy or semantic similarity above the partial threshold but below the correct threshold). PageIndex shows a qualitatively different error profile—a quarter of its answers are scored incorrect, indicating not just imprecision but substantive divergence from the expected answer.

The high partial rate for both pipelines (50–53%) warrants interpretation. Many partial scores reflect answers that are semantically in the right neighborhood—containing relevant information from the correct document section—but phrased differently enough from the ground truth to fall below the 0.90 fuzzy threshold. Whether these represent genuine quality differences or scoring artifacts is addressed in Section VII.

By Question Type

Type	PageIndex Correct	Traditional Correct	n
Single-fact extraction	26% (7/27)	56% (15/27)	27
Multi-location synthesis	14% (1/7)	14% (1/7)	7
Negative/absence	0% (0/2)	50% (1/2)	2

The traditional pipeline's advantage concentrates in single-fact extraction, where its top-k chunk retrieval reliably surfaces the passage containing the target fact. For multi-location synthesis—questions requiring information from multiple document sections—both pipelines perform equally poorly at 14% correct. This suggests that neither vector similarity nor tree traversal, as currently configured, reliably identifies and combines information scattered across non-adjacent sections.

The negative/absence category has too few questions (n=2) for meaningful comparison, though the initial 16-question pilot showed both pipelines handling this type well (100% correct on 2 questions each). Conditional logic and tabular reasoning questions were not represented in the 36-question run.

By Difficulty Level

Level	PageIndex Correct	Traditional Correct	n
Level 1 (easy)	29% (6/21)	52% (11/21)	21
Level 2 (medium)	13% (2/15)	40% (6/15)	15

The traditional pipeline's advantage holds across both difficulty levels with a consistent gap of 23–27 percentage points. Neither pipeline was tested on Level 3 (hard) questions in the primary

run, though the pilot showed both struggling at that tier (PageIndex 0%, Traditional 33% on 3 questions).

Retrieval Precision and Recall

Pipeline	Avg Precision	Avg Recall
Traditional	0.205	0.799
PageIndex	0.244	0.579

The two pipelines exhibit a precision-recall tradeoff that reflects their architectural differences.

Traditional retrieval favors recall. With top-k=5 chunk retrieval, the traditional pipeline surfaces nearly 80% of ground-truth source pages on average. This broad retrieval strategy ensures the generation model usually has access to the relevant information, at the cost of including irrelevant chunks in the context window. The low precision (0.205) indicates that most retrieved chunks do not correspond to ground-truth source pages—though this may partially reflect a scoring artifact, since chunks from adjacent pages that provide supporting context are penalized as irrelevant even when they contribute to a correct answer.

PageIndex retrieval favors precision. The tree traversal identifies fewer pages overall but with modestly higher precision (0.244 vs. 0.205). However, its lower recall (0.579) means it misses over 40% of relevant source pages on average. For multi-location synthesis questions, this targeted-but-incomplete retrieval likely explains the poor accuracy: if the traversal terminates at one relevant branch but fails to navigate to a second, the generation model cannot synthesize across both locations.

The correlation between retrieval recall and answer accuracy is strong: the traditional pipeline's higher recall (0.799 vs. 0.579) maps directly onto its higher correctness rate (47.2% vs. 22.2%). Providing the generation model with more relevant context, even at the cost of diluting it with irrelevant chunks, produces better answers than providing less context with marginally better targeting.

Hallucination and Citation Fidelity

Pipeline	Hallucination Rate	Citation Fidelity
Traditional	0.000	0.000
PageIndex	0.255	0.051

The traditional pipeline produces zero hallucinations. Every claim in its generated answers is supported by the retrieved chunk context. This is a direct consequence of the architecture: the generation model sees only the retrieved chunks and is instructed to answer from that context. When the relevant information is present in the chunks (high recall), the model generates grounded answers. When it is absent, the model tends to produce partial rather than fabricated responses.

PageIndex hallucinates approximately one quarter of its claims. A hallucination rate of 0.255 means that for every four claims in a PageIndex answer, one is unsupported by the retrieved context. Combined with the high partial rate, this suggests PageIndex's generation step mixes genuinely retrieved information with elaboration or inference that goes beyond what the tree traversal surfaced. Whether this reflects the PageIndex API's internal prompt design (encouraging the model to be comprehensive rather than conservative) or a limitation of the tree-based context selection is not observable from outside the API.

Citation fidelity is near-zero for both pipelines. Neither pipeline consistently maps claims to specific page numbers in a format the LLM-as-judge scorer can reliably detect. The traditional pipeline formats context as `[Pages N, M]\n{text}`, but the generation model does not consistently propagate these page references into its answer text. PageIndex answers occasionally include page references (fidelity of 0.051) but not systematically. This metric requires either prompt engineering to force explicit citations or a redesigned scoring approach; in its current form, it does not meaningfully differentiate the pipelines.

Comparability caveat. The hallucination metric is not perfectly symmetric across pipelines. The traditional pipeline's quality scoring uses actual retrieved chunks as context, while PageIndex quality scoring substitutes ground-truth text because the PageIndex API does not expose the raw retrieved content. This means PageIndex hallucination is measured against an idealized context rather than what the model actually saw, potentially overstating the rate.

Cost Breakdown

Pipeline	Total Cost	Cost per Question	Cost per Correct Answer
Traditional	\$0.234	\$0.0065	\$0.0137
PageIndex	\$0.154	\$0.0043	\$0.0193

PageIndex is cheaper in absolute terms—34% lower total cost across the evaluation. However, when normalized by output quality, the economics invert: the traditional pipeline's cost per correct answer (\$0.0137) is 29% lower than PageIndex's (\$0.0193) because its higher accuracy amortizes the cost over more successful queries.

The cost structures differ qualitatively. Traditional pipeline costs are dominated by Claude Sonnet 4 generation tokens ($3.00/$15.00 per million input/output), with Voyage embedding costs negligible at $0.06 per million tokens. PageIndex costs combine per-page indexing ($0.01/page) with per-query charges ($0.03 base plus token usage). For the corpus and question set tested, the traditional pipeline's higher per-query generation cost is partially offset by the absence of per-page indexing charges—embedding the chunked corpus is cheap relative to PageIndex's page-level indexing fee.

The cost comparison would shift under different usage patterns. A large corpus queried infrequently would favor the traditional pipeline's cheap embedding-based indexing. A small corpus queried heavily would favor PageIndex's lower per-query marginal cost—provided its accuracy gap were acceptable.

Latency Comparison

Pipeline	Avg Latency	P95 Latency
Traditional	4,713 ms	—
PageIndex	27,229 ms	55,800 ms

The traditional pipeline responds 5.8x faster on average. Its latency is dominated by a single Claude API call (the Voyage embedding and FAISS search contribute negligible overhead), producing responses in under 5 seconds.

PageIndex's latency reflects its multi-step architecture: document submission, tree traversal polling (submit query, wait for retrieval completion), and finally the chat completions call. The polling step alone accounts for the majority of the overhead—each poll interval is 2 seconds, and retrieval may require multiple polls to complete. The P95 latency of 55.8 seconds means that roughly 1 in 20 queries takes nearly a minute, a significant constraint for interactive applications.

The latency difference is architectural rather than incidental. Vector similarity search is a single sub-millisecond operation followed by one generation call. Tree traversal requires multiple sequential LLM reasoning steps (one per tree level), each adding network round-trip and inference time. This is the fundamental cost of replacing geometric lookup with reasoning-based retrieval: each query pays for the reasoning at query time rather than amortizing it into the indexing step.

Stability Across Runs

Comparing the 36-question primary run against the 16-question pilot provides a rough stability check, though the question sets overlap (the pilot's 16 questions are a subset of the primary run's 36).

Metric	PageIndex (pilot → primary)	Traditional (pilot → primary)
Accuracy	18.8% → 22.2%	56.2% → 47.2%
Hallucination	0.062 → 0.255	0.000 → 0.000
Avg Latency	31,305 ms → 27,229 ms	5,334 ms → 4,713 ms

Traditional accuracy declined from 56.2% to 47.2% with the expanded question set, suggesting the pilot over-represented easier questions. PageIndex's hallucination rate increased substantially (0.062 to 0.255), likely reflecting the additional questions targeting more complex passages where the tree traversal retrieves less complete context. Latency improved slightly for both pipelines, consistent with normal API response time variance rather than a systematic change.

The traditional pipeline's zero hallucination rate is stable across both runs, reinforcing that its chunk-based context constraint is architecturally robust rather than an artifact of the pilot's question selection.

VII. Discussion

Where PageIndex Outperforms, Matches, or Underperforms

The results reveal a clear performance hierarchy rather than a nuanced tradeoff space.

PageIndex underperforms on answer correctness. The 25-percentage-point accuracy gap (47% vs. 22%) is substantial and consistent across difficulty levels. This is not a marginal difference that might reverse with parameter tuning; it reflects a fundamental retrieval limitation. PageIndex's tree traversal misses relevant content that vector similarity reliably surfaces.

PageIndex underperforms on hallucination. A 25.5% hallucination rate versus 0% is not a tradeoff—it is a reliability gap. For financial document analysis where unsupported claims carry regulatory and business risk, this difference alone may disqualify PageIndex from production consideration regardless of other metrics.

PageIndex matches on multi-location synthesis. Both pipelines perform poorly (14% correct) on questions requiring information from multiple document sections. This parity suggests that neither architecture, as configured, solves the multi-hop retrieval problem. The failure mode differs—vector search retrieves broadly but without synthesis guidance; tree traversal navigates to one location but fails to identify others—but the outcome is equivalent.

PageIndex outperforms on per-query cost. At $0.0043 versus $0.0065 per question, PageIndex is 34% cheaper in raw terms. This advantage evaporates when normalized by quality (cost per

correct answer favors traditional by 29%), but for workloads where partial answers have value or where human review catches errors, the absolute cost difference may matter.

PageIndex underperforms on latency. The 5.8x latency gap (27 seconds vs. 5 seconds average) and the P95 tail of 56 seconds make PageIndex unsuitable for interactive applications. Users will not wait a minute for one-in-twenty queries. Batch processing or asynchronous workflows could tolerate this latency, but the accuracy gap makes it unclear why one would accept the tradeoff.

The hypothesis motivating this evaluation—that reasoning-based retrieval might outperform vector similarity on complex queries while underperforming on simple lookups—is not supported. PageIndex underperforms across the board, with no question type or difficulty level where it demonstrates an advantage.

Question-Type Sensitivity

The question-type breakdown suggests where each architecture's failure modes manifest.

Single-fact extraction (n=27) shows the largest gap: 56% traditional versus 26% PageIndex. This is the opposite of the expected pattern. Tree traversal should excel at navigating to a specific location when the question implies a clear structural target ("What year was the building constructed?" points to a property description section). Instead, vector similarity's broad retrieval—surfacing multiple potentially-relevant chunks—outperforms targeted navigation. One interpretation: the tree's node summaries are insufficiently detailed to guide traversal to the right page, while embedding similarity reliably matches question terms to passage terms even without structural reasoning.

Multi-location synthesis (n=7) shows parity at 14%. Neither architecture handles questions requiring information from pages 12 and 47 combined. Vector search retrieves nearby chunks that may not span the needed locations; tree traversal descends one branch and terminates. Solving this failure mode likely requires explicit multi-hop retrieval strategies—iterative search, query decomposition, or graph-based navigation—that neither pipeline implements.

Negative/absence questions (n=2) are too sparse for conclusions, but the pilot run showed both pipelines handling them well. This is expected: both architectures can recognize when retrieved context does not contain the requested information, provided retrieval is reasonably complete.

The absence of tabular reasoning and conditional logic questions in the primary run limits the analysis. Future work should ensure balanced representation across all question types to identify potential PageIndex strengths not captured here.

Cost-Quality Tradeoff Analysis

The cost comparison depends on how one values partial answers and how costs scale with corpus size and query volume.

Under strict correctness criteria, the traditional pipeline dominates. Its cost per correct answer ($0.0137) is 29% lower than PageIndex ($0.0193), and it produces 2.1x more correct answers per dollar spent. For workflows where only fully correct answers have value—automated data extraction, compliance checking, audit support—traditional RAG is unambiguously more cost-effective.

Under lenient criteria (counting partial as 0.5 correct), the gap narrows but does not close. Traditional achieves an effective 72% score (17 correct + 18×0.5 partial = 26 effective correct out of 36) versus PageIndex's 49% (8 + 19×0.5 = 17.5). Cost per effective correct answer: $0.009 traditional versus $0.0088 PageIndex—near parity, but with traditional still producing higher absolute quality.

At scale, the cost structures diverge. Traditional pipeline indexing costs (Voyage embeddings) scale at $0.06 per million tokens—negligible even for large corpora. PageIndex indexing costs $0.01 per page, or roughly $30 per 3,000-page corpus. For a corpus queried thousands of times, PageIndex's lower per-query cost ($0.0043 vs. $0.0065) would eventually offset the indexing premium—but only if accuracy is acceptable. At current accuracy levels, scaling PageIndex means scaling errors proportionally.

The latency cost is not priced in dollars but constrains use cases. A 27-second average response time excludes interactive workflows. If PageIndex queries must be batched overnight or run asynchronously, the operational complexity may exceed the dollar savings.

Limitations of This Evaluation

Several factors constrain the generalizability of these results.

Corpus size. Thirty-six evaluated questions across two documents is sufficient to identify large effect sizes but not to characterize performance distributions with precision. Confidence intervals on the accuracy estimates span roughly ±15 percentage points at 95% confidence. A larger corpus and question set would tighten these bounds and enable finer-grained analysis by document type.

Domain specificity. Commercial loan documents represent one vertical within financial services. Performance on SEC filings, insurance policies, legal contracts, or technical documentation may differ. The heterogeneous structure of the test corpus likely disadvantages tree-based retrieval relative to standardized document types.

Cloud API as black box. The PageIndex API does not expose retrieved context, internal prompts, or model configuration. Hallucination scoring against ground-truth text rather than

actual retrieved content may overstate the rate. The inability to inspect traversal decisions limits diagnosis of failure modes. A self-hosted deployment with full observability would enable deeper analysis.

Scoring framework sensitivity. The tiered correctness cascade (exact → fuzzy → semantic) determines how partial matches are credited. Different thresholds would shift the partial/incorrect boundary. The high partial rate for both pipelines (50–53%) suggests many answers are in the "right neighborhood" but phrased differently than ground truth; whether these represent genuine quality differences or scoring artifacts is not fully resolved.

Single evaluation run. Temperature 0.0 produces deterministic outputs, but API behavior may vary over time due to model updates or infrastructure changes. The pilot-to-primary comparison provides a rough stability check, but longitudinal evaluation would strengthen confidence in the findings.

Threats to Validity

Internal validity. The pipelines differ in more than retrieval mechanism: the traditional pipeline calls Anthropic directly while PageIndex uses Claude server-side with unknown prompt configuration. If PageIndex's internal prompt encourages comprehensive answers at the cost of grounding, this would inflate hallucination rates independent of retrieval quality. The comparison isolates retrieval mechanism imperfectly.

External validity. Results from commercial loan documents may not transfer to other domains. The 47% traditional accuracy matches FinanceBench baselines, suggesting the corpus is representative of financial document difficulty, but PageIndex's 22% may reflect domain mismatch rather than architectural limitation.

Construct validity. The metrics assume ground-truth answers are correct and complete. For questions with legitimate alternative answers or ambiguous phrasing, strict scoring penalizes valid responses. Human evaluation would address this but was not completed for the primary run.

Statistical conclusion validity. With n=36 questions, the evaluation has approximately 80% power to detect a 20-percentage-point accuracy difference at alpha = 0.05. The observed 25-point gap exceeds this threshold, suggesting the finding is robust. However, subgroup analyses (by question type, difficulty) have smaller samples and wider uncertainty.

Comparison to Published Benchmarks

VectifyAI reports that Mafin 2.5, their RAG system built on PageIndex, achieves 98.7% accuracy on the FinanceBench benchmark—substantially higher than both pipelines evaluated here. Three

factors explain this discrepancy, none of which invalidate either set of results.

System scope differs. The 98.7% figure reflects Mafin 2.5, described as VectifyAI's "latest RAG model" built *on* PageIndex, not the PageIndex Cloud API in isolation. Mafin 2.5 likely includes additional orchestration, prompt engineering, and possibly multi-step reasoning that the raw API does not expose. This evaluation tests the retrieval primitive; Mafin 2.5 is a tuned application layer. The distinction matters: a retrieval API underperforming does not preclude a well-engineered system built atop it from excelling, just as a high-performing system does not guarantee its components will transfer to other contexts.

Evaluation methodology differs. VectifyAI's benchmark methodology includes human re-annotation: "In cases where questions are ambiguous, have multiple valid answers, or are deemed invalid, we rely on expert human annotations to ensure fair and accurate evaluation." This is a reasonable approach for establishing ceiling performance but differs from the automated, ground-truth-fixed scoring used here. The FinanceBench paper itself notes that GPT-4-Turbo with retrieval achieved only 19% accuracy under strict evaluation; Ragie, an optimized RAG system, reached 51% on single-store retrieval. The traditional pipeline's 47% accuracy in this evaluation tracks the published baseline exactly, suggesting the scoring framework is calibrated appropriately.

Document characteristics differ. FinanceBench comprises standardized SEC filings—10-Ks, 10-Qs, and earnings reports—with predictable hierarchical structure (Item 1, Item 1A, Item 7, etc.) that tree-based indexing can exploit reliably. Commercial loan documents—credit memoranda, appraisals, bespoke loan agreements—exhibit heterogeneous structure that varies by institution, deal type, and preparer. Tree traversal may perform better on documents with consistent organizational patterns than on structurally diverse corpora. This evaluation deliberately targets the latter to assess generalization beyond benchmark-friendly filings.

These factors do not suggest VectifyAI's claims are invalid, nor do they indicate flaws in this evaluation. They reflect different questions: "How well can a tuned system perform on standardized documents under favorable scoring?" versus "How does the raw API perform on heterogeneous documents under strict automated evaluation?" Both answers are useful; practitioners should weight them according to their deployment context.

VIII. Conclusion

Summary of Findings

This evaluation tested the hypothesis that PageIndex's reasoning-based retrieval might match or exceed traditional vector-based RAG on complex document queries while trading off performance on simple lookups. The hypothesis is not supported.

Across 36 questions on commercial loan documents, the traditional RAG pipeline (PyMuPDF4LLM, Voyage 3 Large embeddings, FAISS retrieval, Claude Sonnet 4 generation) substantially outperformed the PageIndex Cloud API on every primary metric:

- **Answer correctness**: 47% vs. 22% (traditional leads by 25 percentage points)
- **Hallucination rate**: 0% vs. 25.5% (traditional produces no unsupported claims)
- **Query latency**: 4.7s vs. 27.2s average (traditional responds 5.8x faster)
- **Cost per correct answer**: $0.014 vs. $0.019 (traditional is 29% more cost-effective)

The only metric favoring PageIndex is raw cost per query ($0.0043 vs. $0.0065), an advantage that disappears when normalized by output quality.

The performance gap is consistent across question types and difficulty levels. Notably, both pipelines struggle equally on multi-location synthesis questions (14% correct each), suggesting that neither vector similarity nor tree traversal, as currently implemented, solves the multi-hop retrieval problem.

Practical Guidance

Based on these findings, practitioners evaluating PageIndex for document intelligence workflows should consider:

Favor traditional RAG when:

- Answer correctness is critical and errors carry business or regulatory risk
- Interactive latency (sub-10-second responses) is required
- Documents exhibit heterogeneous structure without consistent hierarchical organization
- Hallucination must be minimized (e.g., financial reporting, compliance, legal analysis)
- The corpus will be queried frequently, amortizing indexing costs

Consider PageIndex when:

- Documents have highly consistent, predictable structure (standardized forms, regulatory filings)
- The full Mafin 2.5 system—not just the raw API—is available and appropriate for the domain
- Partial or approximate answers have value and human review is standard practice
- Latency tolerance exceeds 30 seconds (batch processing, asynchronous workflows)
- Eliminating vector infrastructure is a priority and the accuracy tradeoff is acceptable

Consider neither when:

- Questions require multi-location synthesis across non-adjacent document sections
- The task demands explicit citation of sources in generated answers

- Perfect retrieval recall is required

For most enterprise document intelligence applications—where accuracy, latency, and auditability matter—traditional RAG remains the pragmatic choice. PageIndex's architectural promise (structure-aware, reasoning-based retrieval) does not yet translate into measurable advantages over the simpler, faster, more reliable vector pipeline.

Future Work

Several directions could extend or refine these findings:

Larger and more diverse corpora. Expanding beyond commercial loan documents to SEC filings, legal contracts, technical manuals, and other domains would test whether PageIndex's underperformance is domain-specific or architectural. A corpus of 200+ documents with 500+ questions would enable statistically robust subgroup analyses.

Self-hosted PageIndex comparison. The open-source PageIndex repository allows local deployment with full control over the tree construction and traversal process. Comparing self-hosted PageIndex (with observable intermediate steps) against the Cloud API would isolate whether the performance gap reflects the API's constraints or the fundamental approach.

Hybrid architectures. Combining vector retrieval with tree-based filtering—or using PageIndex's tree structure to guide chunk selection—might capture benefits of both approaches. The structural metadata PageIndex generates could augment traditional RAG without replacing it.

Multi-hop retrieval strategies. Both pipelines fail on multi-location synthesis. Iterative retrieval, query decomposition, or graph-based navigation could address this shared weakness. Evaluating such strategies against the baselines established here would quantify their incremental value.

Longitudinal evaluation. Repeating the evaluation periodically as both PageIndex and traditional RAG tooling evolve would track whether the performance gap narrows, widens, or shifts to different question types.

Human evaluation. Completing blinded human scoring on the full question set would validate automated metrics and characterize the qualitative differences between pipeline outputs that correctness scores alone do not capture.

This evaluation provides a data point, not a verdict. PageIndex represents a genuine architectural alternative to vector-based RAG, and its reasoning-based approach may prove valuable in contexts not tested here. For the commercial loan documents and question types evaluated, however, traditional RAG delivers better answers, faster, with fewer hallucinations, at lower effective cost.

Practitioners should weight these findings against their specific requirements—and remain open to re-evaluation as both approaches mature.