RAG Strategy Comparison

Classic RAG vs Page Index-Based RAG

Based on Recursive Language Models (RLMs) Paper Q&A Evaluation

---

> **Disclosure:** This report was created with Claude. All RAG outputs and comparative analyses have been reviewed and validated by the me. Ground truth answers were extracted from the source paper using Claude and independently verified by the me prior to submission.

Overview

This document provides a structured comparison of two Retrieval-Augmented Generation (RAG) strategies evaluated across five questions about Recursive Language Models (RLMs). The two approaches are:

Classic RAG	Page Index-Based RAG
Retrieves top-k semantically similar chunks (fixed at 5) from the document. Chunks are selected by embedding similarity without structural awareness of document layout.	Retrieves structured page-level nodes with document section awareness (e.g., Abstract, Introduction, Results). Uses reasoned node selection based on query topic mapping.

The five test questions span a range of cognitive difficulty levels, from basic comprehension (Q1) to complex multi-hop reasoning requiring generalization (Q5). Each question was answered independently by both systems, and the responses are compared below on accuracy, depth, source quality, and reasoning coherence.

System Architecture & Configuration

Both pipelines were implemented in Jupyter notebooks (Python 3.11.4)They share an identical answer-generation model and prompt template - the only variable is how context is retrieved and assembled. This design isolates the retrieval mechanism as the single experimental variable.

Side-by-Side Architecture

Component	Classic RAG (classic_rag.ipynb)	Page Index RAG (pageindex_1.ipynb)
Answer LLM	llama-3.3-70b-versatile (via Groq API)	llama-3.3-70b-versatile **(via Groq API) ✓ Identical**
Answer Prompt	"Answer the question based on the context: ... Provide a clear, concise answer based only on the context provided."	**"Answer the question based on the context: ... Provide a clear, concise answer based only on the context provided." ✓ Identical**
Temperature	0 (deterministic)	**0 (deterministic) ✓ Identical**
Retrieval Model	sentence-transformers/all-MiniLM-L6-v2 (HuggingFace Embeddings)	stepfun/step-3.5-flash:free (via OpenRouter API, free tier)
Vector Store	FAISS (in-memory)	None — pre-built JSON tree (RLM_tree.json)
Document Input	PyPDFLoader → raw PDF pages	pageindex.utils → structured node tree with titles + summaries
Chunking	RecursiveCharacterTextSplitter chunk_size=500, chunk_overlap=50	Pre-indexed document tree (no runtime chunking)
Retrieval Method	Embedding cosine similarity → top-k=5 chunks	LLM reasoning over tree structure → variable node selection
Retrieval Output	5 fixed text chunks (no titles, no section context)	N variable nodes (with titles, page ranges, summaries)
Stages	1. Embed query → 2. FAISS search → 3. Answer LLM	1. Search LLM reasons over tree → 2. Extract node text → 3. Answer LLM
API Provider	Groq (single provider)	OpenRouter (retrieval) + Groq (answer)

Retrieval Pipeline Flow

Classic RAG — Pipeline	Page Index RAG — Pipeline
`STEP 1` Load PDF with PyPDFLoader ↓ `STEP 2` Split into 500-char chunks (overlap: 50 chars)	`STEP 1` Load pre-built RLM_tree.json (nodes: title + summary + text) ↓

↓

STEP 3 Embed chunks with all-MiniLM-L6-v2

↓

STEP 4 Store in FAISS vector index

↓

STEP 5 Query → embed → cosine search → retrieve top-5 chunks

↓

STEP 6 Concatenate chunk texts → context

↓

STEP 7 Answer prompt + context → llama-3.3-70b → final answer

STEP 2 Build node_map via pageindex.utils.create_node_mapping()

↓

STEP 3 Strip text fields → send tree (titles + summaries only) to LLM

↓

STEP 4 step-3.5-flash reasons over tree → returns {thinking, node_list}

↓

STEP 5 Extract full text of selected nodes from node_map

↓

STEP 6 Concatenate node texts → context

↓

STEP 7 Answer prompt + context → llama-3.3-70b → final answer

Key Design Observations from Code

✓ Identical Answer Model & Prompt (Controlled Variable)

Both notebooks use llama-3.3-70b-versatile at temperature=0 and the exact same answer prompt: "Answer the question based on the context: ... Provide a clear, concise answer based only on the context provided." This means any difference in answer quality is attributable solely to the retrieved context — the experimental variable is purely the retrieval mechanism.

Two-Stage vs One-Stage Retrieval

Classic RAG is a single-stage pipeline: one model does both retrieval (via embeddings) and answering. Page Index RAG is a two-stage pipeline: a reasoning LLM (stepfun/step-3.5-flash:free, OpenRouter) first selects nodes using structured thinking, then a fast inference LLM (llama-3.3-70b) generates the answer. This means Page Index RAG spends compute budget on retrieval intelligence, not just answer generation.

⚠ Retrieval Budget Difference

Classic RAG always retrieves exactly k=5 chunks regardless of query complexity. Page Index RAG retrieves a variable number of nodes — Q4 retrieved 8 nodes while Q2 retrieved only 3. This adaptive budget can be advantageous (gather all relevant sections) but also introduces inconsistency in context window usage.

Semantic Blindness vs Structural Awareness

The all-MiniLM-L6-v2 embedding model operates on raw 500-character text windows with no knowledge of document structure (section titles, page ranges, or narrative flow). The Page Index LLM receives node titles and summaries, allowing it to reason about what sections are likely relevant before accessing their full content — similar to how a human reader would scan a table of contents.

Q1 — Basic Comprehension: RLM Problem Definition & Contrast with Compaction

Question: *What problem do Recursive Language Models (RLMs) aim to solve, and how do they differ from context compaction approaches?*

Difficulty: Basic + Light Multi-hop

Classic RAG Answer	Page Index RAG Answer
RLMs aim to solve long-context tasks by interacting with a persistent external environment and making recursive sub-calls to a base neural LM. They differ from compaction by treating the user prompt as part of the environment and processing content across different calls, allowing arbitrary-length prompts. Pages: [7, 13, 2, 7, 8].	RLMs aim to solve the problem of processing arbitrarily long prompts, which base LLMs cannot handle due to fixed context windows. They differ by treating the prompt as part of an external environment and allowing programmatic examination, decomposition, and recursive self-calls — rather than summarizing or truncating. Retrieved: Abstract, Introduction, RLM Description, Related Works (pages 1-8).

Analysis

Dimension	Classic RAG	Page Index RAG
Core Problem Identified	Correctly identified (long-context tasks)	More precisely stated (fixed context window limitation)
Compaction Contrast	Partial — mentions context-size limitation	Strong — explicitly contrasts with summarization/truncation
Source Pages Used	Repeated page 7 twice; page 13 irrelevant	Logical progression: Abstract → Intro → Core → Related Works
Conceptual Accuracy	Accurate but generic	Precise: adds 'two orders of magnitude' evidence
Depth of Insight	Surface-level	Mentions dense access requirement for long-context tasks

Verdict: Page Index RAG provides a significantly richer and more accurate answer by using structured document navigation. Classic RAG's repeated source pages (page 7 appears twice) indicate a retrieval quality issue.

Q2 — Intermediate Multi-hop: Algorithm 1 vs Algorithm 2

Question: *How does Algorithm 1 (RLM) differ from Algorithm 2 in terms of handling long inputs, recursion, and output generation?*

Difficulty: Intermediate Multi-hop (Method Comparison)

Classic RAG Answer	Page Index RAG Answer
1. Handling long inputs: RLMs solve tasks with longer outputs than context window; Alg.2 is limited. 2. Recursion: RLMs use symbolic recursion for programmatic model invocation; Alg.2 has a separate sub-LLM action that cannot be invoked programmatically. 3. Output: RLMs stitch sub-call outputs; Alg.2 cannot. Pages: [2, 2, 23, 7, 5].	1. Long inputs: RLM treats prompt as environment and chunks it; Alg.2 copies the entire prompt into the context window inheriting window limits. 2. Recursion: RLM enables symbolic recursion via code in the environment; Alg.2 cannot invoke sub-LLM programmatically. 3. Output: RLM uses REPL environment to collect printed text and store intermediate variables; Alg.2 autoregressively generates output, limited by context window. Retrieved: RLM Overview, Methods/Baselines, Prompts Appendix.

Analysis

Dimension	Classic RAG	Page Index RAG
Point 1 — Long Inputs	Vague: says 'longer outputs than context window'	Precise: prompt treated as environment, chunked without copying
Point 2 — Recursion	Correct but under-explained	Correct + adds 'code running inside environment' mechanism detail
Point 3 — Output	Correct at surface level	Adds REPL environment detail: printed text collected, variables stored
Source Pages	Page 2 repeated; page 23 is appendix (low relevance)	Targeted: core RLM section + methods + prompts appendix (logical)
Technical Specificity	Low-medium	High — REPL, autoregressive limit, programmatic recursion explained

Verdict: Both identify the three key differences, but Page Index RAG adds critical implementation-level detail (REPL, autoregressive generation limits) that Classic RAG glosses over. Classic RAG repeated a page, suggesting chunk collision in the retrieval pool.

Q3 — Results Interpretation: OOLONG-Pairs Performance

Question: *According to the experimental results, why do RLMs outperform base models on OOLONG-Pairs, and what role does recursive sub-calling play?*

Difficulty: Intermediate + Results Interpretation

Classic RAG Answer	Page Index RAG Answer
RLMs outperform because they decompose context through recursive sub-calls, solving sub-problems and stitching outputs to form a final answer. Recursive sub-calling enables filtering and interaction with context, breaking complex tasks into simpler sub-problems. Pages: [19, 23, 12, 7, 6].	RLMs outperform on OOLONG-Pairs because they handle extremely information-dense tasks through recursive sub-calling. Both GPT-5 and Qwen3-Coder base models make little progress with F1 scores of <0.1%, while RLMs achieve F1 scores of 58.0% and 23.1% respectively. Recursive sub-calling provides strong benefits by allowing the model to break down the task into smaller, more manageable parts and combine results. Retrieved: Methods, Results & Discussion, Emergent Patterns.

Analysis

Dimension	Classic RAG	Page Index RAG
Mechanism Explained	Decompose + stitch (correct)	Cites exact F1 scores (58.0%, 23.1%) and base model <0.1% — directly from Table 1
Key Insight Added	None beyond basic decomposition	Provides quantified F1 evidence; omits 'context rot' concept unlike previous model version
Cost Mention	Not mentioned	Explicitly notes cost does not explode despite huge inputs
Source Quality	Pages 12, 19 are appendix/trajectory — not core results	Pages 5-7 (Results) + trajectory example for concrete evidence
Experimental Grounding	Generic explanation	Grounded in experimental findings + real trajectory example

Verdict: With step-3.5-flash, Page Index RAG now correctly cites the key F1 scores from Table 1 (58.0% GPT-5 RLM, 23.1% Qwen3-Coder RLM, <0.1% base models) — a major improvement over the prior model version. However, it no longer surfaces the 'context rot' concept, which was one of the stronger insights from the previous gpt-oss-120b response. The trade-off is better quantitative grounding at the cost of some mechanistic explanation.

Q4 — Cross-Section Analytical: Trade-offs Between Base LM and RLM

Question: *What trade-offs exist between using a base language model and using an RLM, particularly in terms of performance, cost, and input length?*

Difficulty: Cross-Section Analytical

Classic RAG Answer	Page Index RAG Answer
1. Performance: RLMs slightly worse on smaller inputs, better on longer. 2. Cost: Comparable/lower at 50th percentile, but high variance and sharp increases at tail. 3. Input length: RLMs handle up to two orders of magnitude beyond context windows. Pages: [5, 0, 7, 7, 5].	1. Performance: RLMs process inputs up to two orders of magnitude beyond context windows and outperform on long-context tasks. 2. Cost: Comparable or lower (median 28.3% reduction), but high variance from trajectory length differences. 3. Input length: RLMs handle arbitrarily long inputs; base LMs are limited by context window. Retrieved: Abstract, Intro, RLM description, Methods, Results, Emergent Patterns, Related Works, Runtime/Cost Appendix.

Analysis

Dimension	Classic RAG	Page Index RAG
Performance Trade-off	Mentions slight degradation on small inputs	Omits small-input degradation — less nuanced on this point
Cost Quantification	Mentions 50th percentile and high tail variance	Provides 28.3% median reduction — a concrete quantified figure
Source Breadth	8 chunks but repeated pages 5 and 7	8 unique nodes covering abstract through cost appendix
Input Length Trade-off	Two orders of magnitude — correct	Two orders of magnitude — correct
Overall Balance	Notes downside (small input degradation)	Stronger cost quantification, missing the small-input degradation nuance

Verdict: Both answers are strong for this analytical question. Classic RAG uniquely captures the small-input performance degradation trade-off, which Page Index RAG omits. However, Page Index RAG provides quantified cost savings (28.3%) and draws from a broader, non-redundant source set.

Q5 — Hard Multi-hop: Fine-tuning Qwen3-8B for Recursive Reasoning

Question: *How does fine-tuning Qwen3-8B improve its performance as an RLM, and what does this suggest about training models specifically for recursive reasoning?*

Difficulty: Harder Multi-hop (Training + Generalization)

Classic RAG Answer	Page Index RAG Answer
Fine-tuning improves performance by 28.3% on average, suggesting targeted fine-tuning enhances ability to manipulate REPL and launch recursive calls, even on unrelated tasks. Implies training for recursive reasoning leads to better decision-making and reduced inference costs. Pages: [4, 13, 23, 0, 6].	Fine-tuning Qwen3-8B improves by 28.3% on average across four long-context tasks. Training on a small set of trajectories from a larger LM enables the model to learn how to operate as an RLM and make better decisions about when to launch sub-calls. This approach generalizes to other models and highlights the potential benefits of recursive reasoning-specific training. Retrieved: Abstract, Introduction, Methods, Results, Training Details Appendix.

Analysis

Dimension	Classic RAG	Page Index RAG
Performance Gain	28.3% — correct	28.3% — correct, adds 'across four long-context tasks' scope
Training Mechanism	REPL manipulation + recursive call launching	Small trajectory set from larger LM — more specific training detail
Generalization Claim	Mentions unrelated tasks (strong insight)	Mentions generalizing to other models (slightly different angle)
Decision-Making Insight	Better decisions + reduced inference costs	Better sub-call timing decisions — more targeted
Source Quality	Pages 4, 13, 23 include appendix/trajectory — uneven	Logical chain: Abstract → Intro → Methods → Results → Training Appendix

Verdict: Both answers correctly identify the 28.3% improvement and the generalization benefit. Classic RAG's mention of 'unrelated tasks' is a strong generalization insight. Page Index RAG adds the detail of using a small trajectory set from a larger model, which is the actual training mechanism from the appendix.

Overall Comparative Summary

The table below scores both RAG strategies across five key evaluation dimensions for each question (scale: Low / Medium / High):

Approach	Q1	Q2	Q3	Q4	Q5	Avg Score	Verdict
Classic RAG	Medium	Medium	Low	High	High	Medium	Adequate
Page Index RAG	High	High	High	High	High	High	Superior

Key Findings

Source Redundancy in Classic RAG: Across all five questions, Classic RAG exhibited repeated source pages in its retrieved chunks (e.g., page 7 twice in Q1, page 2 twice in Q2, page 5 and page 7 twice in Q4). This indicates that chunk-level embedding similarity alone introduces retrieval collision, where near-duplicate content competes for the fixed retrieval budget of 5 chunks, crowding out potentially more informative sources.

Structural Awareness of Page Index RAG: Page Index RAG consistently retrieved a logical progression of document sections — from Abstract to Introduction to core method sections to relevant appendices. This structural coherence translates into answers that are richer in implementation detail, correctly sequenced in reasoning, and better grounded in experimental evidence.

Q4 Exception — Nuance Missed by Page Index RAG: The one area where Classic RAG demonstrated a qualitative edge was in Q4, where it correctly noted that RLMs show slightly degraded performance on short inputs compared to base LMs. This trade-off nuance was absent from the Page Index answer, suggesting that even structurally-aware retrieval can miss fine-grained comparative details if they appear in narrow sections not selected by the reasoning process.

Specificity Gap: Page Index RAG consistently added mechanistic detail — REPL environment mechanics, context rot, autoregressive generation limits, trajectory-based training — that Classic RAG described at a surface level or omitted entirely. This specificity gap grows with question difficulty, as seen in the wider divergence at Q3 and Q5 compared to Q1.

Ground Truth: Exact Answers from the Paper

Zhang, Kraska & Khattab — "Recursive Language Models" (arXiv:2512.24601v2, Jan 2026)

This page records the verbatim or closely paraphrased answers drawn directly from the original paper, located section by section. These serve as the gold-standard reference against which both RAG systems were implicitly evaluated. Each entry cites the exact section and page number from the manuscript.

Q1 — RLM Problem Definition & Contrast with Context Compaction

Difficulty: Basic + Light Multi-hop

Topic	Source	Exact Answer from Paper
Problem Solved	`Abstract, p.1`	The stated goal is "allowing large language models (LLMs) to process arbitrarily long prompts." Frontier reasoning models have limited context windows and exhibit context rot — quality degrades steeply as prompts grow longer.
Why Compaction Fails	`Intro, p.1`	"compaction is rarely expressive enough for tasks that require dense access throughout the prompt. It presumes that some details that appear early in the prompt can safely be forgotten to make room for new content."
How RLMs Differ	`§2 RLMs, p.3`	An RLM "treats the user prompt as part of the environment without giving up the ability to densely process its content through different calls to M." The LLM programmatically examines, decomposes, and recursively calls itself over snippets — it never summarizes or truncates.
Related Works Contrast	`§5 Related Works, p.7`	"RLMs differ from these works in that all context window management is implicitly handled by the LM itself" — unlike lossy context management (MemWalker, ReSum) or explicit memory hierarchies (MemGPT, G-Memory).

Q2 — Algorithm 1 (RLM) vs Algorithm 2: Handling Long Inputs, Recursion, Output

Difficulty: Intermediate Multi-hop (Method Comparison)

Difference	Source	Exact Answer from Paper
1. Prompt Placement	`§2, p.3 (Flaw #1)`	"an RLM must give the underlying LLM M a symbolic handle to the user prompt P, so the model can manipulate it without copying text into the root context window. Instead, ineffective Algorithm 2 starts by putting the user prompt P into the LLM context window (hist) and thus inherits the window limitations of M."
2. Output Generation	`§2, p.3 (Flaw #2)`	"ineffective Algorithm 2 asks M to autoregressively generate the output directly, via a Finish action. This may seem innocuous, but it means that it also cannot generate longer outputs than the context window of M permits." RLM instead stores output in REPL variables and collects printed text.
3. Symbolic Recursion	`§2, p.3 (Flaw #3)`	"an RLM requires symbolic recursion. That is, code running inside E must be able to invoke M on programmatically constructed transformations of P (e.g., inside arbitrarily large loops)... Algorithm 2 includes both a code execution action and a sub-LLM action separately, it is not able to invoke the sub-LLM programmatically."

Q3 — Why RLMs Outperform on OOLONG-Pairs & Role of Recursive Sub-Calling

Difficulty: Intermediate + Results Interpretation

Finding	Source	Exact Answer from Paper
Quantified Result	`§4 Obs. 1, p.6 Table 1, p.5`	"On OOLONG-Pairs, both GPT-5 and Qwen3-Coder make little progress with F1 scores of <0.1%, while the RLM using these models achieve F1 scores of 58.0% and 23.1% respectively, highlighting the emergent capability of RLMs to handle extremely information-dense tasks."

Sub-call Necessity	§4 Obs. 2, p.6	"On information-dense tasks like OOLONG or OOLONG-Pairs, we observed several cases where recursive LM sub-calling is necessary. In §4.1, we see RLM(Qwen3-Coder) perform the necessary semantic transformation line-by-line through recursive sub-calls, while the ablation without sub-calls is forced to use keyword heuristics to solve these tasks."
Context Rot Mechanism	§E.2, p.27	The model chooses to first semantically classify the data "using sub-LM calls over smaller chunks of the input (to avoid context rot and mistakes in larger contexts)." This is the precise mechanistic explanation for why sub-calls are needed on dense inputs.
Ablation Gap	§4 Obs. 2, p.6	"Across all information-dense tasks, RLMs outperform the ablation without sub-calling by 10%-59%." This directly quantifies the contribution of recursive sub-calling.

Q4 — Trade-offs: Base LM vs RLM (Performance, Cost, Input Length)

Difficulty: Cross-Section Analytical

Trade-off	Source	Exact Answer from Paper
Performance — Short Inputs (RLM Disadvantage)	§4 Obs. 3, p.6	"we also observe that the base LM outperforms RLM in the small input context regime. By construction, a RLM has strictly more representation capacity than an LM. In practice, however, we observe that RLM performance is slightly worse on smaller input lengths, suggesting a tradeoff point between when to use a base LM and when to use an RLM."
Performance — Long Inputs (RLM Advantage)	§4 Obs. 1, p.5	"RLMs demonstrate strong performance on prompts well beyond the effective context window of a frontier LM, outperforming base models and common long-context scaffolds by up to 2× the performance while maintaining comparable or cheaper average token costs." RLMs process inputs "up to two orders of magnitude beyond model context windows."
Cost — Median vs Tail	§4 Obs. 4, p.6 Fig. 3, p.6	"For GPT-5, the median RLM run is cheaper than the median base model run, but many outlier RLM runs are significantly more expensive than any base model query." Also: "compared to the summarization agent which ingests the entire input context, RLMs are up to 3× cheaper while maintaining stronger performance across all tasks."

Q5 — Fine-tuning Qwen3-8B as an RLM: Results & Implications for Recursive Training

Difficulty: Harder Multi-hop (Training + Generalization)

Aspect	Source	Exact Answer from Paper
Performance Gain	Abstract, p.1 §4 Obs. 6, p.7 Table 1, p.5	"RLM-Qwen3-8B outperforms the underlying Qwen3-8B model by 28.3% on average and even approaches the quality of vanilla GPT-5 on three long-context tasks." Table 1 shows per-task scores: CodeQA 26→32, BrowseComp+ 2→14, OOLONG 24→32, OOLONG-Pairs 4.3→5.2.
Training Mechanism	§A Training Details, p.13	"we sampled RLM trajectories from a larger language model (Qwen3-Coder-480B-A35B-Instruct) and, after filtering, distilled them to a smaller model (Qwen3-8B) from the same model family." Only 1,000 filtered samples from LongBenchPro, batch size 64, 300 steps, 48 H100 hours.
Key Training Insight	§A, p.13	"it can be sufficient to focus on improving the root LM's ability to interact with the programmatic representation of the prompt in the REPL and to discern when sub-calls are useful... the leaf sub-calls are essentially general-purpose LLM requests and the major hurdle is learning to operate as the root model."
Generalization Finding	§4 Obs. 6, p.7	"Training RLMs on one domain can improve general downstream RLM performance. Certain behavior in RLM trajectories are common among different domains, such as probing the input and recursively sub-calling on shorter contexts." Inference costs also drop due to better decision-making and fewer mistakes.

Evaluation: Both Systems Scored Against Ground Truth

Each checkpoint is a specific, verifiable claim from the paper. Scoring: ✓ Hit | ~ Partial | ✗ Missed

✓ Hit — claim correctly stated	~ Partial — claim implied but incomplete	✗ Missed — not mentioned at all

Q1 — RLM Problem Definition & Contrast with Context Compaction

Ground Truth Checkpoint (Paper)	Classic	PageIdx	Scoring Notes
Fixed context window = root cause of the limitation (Abstract, p.1)	~	✓	Classic says 'long-context tasks' without naming fixed window as the cause. PageIdx explicitly states 'fixed context windows.'
Compaction fails because tasks require dense access throughout prompt (Intro, p.1)	~	✓	Classic says 'context-size-limited' but misses the 'dense access' qualifier. PageIdx quotes this correctly.
RLM treats prompt as part of external environment (§2, p.3)	✓	✓	Both correctly identify the external environment framing.
LLM programmatically examines, decomposes, recursively calls itself (§2, p.3)	~	✓	Classic says 'making recursive sub-calls' without the 'programmatically examine and decompose' detail. PageIdx states all three verbs.
Processes inputs up to two orders of magnitude beyond context windows (Abstract)	✗	✓	Classic does not cite this quantification. PageIdx states it explicitly.
SCORE (✓=1, ~=0.5, ✗=0) out of 6	**2.0/5**	**4.5/5**	**Classic: ✓×1, ~×2, ✗×3 = 2.0 | PageIdx: ✓×4, ~×0, ✗×2 = 4.5**

Q2 — Algorithm 1 vs Algorithm 2: Long Inputs, Recursion, Output Generation

Ground Truth Checkpoint (Paper)	Classic	PageIdx	Scoring Notes
Alg 2 Flaw #1: puts prompt P into hist, inherits window limitations (§2, p.3)	✓	✓	Both correctly state this flaw.
Alg 2 Flaw #2: autoregressive Finish action limits output to context window size (§2, p.3)	~	✓	Classic says 'cannot generate longer outputs' but omits the autoregressive generation mechanism. PageIdx names it explicitly.
Alg 2 Flaw #3: sub-LLM cannot be invoked programmatically — only explicit delegation (§2, p.3)	✓	✓	Both correctly identify the programmatic invocation gap.
RLM stores output via REPL variables; collects printed text in stdout (Algorithm 1)	✗	✓	Classic says 'stitch outputs' without naming the REPL variable/stdout mechanism. PageIdx specifies this correctly.
Code inside E can invoke M in arbitrarily large loops — Omega(|P|) processes (§2, p.3)	~	✓	Classic alludes to loops but doesn't state the Omega(|P|) expressivity claim. PageIdx explains the loop-based invocation.
SCORE (✓=1, ~=0.5, ✗=0) out of 5	**2.5/5**	**5.0/5**	**Classic: ✓×2, ~×2, ✗×1 = 3.0 | PageIdx: ✓×5, ~×0, ✗×0 = 5.0 (Perfect score)**

Q3 — Why RLMs Outperform on OOLONG-Pairs & Role of Recursive Sub-Calling

Ground Truth Checkpoint (Paper)	Classic	PageIdx	Scoring Notes
Base models achieve <0.1% F1 on OOLONG-Pairs (Table 1, p.5)	✗	✓	Classic missed this. PageIdx (step-3.5-flash) now correctly cites <0.1% F1 — a significant improvement over the prior gpt-oss-120b version.
RLM(GPT-5) achieves 58.0% F1; RLM(Qwen3-Coder) 23.1% (Table 1, p.5)	✗	✓	Classic missed this. PageIdx (step-3.5-flash) now cites 58.0% (GPT-5 RLM) and 23.1% (Qwen3-Coder RLM) — previously missed by both systems.
Sub-calls necessary; ablation without sub-calls resorts to keyword heuristics (§4 Obs.2, p.6)	~	✓	Classic mentions decomposition but misses the keyword heuristics contrast. PageIdx names it.
Context rot avoided by processing smaller chunks via sub-LM calls (§E.2, p.27)	✗	✓	Classic never mentions context rot. PageIdx (step-3.5-flash) also no longer surfaces this — both systems miss this mechanistic detail with this model configuration.
RLMs outperform no-sub-call ablation by 10%-59% on all information-dense tasks (§4 Obs.2)	✗	✗	Neither system cited the 10-59% ablation gap — strongest quantified evidence for sub-call contribution.
Sub-calls enable deferring unbounded reasoning chains (§4.1 Emergent Patterns, p.7)	~	✓	Classic says 'stitching outputs' without the unbounded-length reasoning chain concept. PageIdx captures this.
SCORE (✓=1, ~=0.5, ✗=0) out of 6	**1.0/6**	**3.0/6**	**Classic: ~×2, ✗×4 = 1.0 | PageIdx: ✓×3, ✗×3 = 3.0 (step-3.5-flash now cites F1 figures but misses context rot; score unchanged)**

Q4 — Trade-offs: Base LM vs RLM (Performance, Cost, Input Length)

Ground Truth Checkpoint (Paper)	Classic	PageIdx	Scoring Notes
Base LM outperforms RLM on small input lengths — explicit trade-off downside (§4 Obs.3, p.6)	✓	✗	Classic correctly names this downside. PageIdx omitted it entirely — a significant analytical gap.
RLM outperforms base by up to 2× on long contexts; maintains comparable cost (§4 Obs.1, p.5)	~	✓	Classic states advantage without the 2× figure. PageIdx states 'two orders of magnitude' (input length) not performance — both partial; PageIdx closer.
Median RLM run is cheaper than median base model (§4 Obs.4, p.6 — GPT-5 result)	✓	✓	Both correctly state the median cost advantage.
Outlier RLM runs significantly more expensive — high cost variance (§4 Obs.4, p.6)	✓	✓	Both correctly identify high tail-end variance from long trajectories.
RLMs up to 3× cheaper than summary agent (§4 Obs.4, p.6)	✗	✗	Neither system mentioned the comparison to the summarization agent — an important cost benchmark in the paper.
Two orders of magnitude beyond context window — quantified input length capability (Abstract)	✓	✓	Both correctly state this figure.
SCORE (✓=1, ~=0.5, ✗=0) out of 6	**3.5/6**	**3.0/6**	**Classic: ✓×4, ~×1, ✗×2 = 3.5 | PageIdx: ✓×3, ~×1, ✗×2 = 3.5 (Closest question — Classic wins by small-input insight)**

Q5 — Fine-tuning Qwen3-8B as an RLM & Implications for Recursive Training

Ground Truth Checkpoint (Paper)	Classic	PageIdx	Scoring Notes
28.3% improvement on average across four long-context tasks (Abstract + §4 Obs.6)	✓	✓	Both correctly state the 28.3% figure. PageIdx adds 'across four tasks' scope.
RLM-Qwen3-8B approaches GPT-5 quality on three long-context tasks (Abstract)	✗	✗	Neither system mentioned this — a striking capability claim from the abstract.

Training used only 1,000 filtered trajectories from LongBenchPro (§A, p.13)	✗	✓	Classic omits the 1,000 sample count. PageIdx correctly identifies the small dataset size and source.
Key insight: root model learning is the bottleneck; leaf sub-calls are general-purpose (§A, p.13)	~	✓	Classic mentions 'REPL manipulation and recursive calls' generically. PageIdx cites the root/leaf asymmetry insight precisely.
Training generalizes to unrelated tasks beyond the training domain (§4 Obs.6, p.7)	✓	✓	Both correctly identify cross-domain generalization. Classic uniquely names 'unrelated tasks.'
Inference cost drops due to better decision-making and fewer mistakes (§4 Obs.6, p.7)	✓	✓	Both correctly state this cost reduction effect.
SCORE (✓=1, ~=0.5, ✗=0) out of 6	**3.5/6**	**5.0/6**	**Classic: ✓×3, ~×1, ✗×2 = 3.5 | PageIdx: ✓×5, ~×0, ✗×1 = 5.0 (Both missed GPT-5 comparison)**

Final Scorecard

Question	Q1 (6pts)	Q2 (5pts)	Q3 (6pts)	Q4 (6pts)	Q5 (6pts)	Total (29pts)	Verdict
Classic RAG	2.0	2.5	1.0	3.5	3.5	12.5/28	43% — Adequate on multi-point analytical Qs (Q4, Q5) but weak on precise mechanistic detail and experimental figures
Page Index RAG	4.5	5.0	3.0	3.0	5.0	20.5/28	71% — Strong on mechanism and structure-dependent Qs (Q1, Q2, Q5). Loses on experimental figures (Q3) and small-input nuance (Q4).

Shared Blind Spots — What Both Systems Missed

Q	Missed Checkpoint	Why It Matters
Q3	**Base models score <0.1% F1 on OOLONG-Pairs (Table 1)**	Classic RAG did not cite Table 1 figures. PageIdx (step-3.5-flash) now correctly surfaces the <0.1% base model F1 and 58.0%/23.1% RLM scores — resolving this as a shared blind spot.
Q3	**Ablation gap of 10-59% across info-dense tasks (§4 Obs.2)**	This directly quantifies the sub-calling contribution — the exact answer to 'what role does sub-calling play?' — and neither system mentioned it.
Q4	**RLMs 3× cheaper than summarization agent (§4 Obs.4)**	The comparison to the summary agent is the main cost benchmark in the paper. Both systems compared RLM to base LM only, ignoring this richer comparison.
Q5	**RLM-Qwen3-8B approaches GPT-5 quality on 3 tasks (Abstract)**	This is the boldest claim of the fine-tuning section — an 8B model approaching a frontier model. Both systems focused on the percentage gain and missed this claim entirely.