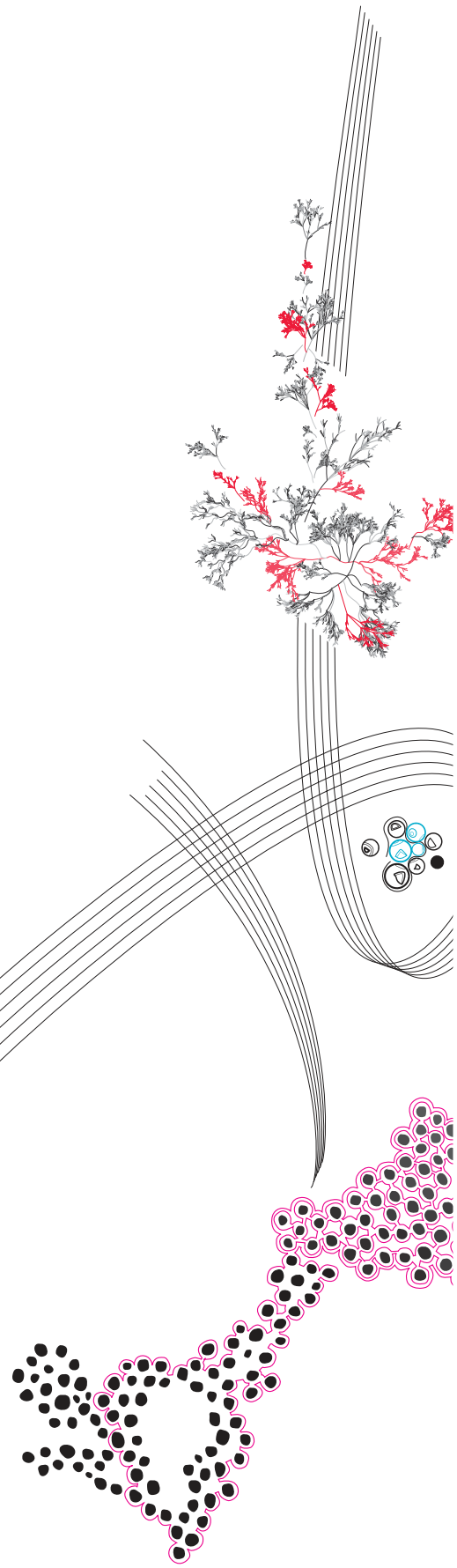

MSc Computer Science
Master Thesis

Does Document Structure Still Pay? A Controlled Benchmark for Repeated-Context Long-Document Question Answering

Beau Jonkhout

Graduation committee:
Tiago Prince Sales (supervisor)
Patricia Ferreira da Silva
Mariet Theune

August, 2026

Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Contents

Abstract

Modern long-context large language models (LLMs) can read an entire document in one prompt, and cache-aware serving makes re-reading an unchanged document far cheaper than reading it cold. This unsettles the premise behind retrieval and document-structuring methods—chunk-based RAG, tree-based summarization such as RAPTOR, and graph-based community summarization such as GraphRAG—which were built for small context windows, when fitting a long document into one prompt was impossible and selecting or summarizing the relevant parts was the only option. This thesis asks whether document structuring still pays once the model can read the whole document, in the *repeated-context* setting where many questions arrive over one fixed document. Its contribution is methodological: a controlled benchmark that evaluates four QA approaches—flat full-context prompting (with cache-aware serving), Naive RAG, RAPTOR, and GraphRAG—on the same documents, questions, answerer, and metrics. One cost-accounting rule prices each method's one-time preprocessing and its per-query answering together, so approaches whose costs fall at different points in the pipeline meet on one cost-quality plane. We instantiate the benchmark on QASPER and NovelQA across 2,336 questions, each answered five times by a single fixed answerer, with NovelQA scored against held-out gold answers and a closed-book control separating reading from memorization. The full run record and analysis code are released, and every reported number, table, and figure regenerates from them. The ranking is identical on both datasets. Flat full-context prompting reaches the highest answer quality: significantly so on the novels under a document-clustered bootstrap, which resamples whole documents, and narrowly over Naive RAG on the short papers. The cost-quality frontier collapses to two points—flat full-context prompting for quality, Naive RAG for budget—and the added tree and graph structure of RAPTOR and GraphRAG improves neither answer quality nor cost-quality position. With long-context models, added document structure does not pay in this setting.

1 Introduction

Consider a researcher who asks ten different questions about the same 50-page paper, or an analyst who repeatedly queries a 300-page regulatory document. In both cases, the underlying text stays the same across questions. Yet current large language model (LLM) systems typically process each question independently, reprocessing the document for each query even though it has not changed.

This pattern—multiple questions over the same long document—appears in several existing benchmarks: QASPER [14] pairs questions with the same research paper, and NovelQA [51] asks questions about the same novel. We call this a *repeated-context* setting: the context is fixed, and multiple queries arrive over time. These documents are long: in this thesis they range from a research paper of a few thousand tokens to a full novel of over 700,000 tokens.

When the entire document is concatenated with each query and sent to the model as one prompt, we call this *flat full-context prompting*. This approach is simple, but it raises two distinct problems—one of quality and one of cost.

The quality problem is that long inputs degrade reasoning: long-context models overlook relevant evidence, underweight the middle of the input, and lose accuracy from length alone [38, 33, 21].

The cost problem is that flat full-context prompting reprocesses the entire document on every query, even though the document is unchanged between questions. Modern serving stacks soften this through *cache-aware serving*: the runtime stores its reading of a shared document prefix and reuses it on later queries [29], so reading less text per query saves less than its token reduction suggests.

These two problems—degraded reasoning on long inputs, and the cost of reprocessing them—are what document structuring was introduced to address. Such methods preprocess each document once into a reusable structure—plain chunks for Naive RAG [30, 18], a summary tree for RAPTOR [46], an entity graph with community summaries for GraphRAG [17]—and use only the relevant part of it at query time, so each query reads less text and reuses work across questions. These approaches have not been compared against each other—or against cache-aware flat full-context prompting—under one cost accounting for the repeated-context setting.

These methods were developed when context-window capacity was small—typically a few thousand tokens, far short of a long document. When a long document could not fit in a single prompt at all, selecting or summarizing the relevant parts was mandatory, and chunk-based RAG reported gains over a no-retrieval baseline, the model answering from memory alone [30], while tree-based summarization such as RAPTOR reported gains over reading truncated or unstructured input [46]. Long-context models now accept hundreds of thousands of tokens [22, 28], and some up to roughly two million [53]. That capacity weakens both of structuring's original rationales at once: such models read long inputs far more reliably than earlier models did, shrinking the quality penalty for length; and they make re-reading cheap, shrinking the cost penalty for reprocessing. What does not shrink is structuring's own side effect: selecting or summarizing a document discards information that the full text retains. This reframes the question that motivated this line of work—once the document fits the window and re-reading it is cheap, does structuring still help, or does the information it discards now cost more than the length it avoids? This thesis answers that question for the repeated-context setting.

Three research questions structure the evaluation.

- **RQ1:** Which existing QA approaches give the best cost-quality tradeoff in the repeated-context setting?
- **RQ2:** What explains the quality differences between the preprocessing-based approaches (summary tree, entity graph, plain chunks)?
- **RQ3:** At what query density—questions asked per document—does a preprocessing approach pay off against reading the document in full, if it ever does?

The contribution of this thesis is a *controlled benchmark*: four existing QA approaches compared in the repeated-context setting under the same documents, the same questions, and the same answerer model, a single long-context large language model (`gemini-3.1-flash-lite-preview`) fixed across all methods. The benchmark is instantiated on QASPER and NovelQA: 2,336 questions, each answered five times. Making that comparison fair across methods that spend their cost in different places—one-time preprocessing versus per-query answering—and across datasets with different task formats requires one uniform cost-accounting rule: every method is scored on a single cost measure covering both the per-query cost of answering and the one-time cost of preprocessing a document, applied identically across QASPER and NovelQA. Quality is reported with each benchmark's native metric. Together these place every method on one cost-quality plane, and

a break-even analysis tests whether the resulting ordering depends on how many queries each document receives. The released code and run artifacts are recorded in Appendix A.

The methods under test are: flat full-context prompting (with cache-aware serving as the baseline [29]), Naive RAG (the most common practical approach [30, 18]), RAPTOR (tree-based hierarchical retrieval [46]), and GraphRAG (graph-based community summarization [17]).

The remainder of this thesis is structured as follows. Section 2 provides the theoretical background on long-context QA, RAG, and the repeated-context setting. Section 3 describes the benchmark methodology: the tasks, datasets, architectures under test, evaluation protocol, and implementation details. Section 4 reports the benchmark results. Section 5 discusses the results and relates them to existing literature. Section 6 discusses threats to validity. Section 7 concludes.

2 Theoretical Background

2.1 Challenges in Long-Context Question Answering

Modern long-context models can process increasingly large inputs, but more tokens do not by themselves solve *selective reasoning*—attending to the relevant evidence in a long input while ignoring the rest. A context is best treated as *long* not at a fixed token threshold but in the range where a model's *effective* context falls short of its advertised window. In that range, accuracy becomes sensitive to input length and to the position of relevant evidence rather than to the question alone [33, 22]. Several distinct failure modes appear in this setting. Liu et al. show that information placed in the middle of a long context is often underused even when it remains technically available to the model [33]. Retrieval quality also degrades as the input grows, so the same fact is recovered less reliably deep in a long prompt than in a short one [21]. Simple single-fact "needle-in-a-haystack" tests overstate this ability: once the question and its supporting evidence no longer use the same words, or once answering requires tracing several facts, accuracy drops sharply well within the advertised window [22, 38, 28]. Broad long-document benchmarks such as LongBench and LongBench v2 [7, 8] likewise show that long-document reasoning remains difficult across question answering, information seeking, and multi-document tasks.

Existing work establishes that long-document reasoning is hard, but it usually evaluates isolated queries. In the repeated-context setting that this thesis studies (Figure 1), the question becomes whether preprocessing the document once can cut the cost of answering later questions without losing much answer quality.

2.2 Preprocessing-Based Approaches

One strategy for controlling repeated-context cost is to build a reusable artifact once per document and answer later questions from that artifact. The one-time build cost is then spread—*amortized*—over the later queries that reuse it (Figure 1): a chunk index, summary tree, or entity graph is constructed once, and each subsequent question uses the artifact instead of re-reading the whole document. This split between a one-time build and a per-query cost amortized over later queries is the long-standing accounting for indexing in information retrieval and approximate-nearest-neighbour search, where construction cost is reported as an axis distinct from per-query cost [35, 48]. A broad survey of retrieval-augmented generation covers three forms of index for the source text: plain

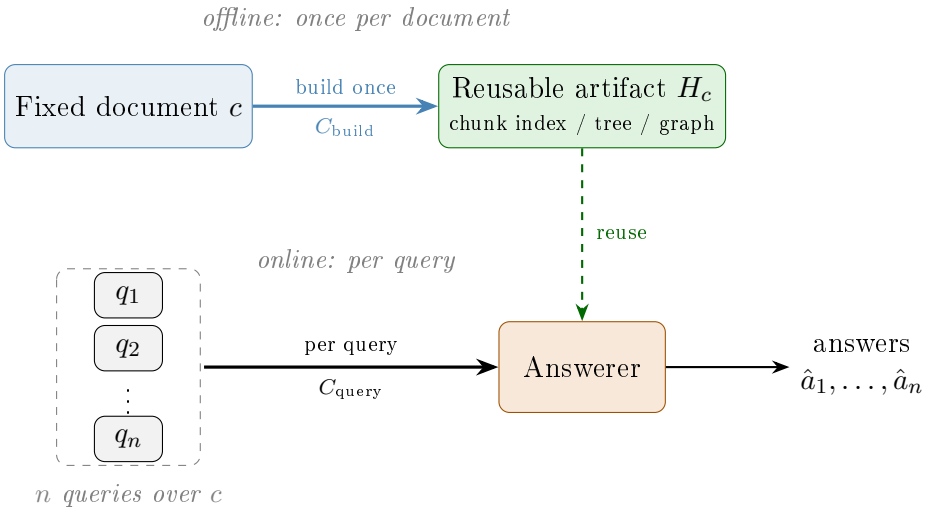

Figure 1: The repeated-context setting. A fixed document c is preprocessed once into a reusable artifact H_c (build cost C_{build}); each of the n later queries over that same document is answered from the artifact (cost C_{query} per query).

chunks, a hierarchy with summaries at its nodes, and a knowledge graph of entities [18]. This thesis benchmarks all three, each through a widely used representative: Naive RAG for chunks, RAPTOR for trees, and GraphRAG for graphs.

Tree-based methods summarize document chunks into a hierarchy, so that evidence can be retrieved at multiple abstraction levels [46, 49]. *Graph-based* methods build a graph of the document's entities and summarize groups of related entities for reuse at query time [17, 43]. *Chunk-based* methods split the document into fixed-size pieces, embed each piece with a dense encoder—a model that maps text to a vector so semantically similar pieces lie close together—and retrieve the top-k matches per query [30]. All three cut the document into pieces before indexing. A common refinement is to choose those pieces from the document's own structure—placing chunk boundaries at sections, paragraphs, and rhetorical units rather than at fixed token offsets—so that each indexed piece stays internally coherent [39, 15, 24]. Other designs vary the indexing rather than the document structure—for example indexing single factual statements (propositions) or linked memory entries [12, 19]—and so fall outside this structural comparison.

Where these preprocessing-based methods pay their cost once, up front, a second class instead addresses long-context cost through *per-query recursive control*—inspecting, decomposing, or critiquing the context at query time [61, 52, 1, 45, 63, 5]. These methods pay their full cost on every question rather than amortizing a one-time structure across queries, so they fall outside the preprocessing-based comparison studied here.

A third class controls cost through *routing and memory mechanisms* that select retrieval behavior, reasoning depth, or memory tier per query under cost constraints [23, 62, 41]. These too adapt cost per query rather than amortizing a one-time structure. Sleep-time compute instead amortizes offline precomputation across related queries about one context, but builds no retrieval structure to compare [32].

2.3 Cache-Aware Serving

Modern inference runtimes store the intermediate attention computations—the key-value (KV) cache—for a shared document prefix and reuse them on later queries that share it:

a *warm prefix cache*. PagedAttention, a memory manager for that cache, makes the reuse efficient [29], and vLLM is a production serving system built on it [50]. For repeated-context QA, cache-aware serving shifts the cost of the simplest baseline: a flat full-context prompt that hits a warm prefix cache costs much less than the same prompt processed cold. Any preprocessing-based method must therefore be compared against this cache-aware baseline rather than an uncached one; otherwise the method looks cheaper than it is, because part of the saving would have come from the cache anyway.

2.4 Reproducibility of LLM Inference

Even under greedy decoding—sampling temperature $T = 0$, where the model always takes its highest-probability token—large language model inference is not reproducible end-to-end, a property that directly shapes how evaluation results must be reported. Atil et al. evaluate five LLMs across eight tasks at $T = 0$ over ten runs and report accuracy variance up to 15% and best-vs-worst gaps up to 70%, with raw outputs rarely bitwise-identical [6].

The root cause is that floating-point addition is not associative, so the order in which a hardware kernel accumulates a sum changes its result in the last bits [60]. He et al. show that this makes inference *batch-variant*: the same prompt produces different logits depending on which other requests it is co-batched with [20]. At provider scale, serving-stack choices such as load balancing and dynamic batching compound these effects [34], and at the decoding step the resulting drift can flip the selected token.

This remaining randomness means a single run per configuration cannot be treated as a point estimate of an architecture's quality.

2.5 Long-Document Evaluation Benchmarks

The benchmark choice must match the evaluation question. This study evaluates QA over long *documents*—real, structured texts such as papers and novels—under the repeated-context requirement that each document carry many questions, so that preprocessing cost can be amortized across queries. Table 1 weighs the candidate benchmarks against both criteria. They split into synthetic long-*context* benchmarks, which lack document structure and (for the probe benchmarks) any question reuse, and long-*document* benchmarks built from real texts. Several real-document benchmarks meet both requirements; QASPER and NovelQA are the pair evaluated here (Table 1).

The pair sits at opposite ends of two axes—context length and gold format—so the thesis evaluates both datasets and reports each with its native metric, Answer-F1 for QASPER and accuracy for NovelQA. NarrativeQA meets both requirements, but its long narratives are the role NovelQA already fills, and its free-form answers are the format QASPER already covers, so it adds no new point on either axis. The remaining candidates measure something else or add no new point. Oolong is a long-*context* benchmark: its tasks count and filter facts spread across the context, over constructed and real conversational inputs, not QA over a structured document. NoCha controls *contamination*—its novels are recent enough not to sit in any model's training data—but it scores claims at the pair level (both true/false variants of a claim must be judged correctly), so it measures something different from per-question QA accuracy, and its copyrighted novels must be sourced separately rather than shipping with the benchmark; and QuALITY would mainly isolate length effects from format effects, which QASPER and NovelQA

Table 1: Long-context and long-document benchmarks weighed against this study's two requirements: real document structure (not a synthetically generated context) and many questions per document (so preprocessing amortizes). Sizes and question counts are approximate.

Benchmark	Task and reuse	Role here
Probe benchmarks	synthetic single-fact retrieval, a fresh context per probe (needle-in-a-haystack [25], RULER [22], NoLiMa [38]); no document structure, no query reuse	out of scope: long-context, not a document
QASPER [14]	free-form QA over research papers; ∼4 questions/paper; ∼5k tokens; Answer-F1	**evaluated** (short, free-form)
NovelQA [51]	multiple-choice QA over novels; ∼25 questions/novel; up to ∼770k tokens; accuracy	**evaluated** (long, multiple choice)
Oolong [10]	long-*context* aggregation—counting/filtering over constructed and conversational contexts, no document structure; 25 questions/context	out of scope: long-context, not a structured document
Narrative-QA [27]	free-form QA over books and film scripts; ∼30 questions/document	deferred: length and format already spanned by QASPER and NovelQA
NoCha [26]	global comprehension via minimal-pair claims (paired true/false variants differing in one detail); many claims per recently published, contamination-controlled novel	deferred: pair-scored claims; texts sourced separately
QuALITY [42]	multiple-choice QA over short (∼5k-token) documents; multiple questions per document	future: length vs. format

already bracket.

Care is needed when reading any static benchmark: scores can be distorted by contamination and by a benchmark measuring something other than the ability it claims to measure [54, 9]. Long-context benchmarks also differ enough in task structure, gold-evidence format, and context scale that results do not transfer directly from one to another. Public leaderboards such as LM Arena [3], which ranks models by aggregating pairwise human votes [13], are useful for a first orientation, but a high position can reflect answer style [4], sensitivity to prompt phrasing [2], or contamination [56] rather than performance in the repeated-context setting.

3 Methods & Materials

The benchmark preprocesses each document once and answers many questions over it, comparing the four architectures under one fixed answerer, one shared encoder for the preprocessing-based architectures, one prompt template per task format, and a single cost-accounting rule, so that observed differences reflect the context-handling strategy rather than the experimental setup. The protocol was fixed before the experiments and is recorded in Appendix A.

3.1 Tasks

The benchmark asks each architecture to answer natural-language questions about a long document, under the repeated-context setting in which many questions share one preprocessed document (Figure 1).

The task type is long-document question answering: the system reads one long source document and answers questions about it, rather than retrieving answers from an open collection of many documents (Section 2). The two evaluation datasets deliberately span a range of question demands. QASPER labels each question by its answer type: *extractive* (copy a span from the paper—for example, which corpus a model is trained on), *abstractive* (compose the answer in new words—for example, how an ablation was set up), *yes/no* (for example, whether a baseline is compared), and explicitly *unanswerable* questions, asked but not answered by the paper [44, 14]. NovelQA labels each question by aspect (character, setting, relation, plot, paraphrase, counting, and span—questions needing the exact wording of the text; the released *meaning* and *times* aspects are displayed as *paraphrase* and *counting* throughout) and by complexity (single-hop, multi-hop that combines evidence from several passages [57], and detail: fine-grained fact lookup) [51]. This range is why answer quality is measured with each format's own metric and why results are not treated as interchangeable across datasets.

Two task formats are covered, and they drive different answer-quality metrics:

- **Free-form short answers** (QASPER): the architecture produces a short text answer that is compared against the gold answer text. Evaluated with Answer-F1: the token-overlap F1 between predicted and gold answer—the harmonic mean of shared-token precision and recall, taking the best match over the annotators' gold answers [14].
- **Multiple choice** (NovelQA): the architecture selects one of a fixed set of options. Evaluated with accuracy: the fraction of questions whose selected option matches the gold option.

3.2 Datasets

The benchmark uses two datasets, each selected for a distinct role:

- **QASPER** [14]: research papers averaging $\approx$ 5k tokens, with multiple questions per paper. Its gold answers ship with the dataset.
- **NovelQA** [51, 40]: novels averaging $\approx$ 230k tokens, up to $\sim$770k (*Les Misérables*), with multiple questions per novel. Its test-split gold labels are kept private on Codabench (the platform hosting the benchmark's official scoring). The dataset also releases manually annotated evidence excerpts, which this study does not use for scoring. One novel (NovelQA's internal ID B48, *The History of Rome*, $\sim$2.6M tokens) is excluded from the main evaluation as a context-length outlier, and a second (*Frankenstein*) because its gold answers could not be obtained from Codabench.

Evaluation is document-centered: within each document, all associated benchmark questions are treated as repeated queries and amortized jointly. Eligible documents must have at least two associated questions, so that amortization over repeated queries is well defined. A small calibration split is set aside before evaluation: four NovelQA novels

Table 2: Dataset roles for the repeated-context study: the document unit, repeated-context fit, task metric, and role in the main claim for each dataset.

Dataset	Document Unit	Repeated-Context Fit	Task Metric	Role in Main Claim
QASPER	Paper	Direct fit: paper-level questions align with repeated-context reuse and native document structure	Answer-F1	Locally scored quality comparison on short documents; the short-document contrast for the granularity analysis.
NovelQA	Novel	Direct fit: multiple questions share one very long narrative source context	Accuracy (multiple choice)	Quality under held-out scoring on long documents, where build costs are large enough that repaying them takes many questions.

and 20 QASPER questions, used only to select the models and settings of Section 3.5.2 (the pilot grid drew questions from three of the four novels). The four novels and all calibration questions are excluded from the scored pools, so calibration data cannot leak into the test metric. Table 2 summarizes each dataset's role and metric.

3.3 Architectures

The four architectures differ in what they build from the document, if anything, and in what context they pass to the answerer at query time. Figure 2 contrasts them side by side.

3.3.1 Flat Full-Context Prompting

The simplest approach concatenates the entire document with each query and sends it to the answerer as a single prompt. It involves no preprocessing and no retrieval, so flat full-context prompting (hereafter *Flat*) is the no-retrieval baseline against which the three preprocessing-based architectures are measured: it builds nothing, and every query pays for reading the entire document. Cache-aware serving (Section 2.3) makes repeated reads of the same document prefix much cheaper, so the benchmark tests the cache-aware variant of this baseline. The cache resets for each new document; the first query reads the document uncached, and the repeated queries after it reuse the stored prefix.

3.3.2 Naive RAG (Chunk-Based Retrieval)

Naive RAG splits the document into chunks of a few hundred tokens (this study: sentence-boundary chunks of 384 tokens; Table A1) [47, 11], embeds each chunk with a dense encoder, and stores the embeddings in a vector index—a data structure that supports fast nearest-neighbour lookup over those embeddings. At query time, the query is embedded with the same encoder and the top-k most similar chunks are retrieved and passed to the answerer.

Its build cost covers chunking and embedding the document; its query cost covers embedding the query, the similarity search, and the answerer call on the retrieved chunks (Section 3.4.1).

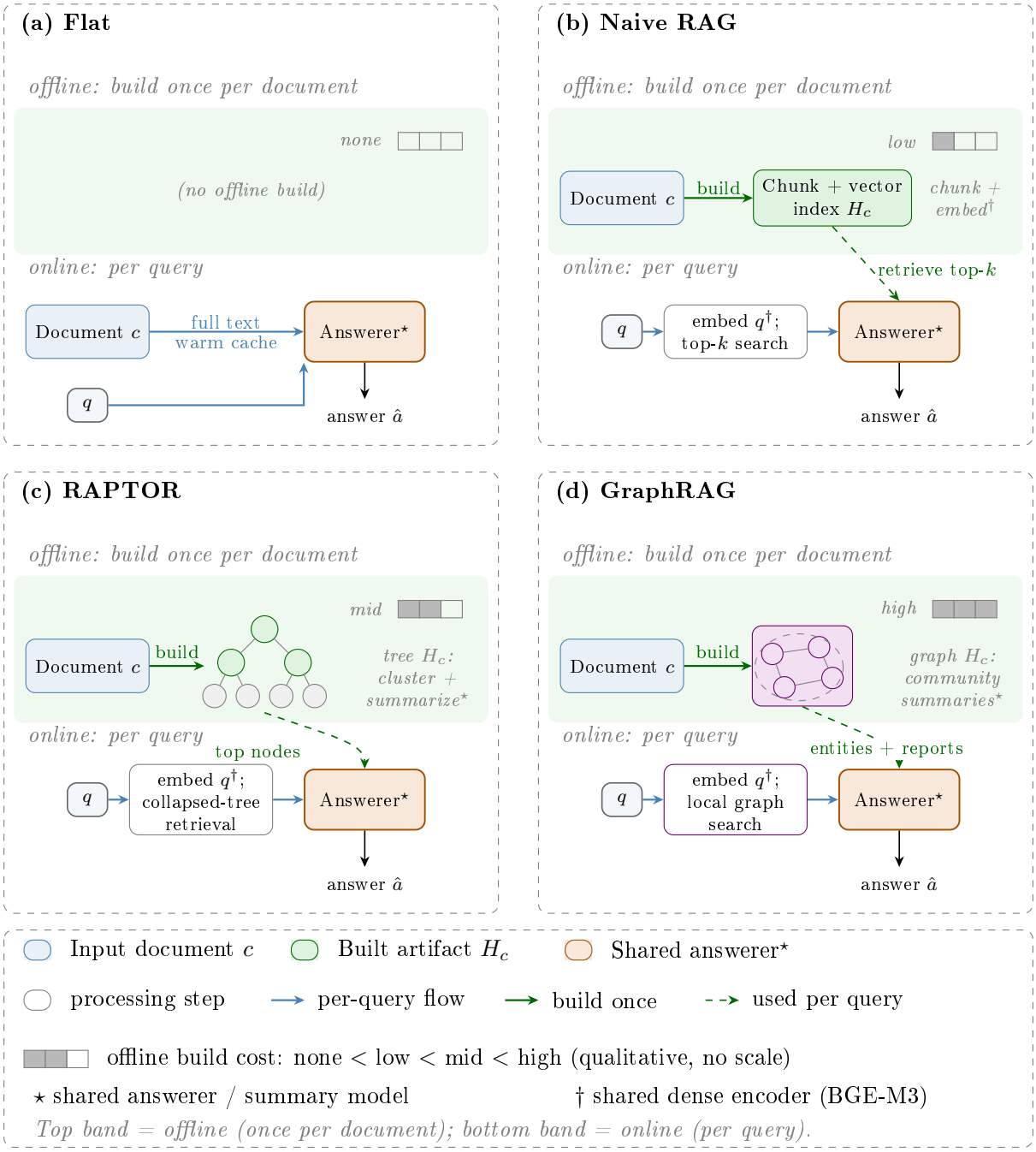

Figure 2: The four context-handling architectures compared in this study, each split into a build-once stage (top band) and a per-query stage (bottom band). Flat builds no artifact and re-reads c on every query; Naive RAG, RAPTOR, and GraphRAG each build a persistent artifact H_c that the per-query path uses. ⋆ marks the shared answerer and summary model (`gemini-3.1-flash-lite-preview`) and † the shared dense encoder (BGE-M3).

3.3.3 RAPTOR (Tree-Based Retrieval)

RAPTOR [46] recursively clusters and summarizes document chunks to build a hierarchical tree. Leaf nodes contain the original text chunks; internal nodes contain abstractive summaries of their children.

At query time, the query is embedded with the same dense encoder used for the tree nodes; no query rewriting, expansion, or decomposition is applied. Following RAPTOR's recommended default—the *collapsed tree* retrieval strategy—cosine-similarity search is performed over all tree nodes (leaves and internal summaries) simultaneously, and top-ranked nodes are concatenated into the retrieved context up to a fixed token budget, without any reranking step. RAPTOR also supports an alternative *tree-traversal* strategy that searches level-by-level; this benchmark uses the collapsed-tree default that the original paper reports as consistently stronger.

Its build cost covers chunking, recursive clustering, and summary generation at each level of the tree; its query cost covers embedding the query, searching over all tree nodes, and the answerer call on the retrieved context.

3.3.4 GraphRAG (Graph-Based Retrieval)

GraphRAG [17] indexes the document as a graph of its entities rather than as retrievable text spans. It uses an LLM to extract entities and relationships from the document, builds a knowledge graph, and applies community detection to partition the graph into groups of closely related entities. Each community is summarized at indexing time. At query time, GraphRAG is run in its *local-search* mode, the mode the reference implementation provides for entity-grounded questions about specific content [36]: the query is embedded with the shared dense encoder, the top-ranked entities and their relationships are selected by similarity, and they are packed—together with their community reports and source text units—into a single fixed-budget context for one answerer call. GraphRAG's alternative *global-search* mode instead answers corpus-wide summarization questions by generating a partial response from every community summary and aggregating them; its map step re-reads the community summaries on each query, a per-query cost profile built for question types these benchmarks do not pose.

Its build cost covers entity extraction, graph construction, community detection, and community summarization; its query cost covers embedding the query, selecting entities and relationships, and the answerer call on the packed graph context.

3.4 Evaluation Protocol

3.4.1 Cost Accounting

Every reported cost comes from one ledger. Each API call records its uncached input tokens, its cached input tokens (which the provider bills at a discount), and its output tokens; each local step (the embedding encoder) records its GPU-seconds; stored artifacts record their size. A price card fixed before the experiments converts each ledger line to dollars: the provider's published API rates for model calls, and a fixed rate for GPU time and storage so that local and API work land in the same currency (exact rates in Appendix A). Costs are reported under two such cards: the *standard card* (published rates; cached input at one-quarter of the uncached rate) and a *cache-discount card* (cached

Table 3: Local notation reference for the repeated-context cost and quality formulation.

Symbol	Definition	Unit / Domain
c	Long source document, such as a paper or novel.	Document instance
q_i	i-th question over document c.	Question instance
n	Number of questions amortized jointly for one document.	Count
N	Number of repeats per (architecture, query) pair ($N = 5$).	Count
H_c	Persistent artifact bundle stored for document c by the chosen preprocessing method (tree, graph, or chunk index with summaries, embeddings, and metadata).	Artifact set
$C_{\text{build}}(c)$	One-time cost of constructing the reusable artifact for document c.	Dollars per document
$C_{\text{store}}(c, H_c)$	Cost of persisting the artifact bundle H_c over the 90-day study horizon (Appendix A).	Dollars per document horizon
$C_{\text{query}}(q_i \mid c)$	Cost of answering question q_i over document c.	Dollars per query
$\bar{C}_{\text{deploy}}(c, n)$	Amortized cost per query for a document that receives n questions.	Dollars per query
$n^\star$	Break-even density (Section 3.4).	Count
$\bar{M}$	Mean quality over the scored questions (benchmark-native metric, normalized).	$[0, 1]$

input at one-tenth), a sensitivity check fixed in advance, because a deeper cache discount is the pricing assumption that most favors full-context re-reading.

Each architecture's dollars split into two parts. The *build cost* C_{build} is paid once per document, before the first question arrives: chunking, embedding, summarizing, or graph extraction. Flat builds nothing, so its $C_{\text{build}} = 0$. The *query cost* C_{query} is paid on every question: embedding the query, any similarity search, and the answerer call on the selected context. For a document c that receives n questions $q_1, \ldots, q_n$, the comparison quantity is the amortized cost per query,

$$\bar{C}_{\text{deploy}}(c, n) = \frac{C_{\text{build}}(c) + \sum_{i=1}^{n} C_{\text{query}}(q_i \mid c)}{n}, \tag{1}$$

which falls as more questions reuse the same build. Storage is excluded from this equation because it grows with time rather than with question count; it is reported separately as a per-architecture footprint. A study-wide deployment total—the same ledger summed over all documents and questions, still excluding storage—is recorded in Appendix A for completeness. Reported deployment costs cover one repeat per query; the run-wide billed total over all five repeats is recorded there as well.

The quality axis is each benchmark's native metric, normalized to $[0, 1]$ so both datasets share one quality scale per panel (Section 3.1). Reported quality, written $\bar{M}$, is the mean over the scored questions. Table 3 collects this notation.

3.4.2 Scoring

Answers are scored against each benchmark's own gold standard. On QASPER, the official Answer-F1 scorer (Section 3.1) runs locally against the released gold answers; on

NovelQA, answer letters are submitted to Codabench and scored against the held-out gold [51]. Per-architecture retrieval settings are set to the cited literature defaults rather than tuned on our data (Table A1), so each architecture runs as its source paper published it.

Statistical reporting. Whether one architecture beats another is decided by a document-clustered *paired* bootstrap: whole documents (papers for QASPER, novels for NovelQA) are resampled with replacement, and every architecture is re-scored on the same resample—which is what makes the comparison paired.

Error-analysis definitions. The error analysis tallies every scored question (the full scored pool). A NovelQA answer is correct when it matches the held-out gold option; a QASPER answer is counted correct at a per-question mean Answer-F1 ≥ 0.5 and wrong at ≤ 0.15, a band fixed in advance so that borderline paraphrases count for neither side, with intermediate scores excluded from the tally. The *losing subset* is the set of questions Flat answers correctly and both summary-based methods (RAPTOR and GraphRAG, the two that answer from summaries) answer wrongly; it is used to count and to read failures, not to compute rates.

Abstention is detected by a fixed list of hedge phrases ("not mentioned", "cannot be determined", and similar; the full list ships with the released analysis code), and a multiple-choice option counts as an "absent" option when its text indicates the item never appears (a keyword rule, released with the analysis code). NovelQA aspect labels are regrouped for display into three granularity groups, and the full ungrouped per-aspect results are reported (Appendix B) so conclusions can be checked under any alternative grouping. Uncertainty for these groups uses the same document-clustered paired bootstrap, reported as 95% confidence intervals, with one shared resample drawn across methods; a difference between two group gaps is recomputed inside every resample.

Break-even reporting. Amortized deployment cost is a continuous function of the questions per document, so break-even is reported as the amortized-cost curve, summarized by the break-even density $n^{\star}$: the smallest number of questions per document at which an architecture's amortized per-query cost falls below cache-aware Flat's per-query cost under the same price card, recovered from the data rather than fixed in advance. Where a curve never crosses, no break-even exists.

Closed-book control. A closed-book control re-asks every question with the document withheld: the model receives only the question, plus the answer options for NovelQA, scored identically. Its floor separates reading from memorization.

3.4.3 Sampling Temperature

The sampling temperature is fixed at $T = 0$, greedy decoding (Section 2.4), for all LLM calls in all four architectures, including preprocessing stages (summarization for RAPTOR; entity extraction and community summarization for GraphRAG) and the final answerer.

This choice has three justifications. First, it is the standard setting in closely related QA work: the RAPTOR reference implementation hard-codes $T = 0$ for its answerer [46]; Self-RAG uses $T = 0.0$ in its reference implementation [5]; Liu et al. explicitly state that

they use greedy decoding when generating outputs [33]; the original RAG paper uses greedy decoding for its QA tasks [30]; and LongBench's reference evaluation script likewise uses greedy decoding [7]. Second, using the same temperature for every architecture means the decoding setting cannot explain any difference between them, so observed differences between architectures are attributable to retrieval and preprocessing structure rather than to decoding hyperparameters. Third, neither the RAPTOR paper [46] nor the GraphRAG paper [17] specifies a temperature for the preprocessing stages; choosing $T = 0$ for those stages follows community reproductions and keeps the built tree or graph stable across constructions.

3.4.4 Number of Repeats

Each (architecture, query) pair is answered $N = 5$ times as a guard against the residual between-repeat variance that greedy decoding reduces but does not eliminate (Section 2.4); N was fixed before the experiments. The repeats are aggregated per question: on QASPER, a question's score is the mean Answer-F1 over the five repeats; on NovelQA, the first repeat's answers are the ones submitted, and correctness is scored once against the held-out gold after checking locally that the repeats agree.

The repeat count balances precision against cost: five repeats keep the between-repeat spread small relative to the quality differences of interest, while staying affordable on NovelQA, where answering a single query over a whole novel is far more expensive for the full-context reader.

The repeats do not decide rankings; that is the job of the paired bootstrap of Section 3.4.2. They serve per-question aggregation and stability checking only.

3.5 Implementation Details

3.5.1 Prompting Setup

Each architecture uses a fixed prompt template per task format (free-form short answers vs. multiple choice). The answer prompt is identical across all four architectures; only the `{context}` slot differs (the full document, retrieved chunks, tree nodes, or the packed graph context). The summary and extraction prompts that RAPTOR and GraphRAG use during preprocessing are those methods' own defaults (Appendix A).

Both templates instruct the answerer to use the provided context and to commit to an answer rather than abstain, because hedging is scored as just another answer: NovelQA's multiple choice offers no abstain option, and on QASPER a hedge scores only against the minority of questions whose gold answer is itself "unanswerable". The free-form template is:

```
Answer the following question based on the provided context.  Be concise
and specific; for yes/no questions reply with one word.

Context:  {context}

Question:  {query}

Answer:
```

The multiple-choice template adds the option list verbatim and requires a single option label even on incomplete context:

Table 4: Model-selection funnel: a twenty-two-candidate set narrowed to one shared answerer. The pilot grid is the 9-candidate × 4-architecture evaluation. Per-candidate outcomes appear in Table A4.

Screen	Criterion	Remaining
Initial set	eight vendor families	22
Provider screen	prompt-caching availability, context-window fit	17
Smoke testing	$T = 0$ / Chat-Completions API support, account-tier rate limits, context-window fit, serving cost and reliability under load	9
Final selection	cheapest closed candidate among the grid survivors; rank stability checked on the pilot grid (Appendix A)	1

```
Answer the multiple-choice question using the provided context.  Always
choose the single most likely option, even if the context is incomplete
or does not fully contain the answer.

Context:  {context}

Question:  {query}

Options:  {options}

Respond with only the letter of the correct option.
```

3.5.2 LLMs Used

The fixed configuration of Section 3 has three model roles: the answerer, the summary model that RAPTOR and GraphRAG use during preprocessing, and the embedding encoder. Each is pinned to one checkpoint—a fixed model version—for the whole study and is not tuned per dataset; Table 5 gives the basis for each choice. Selection ran as a funnel (Table 4) that narrowed a twenty-two-candidate set spanning eight vendors—listed per candidate in Table A4—to one. A provider-level screen and a smoke test removed candidates that could not serve all four architectures at $T = 0$ under the repeated-context interface. A pilot then ranked the nine survivors, across three vendors (Google, xAI, DeepSeek), on a calibration pool of 15 NovelQA questions (from three of the four held-out calibration novels) and 20 QASPER questions, scored through each benchmark's own gold route (Section 3.4.2) on the four architectures.

The selection rule is the cheapest closed-weights (API-served) candidate among the pilot-grid survivors; before one answerer was fixed, rank stability across the surviving candidates was checked with Kendall's τ_b—a rank-correlation coefficient in $[-1, 1]$—on each candidate's four-architecture score vector (Appendix A).

Concrete model checkpoints and hyperparameter values are recorded in Appendix A.

4 Results

4.1 Scored Pool and Exclusions

On NovelQA, the scored pool is 1,381 multiple-choice questions over 55 novels: what remains after the exclusions fixed in Section 3.2 (the four calibration novels, *Frankenstein*, and B48). On QASPER, all 249 research papers are scored—no document is excluded, and only the held-out calibration questions leave the pool (Appendix A)—giving 955 free-form questions. The evaluation therefore covers 2,336 distinct questions, scored as

Table 5: The three fixed model roles and the basis for each choice.

Role	Model	Basis
Answerer	`gemini-3.1-flash-lite-preview` (Google)	selected by the funnel of Table 4; acceptable serving latency at 600k-token inputs
Summary model	`gemini-3.1-flash-lite-preview` (same checkpoint)	reused for the RAPTOR and GraphRAG preprocessing stages; pinned identically so built artifacts stay reusable across repeats and any later cross-vendor comparison sees the same preprocessing
Embedding encoder	`BAAI/bge-m3` (served locally via Ollama, a local model server)	shared dense encoder for the preprocessing-based architectures; its top-20 gold-evidence retrieval (Recall@20: the share of questions whose gold evidence appears among the top 20 retrieved chunks) cleared the ≥ 0.85 threshold fixed in advance on the QASPER calibration pool

specified in Section 3.4. The five repeats returned identical NovelQA answers on 99.7% or more of questions for every architecture; QASPER needs no such identity check, because all five of its repeats are scored and averaged (Section 3.4.4); its repeats are nearly as stable, with a mean within-question spread across repeats of at most 0.011 Answer-F1 for any architecture.

4.2 Per-Architecture Answer Quality

The answer-quality ranking is the same on both datasets: Flat first, Naive RAG and RAPTOR in the middle, and GraphRAG last (Figure 3). On QASPER, mean Answer-F1 is Flat 0.457, Naive RAG 0.444, RAPTOR 0.413, and GraphRAG 0.403; on NovelQA, mean accuracy is Flat 0.757, Naive RAG 0.654, RAPTOR 0.646, and GraphRAG 0.600.

Table 6 lists each pairwise difference with its clustered-bootstrap confidence interval and Holm-adjusted p (the Holm correction tightens each test's threshold so that the six tests per dataset jointly keep the 5% error rate); a pair is significant under the *interval rule* when its 95% interval excludes zero. Flat scores highest on both datasets. On NovelQA it beats every other architecture, both by the interval rule and after the Holm correction. On QASPER it beats RAPTOR and GraphRAG by both rules too; its lead over Naive RAG is smaller ($\Delta = 0.013$, $p = 0.040$), and holds by the interval rule but not after the Holm correction. On NovelQA, Naive RAG and RAPTOR are statistically tied ($\Delta = 0.008$; the confidence interval includes zero). GraphRAG trails every architecture significantly on NovelQA; on QASPER it is significantly below Flat and Naive RAG and statistically tied with RAPTOR.

This pattern holds across a 46× change in mean document length ($\approx$ 5k-token papers versus $\approx$ 230k-token novels).

4.3 Error Analysis: Where Structure Loses

To explain the ranking (RQ2) we ask not only how much each method trails Flat but on which questions, using each benchmark's own question labels and reading the failing answers. On NovelQA, 140 of the 1,381 scored questions are answered correctly by Flat but missed by both summary-based methods (RAPTOR and GraphRAG)—the losing subset of Section 3.4. On 47 of these, Naive RAG—which retrieves plain chunks without

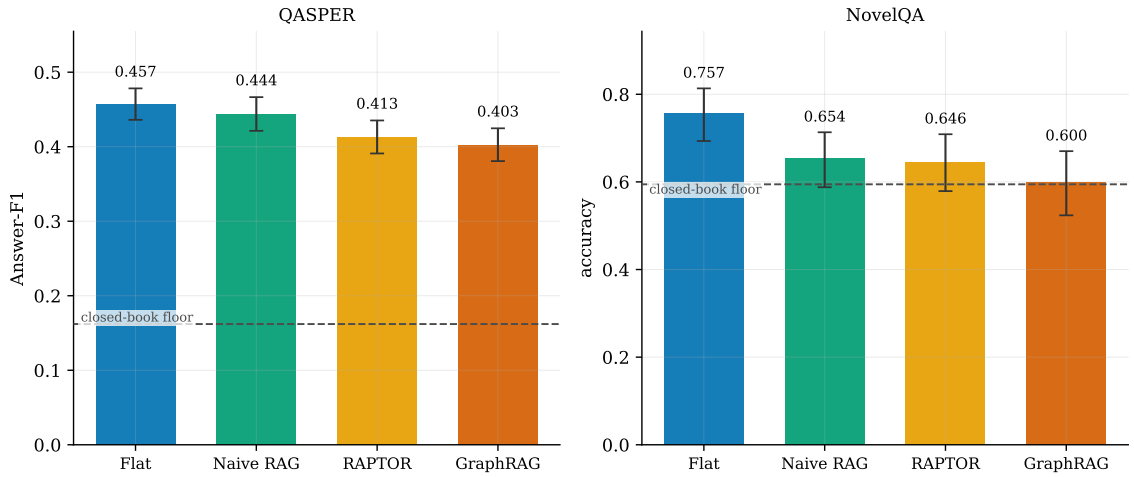

Figure 3: Per-architecture answer quality on each dataset. Error bars are 95% confidence intervals from the document-clustered paired bootstrap (clusters are papers for QASPER and novels for NovelQA); the repeats do not enter the intervals. The dashed line is the closed-book floor: the same questions answered without the document (Section 4.5).

summarizing—is also correct: for that subset, plain chunk retrieval preserves the evidence the question needs (Table 7); on the remainder, plain chunk retrieval misses the answer too.

Figure 4 groups the scored NovelQA questions by information demand, from gist (384 questions) through mid-level (371) to fine-grained detail (626). The grouping orders NovelQA's aspect labels by the access the answer needs: plot and relational questions ask for narrative structure a summary is designed to preserve (gist); character and setting questions target localized facts a summary typically keeps (mid-level); paraphrase, counting, and span questions need exact wording or exhaustive enumeration over the full text (fine-grained). The full per-aspect results are in Appendix B. The Flat-to-worst gap widens with information demand: 0.08 on gist questions, 0.16 at mid granularity, and 0.20 on fine-grained questions that need verbatim wording or exhaustive counting. Both steps away from gist are significant (mid − gist +0.08, 95% CI [+0.02, +0.13]; detail − gist +0.12, CI [+0.04, +0.17]). The last step is not: mid and detail are statistically indistinguishable (detail − mid +0.04, CI [-0.03, +0.10]).

The same pattern appears under NovelQA's own complexity labels, with no regrouping at all: the gap is 0.11 on single-hop questions, 0.16 on multi-hop, and 0.22 on the benchmark's detail class. The detail-to-single-hop difference is +0.12 (CI [+0.05, +0.17]); Table A6 gives the full breakdown.

Presence-and-count questions ask whether, or how often, something occurs in the novel. The relevant subset is the 63 such questions that offer a "never appears" option, whose gold answer is a present option, and that Flat answers correctly. On this subset RAPTOR picks the absent option 35 times and GraphRAG 37, versus 24 for Naive RAG.

On QASPER the same gap stays small across answer types (at most 0.105), with the largest gap on yes/no questions rather than on the extractive questions closest to verbatim access (Appendix B). The residual QASPER loss falls on document-wide judgments, where the preprocessing-based architectures sometimes abstain on questions the full text settles (of the 413 questions Flat answers correctly and concretely, GraphRAG abstains on 21, Naive RAG on 15, and RAPTOR on 13).

Table 6: Paired clustered-bootstrap significance (10,000 resamples; percentile 95% CI on the mean difference; two-sided bootstrap p with the Holm–Bonferroni adjusted p_{Holm} over the six tests per dataset; clusters are papers for QASPER and novels for NovelQA). A pair whose 95% CI includes zero is statistically tied. † significant by the interval rule but not after the Holm correction.

Pair	Δ mean	95% CI of Δ	p	p_{Holm}	Verdict
QASPER (Answer-F1)					
Flat vs Naive RAG	+0.013	[+0.001, +0.026]	0.040	0.081	significant†
Flat vs RAPTOR	+0.044	[+0.029, +0.060]	< 0.001	< 0.001	significant
Flat vs GraphRAG	+0.055	[+0.039, +0.071]	< 0.001	< 0.001	significant
Naive RAG vs RAPTOR	+0.031	[+0.017, +0.045]	< 0.001	< 0.001	significant
Naive RAG vs GraphRAG	+0.042	[+0.025, +0.058]	< 0.001	< 0.001	significant
RAPTOR vs GraphRAG	+0.011	[-0.007, +0.028]	0.238	0.238	tied
NovelQA (accuracy)					
Flat vs Naive RAG	+0.104	[+0.082, +0.125]	< 0.001	< 0.001	significant
Flat vs RAPTOR	+0.112	[+0.085, +0.139]	< 0.001	< 0.001	significant
Flat vs GraphRAG	+0.158	[+0.127, +0.190]	< 0.001	< 0.001	significant
Naive RAG vs RAPTOR	+0.008	[-0.018, +0.036]	0.532	0.532	tied
Naive RAG vs GraphRAG	+0.054	[+0.027, +0.085]	< 0.001	< 0.001	significant
RAPTOR vs GraphRAG	+0.046	[+0.017, +0.078]	0.001	0.002	significant

4.4 Cost, the Pareto Frontier, and Break-even

Deployment cost is accounted per architecture under the cost rule of Section 3.4.1 and reported under the two price cards fixed there; under either card the cache discount reaches only Flat, the one architecture that re-reads a cached document prefix. Table 8 gives the per-architecture cost *within each dataset*: a novel costs far more to build and to read than a paper, so a single pooled total is dominated by the novels and hides how the ranking differs between datasets.

The cost ranking does not follow the quality ranking, and it differs by dataset (Table 8). Naive RAG is cheapest and GraphRAG most expensive on both; in between, Flat and RAPTOR *swap*. On the short QASPER papers Flat is cheaper ($0.38 vs. RAPTOR's $0.60, summed over all 249 papers), because a paper is cheap enough to re-read that RAPTOR's one-time tree-summarization build does not repay at the observed query density; on the long NovelQA novels RAPTOR is cheaper ($6.97 vs. Flat's $10.52, summed over all 55 novels), because Flat re-reads the whole novel on every query. The pooled ranking (Naive RAG < RAPTOR < Flat < GraphRAG) is the novels' ranking and does not hold on the papers. Persistent-artifact storage stays a minor term throughout—at most 2.5% of an architecture's own total spend, for RAPTOR (Table A2).

Placing each architecture on its dataset's cost-quality plane (Figure 5) collapses the design space, on *both* datasets, to the same two-point *Pareto frontier*—the choices that no alternative beats on both cost and quality: *Flat*, the quality-maximizing choice, and *Naive RAG*, the budget choice. RAPTOR and GraphRAG are both *dominated*: another architecture is better on one axis and no worse on the other. Naive RAG matches or beats RAPTOR's quality (Table 6) at lower cost on both datasets, far lower on the novels (Table 8), so RAPTOR is off the frontier. GraphRAG, the most expensive architecture on both datasets, sits at the quality bottom (Section 4.2) and is dominated on *both* axes. Adding tree or graph structure therefore buys neither quality nor cost position in this setting.

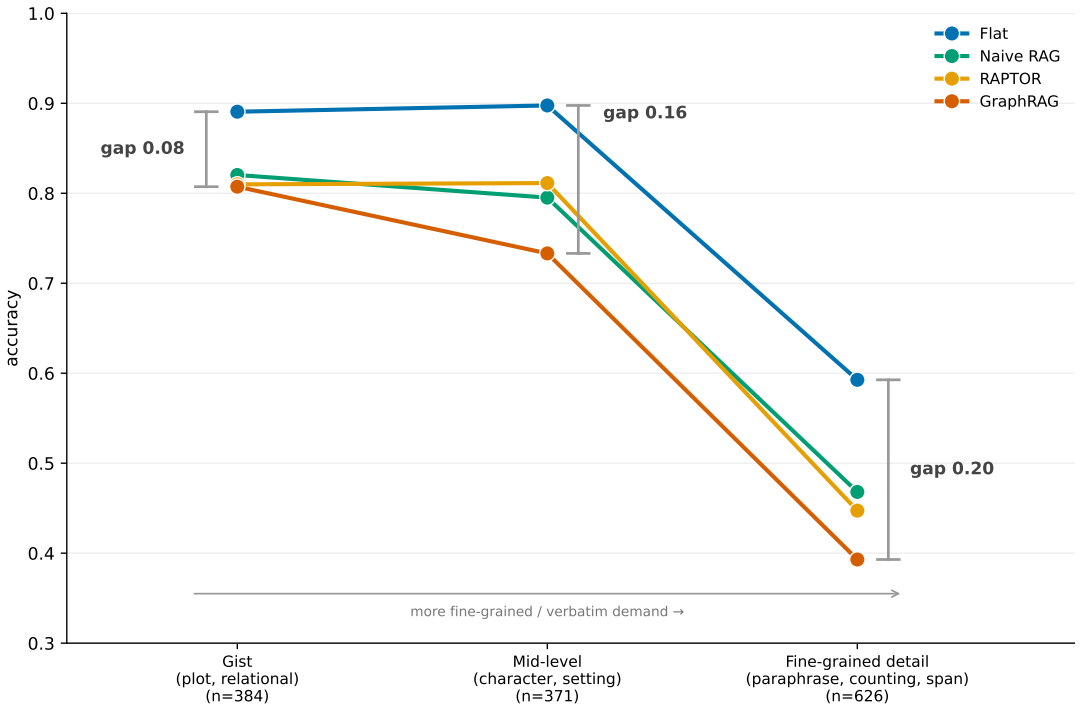

Figure 4: NovelQA accuracy by question group. Per-method accuracy on three groups of NovelQA questions ordered by how much verbatim or exhaustive text access the answer needs; the grey whiskers mark the Flat-to-worst gap.

The one remaining trade—pay for Flat's quality or save with Naive RAG—is quantified by a break-even analysis. For RAPTOR and GraphRAG, amortized cost crosses Flat's only above the break-even density $n^\star$ (Section 3.4), when it crosses at all (Table 9, Figure 6); Naive RAG needs no threshold, staying cheaper than Flat at any density on both datasets. The thresholds are read per dataset. On the short QASPER papers (≈ 4 questions per paper) neither summary-based method repays—RAPTOR needs $n^\star \approx 9$ questions under the standard card, above the observed density, and GraphRAG *never* breaks even, because its per-query pipeline already costs more than Flat's cache-warm re-read of the short paper. On the long NovelQA novels (≈ 25 questions per novel) only RAPTOR repays, and only under the standard card ($n^\star \approx 16$, rising above the density to 28 under the cache-discount card); GraphRAG needs ≈ 71 questions per novel under the standard card and ≈ 126 under the cache-discount card—far beyond the observed density either way—because its whole-novel entity extraction dominates the build.

4.5 Closed-Book Control

Under the closed-book control (Section 3.4), the NovelQA floor is 0.59, well above the 0.25 chance rate of four-way multiple choice (Table 10). The document raises accuracy unevenly across architectures: Flat gains +0.16 over the floor, while GraphRAG adds +0.01. On QASPER the floor is 0.16, far below every architecture's score, and every architecture lifts substantially (Flat +0.30, GraphRAG +0.24). The architecture ranking on this low-floor dataset matches the NovelQA ranking (Section 4.2). Figure 3 draws both floors as dashed lines.

Table 7: Example questions on which Flat and Naive RAG answer correctly while RAPTOR and GraphRAG do not. Answers are the verbatim model outputs from the first of the five repeats (run 0); for QASPER the gold column lists each annotator's reference answer. Question and gold text are quoted verbatim from the datasets, including any typographical errors; long texts are shortened for space, and a trailing ellipsis marks the table's own truncation.

Question	Gold	Flat / Naive RAG / RAPTOR / GraphRAG
NovelQA: Who are in Valeria's family?	D: Valeria, Eustace Woodville	D: Valeria, Eustace Woodville / D: Valeria, Eustace Woodville / B: Uncle Starkweather, Rosamond / B: Uncle Starkweather, Rosamond
NovelQA: In the novel The Law and the Lady, in which chapter does there exist a sentence with the...	C: Chapter 14	C: Chapter 14 / C: Chapter 14 / D: Chapter 18 / D: Chapter 18
NovelQA: Agnes Fleming, Oliver's mother, was the daughter of whom?	D: A retired naval officer.	D: A retired naval officer. / D: A retired naval officer. / B: A local parish clerk. / B: A local parish clerk.
QASPER: Do they build a model to automatically detect demographic, lingustic or psycological dime...	No / No	No. / No. / Yes. / Yes.

5 Discussion

5.1 Interpretation of Outcomes

The likely mechanism behind the ranking is that long-context models remove the small-window constraint these methods were built for (Section 2). When the whole document fits the answerer's window—as it does for both the QASPER papers and the scored NovelQA novels—selection can only discard information the model would otherwise have used, and summarization can only lose detail the answerer might have needed; cache-aware serving then makes the repeated re-reads cheap. The lost-in-the-middle phenomenon [33] that retrieval was meant to mitigate is, in this setting, smaller than the evidence loss retrieval introduces. GraphRAG suffers most because its entity-graph summarization is the most lossy transformation of the source text, which is consistent with its being last on quality on both datasets.

The error tallies of Section 4.3 support this reading. On the presence-and-count questions, the summary-based methods conclude a phrase is absent more often than plain chunk retrieval does (Section 4.3): the abstraction discards the exact wording the question asks about, so the method concludes it is absent. The penalty also does not track the amount of context retained: GraphRAG packs the largest per-query context of the three preprocessing-based architectures yet answers worst, while Naive RAG's mid-sized budget of raw chunks matches or beats both summary-based methods (Table A1). This is consistent with the penalty tracking how much of the source wording the retained context preserves, and with the closed-book lifts (Section 4.5), where GraphRAG supplies little usable evidence above the floor. In class terms: the whole document (Flat) preserves everything, verbatim excerpts (Naive RAG) preserve what similarity search selects, and summary-based context (RAPTOR and GraphRAG) rewrites what survives.

The short QASPER papers act as the contrast case: there the gap stays small across answer types, with none of the widening seen on NovelQA (Section 4.3), consistent with

Table 8: Per-architecture deployment cost in USD, computed *within each dataset* (standard card): the one-time build cost (the per-document C_{build} summed over the dataset's documents), the answering cost summed over its queries, their sum (Total)—the total deployment spend, i.e. the numerator of $\bar{C}_{\text{deploy}}$ before amortizing over the query count—and the marginal per-query answering cost C_{query} (m\$, 10^{-3} USD).

Architecture	Build (USD)	Answer (USD)	Total (USD)	C_{query} (m\$)
QASPER (≈ 4 questions/paper)				
Flat	0.00	0.38	0.38	0.39
Naive RAG	0.01	0.26	0.27	0.27
RAPTOR	0.37	0.22	0.60	0.23
GraphRAG	2.50	0.50	3.00	0.52
NovelQA (≈ 25 questions/novel)				
Flat	0.00	10.52	10.52	7.62
Naive RAG	0.02	0.39	0.41	0.28
RAPTOR	6.67	0.30	6.97	0.21
GraphRAG	27.33	0.85	28.17	0.61

the penalty arising where the summaries have substantial text to compress—a short paper leaves little of the exact wording to discard. That comparison is corpus-level: the two datasets also differ in metric and question taxonomy, so it bounds rather than isolates the effect of document length.

5.2 Relation to Existing Literature

The headline that Flat is hard to beat in the repeated-context setting is consistent with the long-context QA literature that motivates this study: "Lost in the Middle" [33] and the LongBench benchmarks [7, 8] document that long-context models read imperfectly, but the present result adds that preprocessing-based remedies do not, on balance, improve quality once the document already fits the window.

Our RAPTOR QASPER Answer-F1 sits below the source paper's own evaluation [46] (Section 6).

GraphRAG was designed for query-focused *sensemaking* over million-token corpora—broad summarize-the-whole-collection questions rather than specific-fact QA [17]; at the single-document scales used here its community-summary machinery has little of the cross-document aggregation it was built for, which is the most likely reason it transfers poorly to focused QA.

The closest prior work compares retrieval against long-context prompting head to head, and the conclusions are mixed. Retrieval has been reported to improve even long-context models regardless of their window size [55], and adding more text than the question needs has been reported to make the answer worse [59]; yet when the window is large enough to hold the input, long-context prompting has also been found to match or beat retrieval on quality, leaving retrieval's advantage mainly in cost. Self-Route [31] turns that trade-off into a per-query router: for each question it answers from the retrieved chunks when the model judges they contain the answer, and otherwise reads the full document. This thesis sharpens that comparison on three fronts. It compares flat full-context

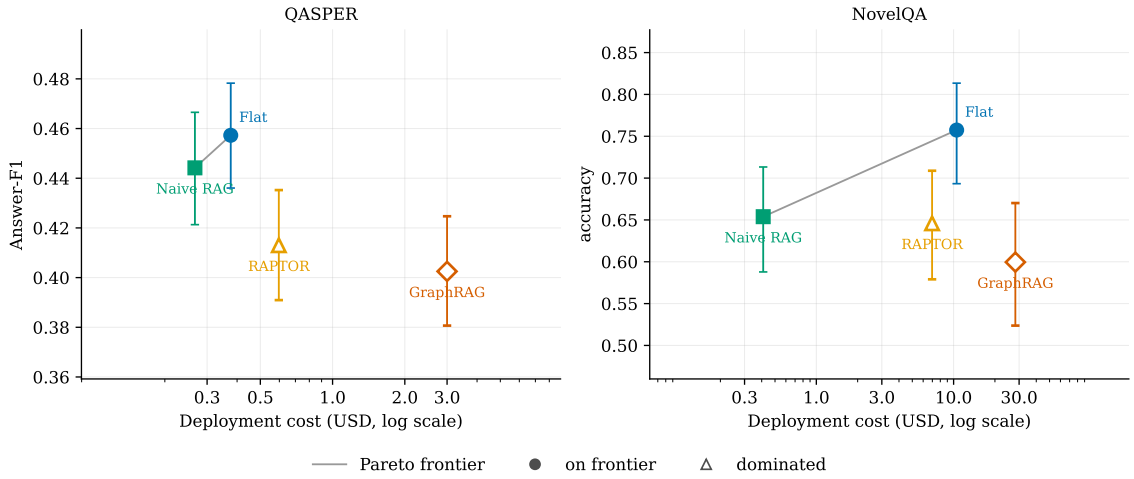

Figure 5: Deployment cost (log USD) against answer quality, one panel per dataset; cost is computed within each dataset, so an architecture sits at its own cost on each panel. Filled markers lie on the Pareto frontier; hollow markers are dominated. Whiskers are 95% clustered-bootstrap intervals on quality.

prompting not only against Naive RAG but against the summary-based methods built to beat it, RAPTOR and GraphRAG. It scores both sides under one resource ledger (Section 3.4.1), so "cheaper" is an amortized cost this study measures rather than assumes. And it holds the whole model configuration fixed across methods (Section 3), isolating the structuring method from model scale—a control that broad long-context benchmarks such as HELMET [58], which vary tasks and models together, are not designed to provide. The result favors reading the whole document when it fits the window, and warm-cache serving closes the cost gap that made retrieval attractive. For a document that already fits, the open question is therefore no longer whether to retrieve or to read, but whether any structuring beyond plain chunking is still worth building.

6 Threats to Validity and Failure Modes

Setting narrowness. The study is narrow on two axes. First, it evaluates the repeated-context setting. Applications that ask only one or a few questions per document fall outside it, and the findings may not transfer to them. Second, it evaluates real long *documents*, not the synthetic long-*context* setting (probe benchmarks, aggregation tasks), so the findings apply to long-document QA rather than to all long-context tasks.

Benchmark and architecture coverage. The comparison spans two datasets and four architectures. The two datasets—one short and free-form, one long and multiple-choice—bracket the length and format axes but cannot represent every document type, and the four architectures are three common preprocessing structures (tree, graph, and chunk) plus the flat baseline, not the full space of retrieval and structuring designs. The ranking reported here is therefore established on this sample of datasets and architectures; whether it transfers to other datasets (for example NarrativeQA [27] or Oolong [10]) or other architectures (indexing single factual statements, memory-graph indexes, or retrieval that combines keyword and embedding search or re-scores retrieved chunks with a second model) is left to future work.

Table 9: Break-even density versus cache-aware Flat, computed *within each dataset*. An architecture's one-time per-document build cost C_{build} amortizes over n questions per document; the two $n^\star$ columns give the break-even density (Section 3.4) under the standard card and the cache-discount card.

Architecture	C_{build} (m$)	C_{query} (m$)	$n^\star$ standard card	$n^\star$ cache-discount card
QASPER (≈ 4 questions/paper; Flat 0.39/0.37 m$/q standard/cache-discount)				
Naive RAG	0.02	0.274	any $n \geq 1$	any $n \geq 1$
RAPTOR	1.51	0.233	9.3	11.0
GraphRAG	10.05	0.522	never	never
NovelQA (≈ 25 questions/novel; Flat 7.62/4.55 m$/q standard/cache-discount)				
Naive RAG	0.30	0.285	any $n \geq 1$	any $n \geq 1$
RAPTOR	121.32	0.215	16.4	28.0
GraphRAG	496.84	0.613	70.9	126.0

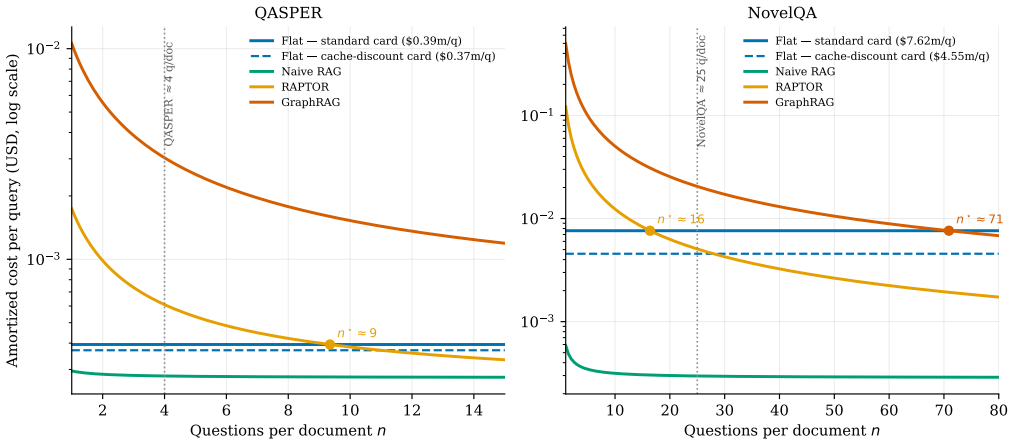

Figure 6: Amortized deployment cost per query against questions per document, one panel per dataset. Each preprocessing-based architecture's build amortizes over the queries; the crossing with Flat's per-query line marks the break-even density $n^\star$ (marker on the standard card), and Flat is shown under both price cards. The dotted vertical line marks each dataset's observed query density (∼4 questions per paper, ∼25 per novel).

Hyperparameters at literature defaults. Each architecture runs at its published defaults (Section 3.4). This leaves open whether settings tuned to these datasets would narrow or reverse the reported gaps; GraphRAG's defaults in particular target corpus-scale summarization questions (Section 5). A tuned re-comparison is left to future work.

Summary-based evidence loss. Summary-based preprocessing (as in RAPTOR and GraphRAG) may discard evidence that later queries need, rather than retrieving it but using it poorly; the answer-quality results are consistent with this. Isolating the effect requires retrieval interfaces that emit spans aligned to QASPER's sentence-level gold annotations, and it is in any case ill-defined for Flat, which ingests the whole document and so trivially contains every gold sentence. This study therefore reports answer quality ($\bar{M}$) only; a span-level evidence-grounding comparison across the preprocessing-based architectures is left to future work.

Table 10: Closed-book control: per-architecture answer quality against the closed-book floor (the same questions asked with the document withheld; Section 3.4). The lift column is the with-document score minus the closed-book floor.

Architecture	QASPER Answer-F1 with doc.	lift	NovelQA accuracy with doc.	lift
Closed-book floor	0.16	—	0.59	—
Flat	0.457	+0.30	0.757	+0.16
Naive RAG	0.444	+0.28	0.654	+0.06
RAPTOR	0.413	+0.25	0.646	+0.05
GraphRAG	0.403	+0.24	0.600	+0.01

Cost accounting. Cost accounting may be incomplete if preprocessing, storage, batching, or serving-side caching are not measured honestly. The resource ledger reports deployment cost under both price cards (Section 3.4.1); the Pareto frontier and the quality ranking hold under both cards (the discount lowers only Flat's cost, so no dominance relation can flip), and the one card-dependent outcome—whether RAPTOR's NovelQA build repays within the observed query density—is reported under both (Appendix A, Section 4.4).

Flat's per-query cost is a modeling choice worth stating. Flat pays an uncached first read of each document and cheaper cached re-reads thereafter, and this analysis charges it the realized mean of the two over the observed questions per document. That first read could instead be treated as a one-time build cost, giving Flat its own amortizing curve rather than a density-independent per-query cost. The two treatments agree exactly at the observed density, so the reported costs and the Pareto frontier are unaffected; they differ away from it, so the break-even densities depend on this choice. Re-deriving break-even under the alternative treatment changes no verdict at the observed densities on either card. The one qualitative change lies well above the observed density on the cache-discount card, where Flat's cached re-reads undercut Naive RAG on short documents.

Benchmark artifacts. NovelQA's multiple-choice format can reward eliminating some wrong options without knowing the answer, and shifts the domain toward long narrative understanding, while QASPER question difficulty depends on the question mix and on how figure and table questions are handled [14, 51].

Expected failure cases. A summary keeps the main content of a document and drops the rest, so the questions at risk under summary-based preprocessing are those whose evidence sits in what gets dropped: detail-seeking questions (exact wording), numeric questions (exact counts), and rare-entity questions (entities mentioned too rarely to survive summarization). The error analysis in Section 4.3 examines the first two kinds; neither dataset labels which questions target rare entities, so that group cannot be formed and is not analyzed. That analysis tallies observed errors; it manipulates nothing experimentally. Group sizes bound its precision (per-group intervals in Appendix B), the failing answers it quotes are illustrative, and the granularity regrouping is an analysis-time choice, checked against the benchmark's own complexity labels (Section 4.3).

Dynamic contexts. Dynamic or continuously updated contexts are out of scope, because rebuild-cost accounting would define a different amortization problem.

Adversarial-context faithfulness. The benchmark does not directly measure whether preprocessing introduces adversarial-context faithfulness failures—summarization that smooths over contradictions in the source text, distinct from the evidence loss above. Contextual faithfulness benchmarks such as FaithEval [37] are left to future work.

Answer randomness. For a given prompt, the same architecture can return different answers across repeats. The protocol mitigates this with $T = 0$ and five repeats per (architecture, query) pair (Section 3.4.4). Greedy decoding at $T = 0$ also narrows the output distribution: it does not reflect the higher-temperature sampling typical of production deployments.

Memorization. Memorization is a general threat to any benchmark drawn from public text, but here it is far more visible on NovelQA than on QASPER. The scored NovelQA novels are public-domain classics that long-context models have very likely seen in pretraining, so absolute NovelQA accuracies are inflated by memorization and should not be read as pure reading comprehension. The closed-book control (Section 4.5) bounds this. Because the same ranking holds on QASPER, whose floor sits far below every score (Section 4.5), the comparative finding survives the memorization caveat even though the NovelQA *absolute* numbers do not. A contamination-controlled benchmark such as NoCha [26] (Section 2) would harden this check and is left to future work.

Held-out scoring and excluded novels. NovelQA's gold answers stay private on Codabench (Section 3.4.2), so all NovelQA correctness values depend on a scorer this study cannot inspect; a scoring error there would pass through undetected. The check available locally was run: each returned score was matched one-to-one to its question, ruling out misalignment. The scored pool and its exclusions are defined in Section 4.1; including the calibration novels changes no headline result and slightly narrows Flat's lead.

Absolute quality levels are configuration-dependent. The absolute Answer-F1 levels reported here sit below the figures the source architectures report under their own configurations, because the benchmark fixes a single small, low-cost answerer and holds the rest of the configuration identical across architectures (Section 3). The claims of this thesis are comparative—which architecture wins, and by how much—and do not depend on reproducing any source paper's absolute level.

Answerer portability. The single-answerer design supports architecture comparisons within one answerer but not across vendors. A cross-vendor study arm—re-running the four architectures under a non-Google answerer that already cleared the study's serving gates, `grok-4-fast-reasoning` (xAI) or `deepseek-v4-pro` (DeepSeek)—would test whether the architecture ranking is answerer-portable, and is left to future work.

Pilot small-sample caveat. The pilot rank-stability statistic that informed the single-answerer design is a preliminary diagnostic computed at a small per-cell sample size, and its median sits at the edge of moderate agreement rather than safely above it (Appendix A). No quantitative claim rests on it—all reported intervals come from the document-clustered paired bootstrap on the full scored pools—and re-estimating rank stability on a larger sample is left to future work.

7 Conclusion

This thesis studied repeated-context question answering through the controlled benchmark of Section 3, comparing flat full-context prompting, Naive RAG, RAPTOR, and GraphRAG. The headline is that with long-context models, the tree and graph structure that RAPTOR and GraphRAG add does not pay in this setting.

On the cost-quality tradeoff (RQ1) the ranking is identical on QASPER and NovelQA: flat full-context prompting answers best, Naive RAG and RAPTOR follow (statistically tied on NovelQA), and GraphRAG is last (statistically tied with RAPTOR on QASPER). The cost-quality plane leaves a two-point Pareto frontier—Flat for quality, Naive RAG for budget—with RAPTOR and GraphRAG both dominated (Section 4.4).

What explains the differences (RQ2) is how each pipeline treats source wording rather than how much context it retains (Section 5.1): retrieval omits evidence the full text retains, and summarization rewrites wording that survives retrieval.

On query density (RQ3) Naive RAG's build is cheap enough to undercut Flat from the first question on both datasets. The summary-based builds need a threshold. Against the ≈ 25 questions each novel is asked, RAPTOR repays only on the novels and only under the standard card ($n^\star \approx 16$), while GraphRAG never crosses on the papers and needs ≈ 71 questions on the novels (Section 4.4).

The controlled benchmark is the contribution; the per-architecture verdict is the finding. Reproducibility was a design goal throughout: the full run record—per-question predictions, resource ledger, and configuration manifest—is public alongside the pipeline and analysis code, and every number, table, and figure in this thesis regenerates deterministically from those artifacts (Appendix A).

Acknowledgment

I would like to thank my supervisor, Tiago Prince Sales, for his guidance throughout this project; Patricia Ferreira da Silva for her valuable feedback; and Mariet Theune for grading this thesis.

This research was sponsored by Prudai B.V. (Enschede), which also provided the facilities in which the work was carried out. I thank Geert Haisma, my internal supervisor at Prudai, for the support throughout, and I am grateful to the company for making this work possible.

Declaration on the Use of Generative AI

During the preparation of this work, I used generative AI tools, including ChatGPT, Claude Code, Codex, and Notebook LM, to support drafting, language editing, restruc-

turing, citation-gap checking, and LaTeX or code-related workflow tasks. These tools were used as assistants rather than as authoritative sources: I independently reviewed the cited literature, verified substantive claims and references, and revised the manuscript where needed. I take full responsibility for the final text, interpretations, and research decisions.

References

[1] Keivan Alizadeh, Parshin Shojaee, Minsik Cho, and Mehrdad Farajtabar. Recursive language models meet uncertainty: The surprising effectiveness of self-reflective program search for long context. *arXiv preprint arXiv:2603.15653*, 2026.

[2] Norah Alzahrani, Hisham Abdullah Alyahya, Yazeed Alnumay, Sultan Alrashed, Shaykhah Alsubaie, Yusef Almushaykeh, Faisal Mirza, Nouf Alotaibi, Nora Al-twairesh, Areeb Alowisheq, M. Saiful Bari, and Haidar Khan. When benchmarks are targets: Revealing the sensitivity of large language model leaderboards. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13787–13805, 2024. arXiv:2402.01781.

[3] Arena Intelligence. Arena leaderboard. `https://arena.ai/leaderboard`, 2026. Leaderboard webpage, accessed 2026-07-20.

[4] Arena Team. Arena-rank: Open sourcing the leaderboard methodology. `https://arena.ai/blog/arena-rank/`, December 2025. Blog post published 2025-12-18, accessed 2026-07-20.

[5] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2024.

[6] Berk Atil, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J. Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Ture, Zhe Wu, Lixinyu Xu, and Breck Baldwin. Non-determinism of "deterministic" LLM system settings in hosted environments. In *Proceedings of the 5th Workshop on Evaluation and Comparison of NLP Systems*, pages 135–148, Mumbai, India, 2025. Association for Computational Linguistics. arXiv:2408.04667.

[7] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, 2024.

[8] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, 2025. arXiv:2412.15204.

[9] Andrew M. Bean, Ryan Othniel Kearns, Angelika Romanou, Franziska Sofia Hafner, Harry Mayne, Jan Batzner, Negar Foroutan, Chris Schmitz, Karolina Korgul, Hunar Batra, Oishi Deb, Emma Beharry, Cornelius Emde, Thomas Foster, Anna Gausen, María Grandury, Simeng Han, Valentin Hofmann, Lujain Ibrahim, Hazel Kim, Hannah Rose Kirk, Fangru Lin, Gabrielle Kaili-May Liu, Lennart Luettgau, Jabez Magomere, Jonathan Rystrøm, Anna Sotnikova, Yushi Yang, Yilun Zhao, Adel Bibi, Antoine Bosselut, Ronald Clark, Arman Cohan, Jakob Foerster, Yarin Gal, Scott A. Hale, Inioluwa Deborah Raji, Christopher Summerfield, Philip H.S. Torr, Cozmin Ududec, Luc Rocher, and Adam Mahdi. Measuring what matters: Construct validity in large language model benchmarks. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025), Datasets and Benchmarks Track*, 2025.

[10] Amanda Bertsch, Adithya Pratapa, Teruko Mitamura, Graham Neubig, and Matthew R. Gormley. Oolong: Evaluating long context reasoning and aggregation capabilities. *arXiv preprint arXiv:2511.02817*, 2025. Preprint; under review at ICLR 2026.

[11] Sinchana Ramakanth Bhat, Max Rudat, Jannis Spiekermann, and Nicolas Flores-Herr. Rethinking chunk size for long-document retrieval: A multi-dataset analysis. *arXiv preprint arXiv:2505.21700*, 2025.

[12] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense X Retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15159–15177, 2024. arXiv:2312.06648.

[13] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating LLMs by human preference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 8359–8388. PMLR, 2024. arXiv:2403.04132.

[14] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, 2021.

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

[16] Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1383–1392. Association for Computational Linguistics, 2018.

[17] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson.

From local to global: A Graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[18] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[19] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.14831.

[20] Horace He and Thinking Machines Lab. Defeating nondeterminism in LLM inference. Thinking Machines Lab Connectionism blog, `https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/`, 2025. Accessed 2026-07-20.

[21] Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts LLM performance. Technical report, Chroma, July 2025. `https://www.trychroma.com/research/context-rot`, accessed 2026-08-17.

[22] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling (COLM)*, 2024. arXiv:2404.06654.

[23] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050, 2024.

[24] Ziyan Jiang, Xueguang Ma, and Wenhu Chen. LongRAG: Enhancing retrieval-augmented generation with long-context LLMs. *arXiv preprint arXiv:2406.15319*, 2024.

[25] Greg Kamradt. Needle in a haystack – pressure testing LLMs. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`, 2023. GitHub repository, accessed 2026-08-17.

[26] Marzena Karpinska, Katherine Thai, Kyle Lo, Tanya Goyal, and Mohit Iyyer. One thousand and one pairs: A "novel" challenge for long-context language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17048–17085, 2024.

[27] Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

[28] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2024. NeurIPS 2024 Datasets and Benchmarks Track.

[29] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pages 611–626. Association for Computing Machinery, 2023.

[30] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Timm Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[31] Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Retrieval augmented generation or long-context LLMs? A comprehensive study and hybrid approach. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 881–893, 2024.

[32] Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E. Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025.

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

[34] LMSYS / SGLang Team. Towards deterministic inference in SGLang and reproducible RL training. LMSYS blog, `https://lmsys.org/blog/2025-09-22-sglang-deterministic/`, September 2025. Accessed 2026-07-20.

[35] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[36] Microsoft. GraphRAG: Local search. `https://microsoft.github.io/graphrag/query/local_search/`, 2024. Project documentation, accessed 2026-08-17.

[37] Yifei Ming, Senthil Purushwalkam, Shrey Pandit, Zixuan Ke, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. FaithEval: Can your language model stay faithful to context, even if "the moon is made of marshmallows". In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.03727.

[38] Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. NoLiMa: Long-context evaluation beyond literal matching. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. arXiv:2502.05167.

[39] Inderjeet Nair, Shwetha Somasundaram, Apoorv Saxena, and Koustava Goswami. Drilling down into the discourse structure with LLMs for long document question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14593–14606, 2023.

[40] NovelQA Team. Dataset card for NovelQA. `https://huggingface.co/datasets/NovelQA/NovelQA`, 2024. Hugging Face dataset card, accessed 2026-07-20.

[41] Zhihong Pan, Kai Zhang, Yuze Zhao, and Yupeng Han. Adaptive model and strategy routing for cost-efficient LLM services. In *Proceedings of the ACM Web Conference 2026 (WWW '26)*. Association for Computing Machinery, 2026. arXiv:2505.19435.

[42] Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R. Bowman. QuALITY: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5336–5358, 2022.

[43] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. Graph retrieval-augmented generation: A survey. *ACM Transactions on Information Systems*, 44(2):35:1–35:52, 2026. arXiv:2408.08921.

[44] Anna Rogers, Matt Gardner, and Isabelle Augenstein. QA dataset explosion: A taxonomy of NLP resources for question answering and reading comprehension. *ACM Computing Surveys*, 55(10), 2023.

[45] Amartya Roy, Rasul Tutunov, Xiaotong Ji, Matthieu Zimmer, and Haitham Bou-Ammar. The Y-Combinator for LLMs: Solving long-context rot with λ-calculus. *arXiv preprint arXiv:2603.20105*, 2026.

[46] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. arXiv:2401.18059.

[47] Brandon Smith and Anton Troynikov. Evaluating chunking strategies for retrieval. Technical report, Chroma, July 2024. `https://research.trychroma.com/evaluating-chunking`, accessed 2026-08-17.

[48] Suhas Jayaram Subramanya, Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 13748–13758, 2019.

[49] Wenyu Tao, Xiaofen Xing, Yirong Chen, Linyi Huang, and Xiangmin Xu. TreeRAG: Unleashing the power of hierarchical storage for enhanced knowledge retrieval in long documents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 356–371, 2025.

[50] vLLM Project. vLLM documentation. `https://docs.vllm.ai/`, 2026. Project documentation accessed 2026-07-20.

[51] Cunxiang Wang, Ruoxi Ning, Boqi Pan, Tonghui Wu, Qipeng Guo, Cheng Deng, Guangsheng Bao, Xiangkun Hu, Zheng Zhang, Qian Wang, and Yue Zhang. NovelQA: Benchmarking question answering on documents exceeding 200K tokens. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. arXiv:2403.12766.

[52] Daren Wang. Think, but don't overthink: Reproducing recursive language models. *arXiv preprint arXiv:2603.02615*, 2026.

[53] xAI. Grok 4 fast. `https://x.ai/news/grok-4-fast`, 2025. Product announcement, accessed 2026-08-17.

[54] Cheng Xu, Shuhao Guan, Derek Greene, and M-Tahar Kechadi. Benchmark data contamination of large language models: A survey. *arXiv preprint arXiv:2406.04244*, 2024.

[55] Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. Retrieval meets long context large language models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[56] Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. Benchmarking benchmark leakage in large language models. *arXiv preprint arXiv:2404.18824*, 2024.

[57] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, 2018.

[58] Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. HELMET: How to evaluate long-context language models effectively and thoroughly. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.02694.

[59] Tan Yu, Anbang Xu, and Rama Akkiraju. In defense of RAG in the era of long-context language models. *arXiv preprint arXiv:2409.01666*, 2024.

[60] Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in LLM inference. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025)*, 2025. arXiv:2506.09501.

[61] Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025.

[62] Haozhen Zhang, Haodong Yue, Tao Feng, Quanyu Long, Jianzhu Bao, Bowen Jin, Weizhi Zhang, Xiao Li, Jiaxuan You, Chengwei Qin, and Wenya Wang. Learning query-aware budget-tier routing for runtime agent memory. *arXiv preprint arXiv:2602.06025*, 2026.

[63] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö. Arik. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

Table A1: Per-architecture protocol parameters used in the benchmark. All embedding-aware methods share one encoder (BGE-M3, served locally via the Ollama model server); the final answerer and all preprocessing summaries use `gemini-3.1-flash-lite-preview` at temperature 0 with a 256-token answer cap. The clustering (UMAP+GMM: UMAP dimensionality reduction followed by Gaussian-mixture clustering), community-detection (Louvain), and gleaning (repeated-extraction) steps are the RAPTOR and GraphRAG defaults [46, 17].

Architecture	**Preprocessing / index**	**Query-time retrieval and answer context**
Flat	None; the full document is used verbatim.	None; the entire document is passed to the answerer.
Naive RAG	Sentence-boundary chunks of 384 tokens with zero overlap; every chunk embedded with the shared encoder.	Top-8 chunks by cosine similarity ($\sim$3,000-token context).
RAPTOR	100-token leaf chunks; recursive UMAP+GMM clustering up to 5 layers; one $\sim$200-token abstractive summary per internal node; leaves and summaries embedded.	Collapsed-tree retrieval of the top-20 nodes within a 2,000-token budget.
GraphRAG	600-token chunks (100-token overlap); per-chunk entity and relationship extraction with one gleaning pass; NetworkX Louvain communities; one report summary per community.	Local search over the top-10 entities and top-10 relationships, packed into an 8,000-token context (community / text-unit proportions 0.15/0.50).

A Protocol Summary

This appendix records the protocol defaults for the benchmark study, all fixed before experiments begin. They are the settings used in this study, not recommended settings in general. To place a new method or dataset on the same cost-quality plane, three inputs suffice: a per-call resource ledger (uncached input, cached input, and output tokens, plus any local compute), the task's native quality metric normalized to $[0, 1]$, and the document-level unit used for clustered resampling.

A.1 Preprocessing and Representation

Table A1 lists the per-architecture preprocessing, indexing, and retrieval parameters used in the benchmark; the architectures themselves are specified in Section 3.3.

A.2 Break-even and Cost Reporting

RQ3 is read off the amortized-cost curve (Sections 3.4.1 and 4.4). Table 9 reports the per-document build cost and per-query cost per dataset, and $n^\star$ under both price cards. Even a warm re-read is not automatically cheap: a long novel still presents a very large context on every read, so a discounted re-read of a big document can cost almost as much as an undiscounted first read of a small one. The fixed GPU and storage rates (Section 3.4.1) set GPU time on the experiment laptop's RTX 4070 at \$0.30 per GPU-hour (8.33×10^{-5} USD per GPU-second), a community-cloud rate for a comparable consumer GPU rather than a datacenter accelerator. Persistent storage is billed at \$0.023 per GiB-month (AWS S3

Standard list price, 2026-04) over a 90-day horizon, a one-quarter-year reporting window fixed in advance. The only GPU-rate term in this study is the BGE-M3 embedding compute.

Per-architecture storage footprints are measured, except Naive RAG's, which is estimated from its index size; Flat's small storage line is the cached raw document text. Serving latency enters the study only as a qualitative model-selection gate (Section 3.5.2), not as a reported cost axis. Table A2 and Figure 7 break each architecture's deployment cost into its parts—one-time build versus per-query answering, each split into LLM API calls and local embedding, with storage shown separately. Build cost is dominated by LLM calls (Table A2); of Flat's answering total, $7.09 is the warm re-reads that the cache-discount card reprices and $3.81 is the uncached first read of each document, which no cache discount touches. The study-wide deployment total is $50.32 under the standard card and $46.06 under the cache-discount card.

Table A2: Where each architecture's deployment dollars go, summed over the whole study (standard card; the per-dataset split is Table 8): the one-time build cost (the per-document C_{build} summed over all documents) and the answering cost (the per-query C_{query} summed over all queries), each split into LLM API calls and local BGE-M3 embedding. Persistent-artifact storage (C_{store}) is shown separately and is excluded from the deployment totals. All values in USD.

	Build		Answer			
Architecture	LLM (USD)	embed (USD)	LLM (USD)	embed (USD)	Total (USD)	Storage (USD)
Flat	0.00	0.00	10.90	0.00	10.90	0.004
Naive RAG	0.00	0.02	0.62	0.03	0.68	0.013
RAPTOR	4.45	2.60	0.45	0.07	7.57	0.192
GraphRAG	29.82	0.01	1.31	0.03	31.17	0.104

Realized versus modeled cost. The dominant LLM term applies the provider's published rates to the realized per-call token ledger and is therefore measured, not estimated; only the embedding GPU rate and the list-price storage term are modeled. The single actually-billed quantity is the run-wide Gemini total of $101.13, from which the per-deployment figures are projected (one repeat over the scored pool, each build charged once per document).

A.3 Model Selection and Statistical Reporting

Leaderboard signals only screen candidates (Section 3.5.2). The calibration pool is used to select the answerer, and its QASPER half to confirm the embedding encoder; calibration questions are excluded from the scored pools, and the four NovelQA calibration novels are additionally held out entirely (Section 4.1). Retrieval settings follow Table A1 (Section 3.4); nothing is tuned on the calibration pool. Each evaluated system's answers over the full evaluation split are recorded under one run manifest per fixed checkpoint (five repeats per question; Section 3.4.4). Final paired confidence intervals are computed offline using 10,000 clustered bootstrap resamples.

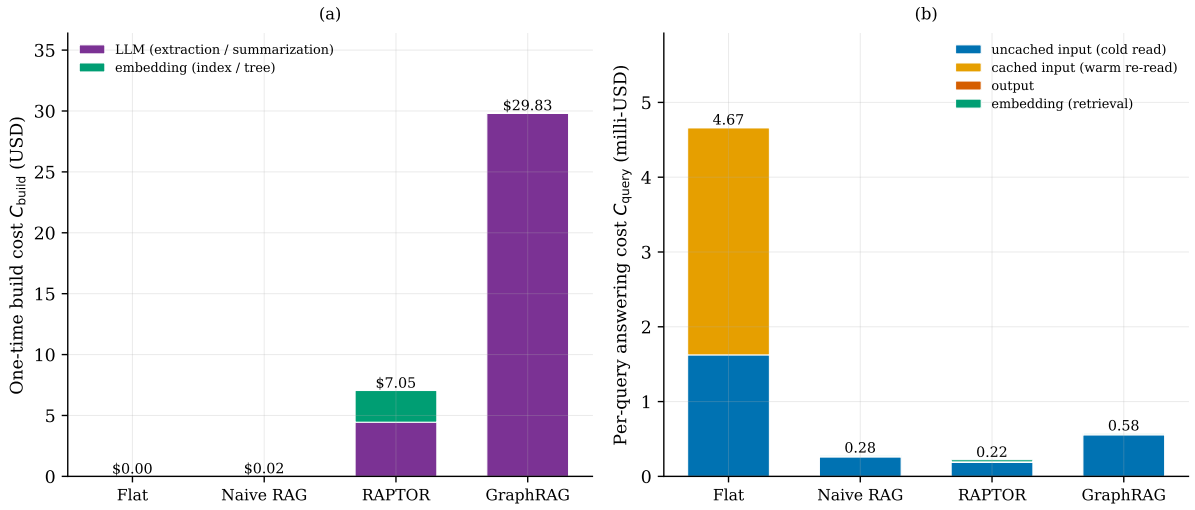

Figure 7: Where each architecture's deployment cost comes from, decomposed by ledger source (standard card, scored pool). **(a)** The one-time build cost C_{build}, split into LLM calls and dense embedding. **(b)** The per-query answering cost C_{query}, split into its uncached-input, cached-input, output, and retrieval-embedding components. The two panels are the two terms of the amortized deployment cost (Section 3.4.1).

Pilot audit summary. The pilot grid and its calibration pool are specified in Section 3.5.2. The grid used three of the four designated calibration novels (Section 4.1); all four remain excluded from scoring. It fixed the configuration the main study inherits. The pilot's scoring scripts are part of the public companion code repository.

Pilot rank stability. Rank stability was checked before the answerer was fixed (Section 3.5.2), over the $\binom{9}{2} = 36$ candidate-pairs per dataset; Table A3 reports the distribution. On both datasets the median pairwise τ_b sits at the value of a single rank swap among four architectures, so cross-candidate agreement is moderate rather than strong—with several pairs in each dataset in perfect agreement. Following Dror et al. [16], bootstrap 95% confidence intervals use 10,000 resamples over candidate-pairs, and one-sided permutation p values use 10,000 candidate-ranking shuffles with add-one smoothing; the permutation test rejects the no-agreement null on both datasets (Table A3). On the four-architecture grid τ_b is confined to $\{-1, -2/3, -1/3, 0, 1/3, 2/3, 1\}$, and the pilot per-cell sample is small (Section 6).

Statistic	NovelQA	QASPER
Median pairwise τ_b	2/3	2/3
Mean pairwise τ_b	0.630	0.722
Permutation p value (one-sided)	5×10^{-4}	3×10^{-4}
Pairs with perfect agreement $\tau_b = 1$	9 / 36	11 / 36
Pairs with $\tau_b \geq 2/3$	26 / 36	31 / 36
Pairs with $\tau_b \leq 0$ (rank-disagreement)	3 / 36	0 / 36

Table A3: Distribution of pairwise Kendall's τ_b across the 36 pilot candidate-pairs on each dataset under gold scoring. Statistics use 10,000 bootstrap resamples and permutation shuffles with add-one smoothing [16].

Two causes eliminate the most candidates in the funnel (Table A4), three each: account-tier rate limits on input tokens, and missing API support for greedy decoding. The rate limits block the Anthropic family and `gpt-5.4` on repeated long-context calls; `gpt-5.4` fails only the funnel's smoke test, because its 600k-token tier returns a rate-limit error rather than a latency or format failure.

Reproducibility artifacts. The main study is a single complete run (one run identifier, fixed answerer, summary model, and embedder above). Its per-cell predictions, resource ledger, run manifest, and the derived analysis artifacts are released as a public dataset,[1] and the pipeline and analysis code in the public companion code repository.[2] Every reported metric, table, and figure is regenerated from those artifacts by deterministic, version-controlled scripts: per-cell scoring, the clustered-bootstrap confidence intervals and paired tests (fixed resample count and seed), the per-architecture cost accounting under both price cards, the break-even analysis, and figure generation. The prompt templates are fixed and quoted in Section 3.5.1, and the locked configuration is recorded in the released run manifest.

B Error-Analysis Details

This appendix gives the full per-method breakdown behind the error analysis of Section 4.3, computed deterministically from the released per-question scores and the datasets' own question labels. Aspect labels follow NovelQA's released question taxonomy [51], displayed under the renaming of Section 3.1; the aspect-to-group mapping behind Figure 4 is stated with its rationale in Section 4.3. Section 4.3 also repeats the analysis under NovelQA's own complexity labels, with no regrouping. The two axes are distinct: counting and span questions all fall in the multi-hop class, not the detail class, yet carry the largest per-aspect penalties (Table A5). The complexity dimension alone would therefore not isolate the verbatim-access effect the mechanism analysis targets, and the complexity check is a complementary test rather than the same grouping re-run. All tables report per-subset sample sizes and 95% clustered-bootstrap confidence intervals (clusters are novels for NovelQA and papers for QASPER). Table A5 reports NovelQA accuracy per architecture by question aspect, and Table A7 reports QASPER Answer-F1 per architecture by answer type.

[1] `https://huggingface.co/datasets/jonkhout/repeated-context-qa-artifacts`
[2] `https://github.com/BCJonkhout/msc-thesis-code`

Table A4: LLM answerer-candidate screening funnel for the pilot. Seventeen candidates across six vendor families entered Step 1 smoke testing; the nine that cleared every serving gate entered the 9×4 evaluation grid. Five further candidates (bottom group) were screened out before Step 1 on provider-level prompt-caching availability or context-window limits. Gates, left to right: **API**—greedy decoding ($T = 0$) and Chat Completions support; **Window**—context window covers the longest non-B48 NovelQA novel plus prompt overhead (∼780k tokens in total); **Smoke**—Step 1 latency below ten minutes and well-formed output across the 5k/150k/600k tiers; **Cache**—Step 2 KV-cache and rate-limit viability under repeated long-context calls; **Cost**—operational cost and serving reliability under load. ✔ passed, ✘ failed (drop point), — not reached. † selected answerer; ‡ candidate designated for the cross-vendor study arm (future work, not run in the main study).

Candidate	API	Window	Smoke	Cache	Cost	**In grid**
Anthropic						
`claude-sonnet-4-6`	✔	✔	✔	✘	—	✘
`claude-opus-4-7`	✔	✔	✘	—	—	✘
OpenAI						
`gpt-5.4`	✔	✔	✘	—	—	✘
`gpt-5.5`	✘	—	—	—	—	✘
`gpt-5.4-pro`	✘	—	—	—	—	✘
`gpt-5.5-pro`	✘	—	—	—	—	✘
Google						
`gemini-3.1-pro-preview`	✔	✔	✔	✔	✘	✘
`gemini-3.1-flash-lite-preview`†	✔	✔	✔	✔	✔	✔
`gemini-flash-latest`	✔	✔	✔	✔	✔	✔
xAI						
`grok-4.3`	✔	✔	✔	✔	✔	✔
`grok-4-1-fast-non-reasoning`	✔	✔	✔	✔	✔	✔
`grok-4-fast-reasoning`‡	✔	✔	✔	✔	✔	✔
`grok-4.20-0309-non-reasoning`	✔	✔	✔	✔	✔	✔
`grok-4.20-0309-reasoning`	✔	✔	✔	✔	✔	✔
DeepSeek (via OpenRouter)						
`deepseek-v4-pro`	✔	✔	✔	✔	✔	✔
`deepseek-v4-flash`	✔	✔	✔	✔	✔	✔
Moonshot (via OpenRouter)						
`kimi-k2.6`	✔	✘	—	—	—	✘
Screened before Step 1 (provider-level)						
`qwen3.6-flash`	no prompt caching exposed via OpenRouter					✘
`minimax-m2.7`	prompt caching unverified					✘
`qwen3.6-27b`	provider (DashScope) inaccessible					✘
`gemma-4-26b-a4b-it`	256k window < 780k requirement					✘
`gemma-4-31b-it`	256k window < 780k requirement					✘

Table A5: NovelQA per-method accuracy by question aspect (full scored pool). Brackets give 95% clustered-bootstrap confidence intervals (percentile method, 10 000 resamples, resampling novels with replacement); n is the number of questions in the subset.

subset	n	Flat	Naive RAG	RAPTOR	GraphRAG
Counting (`times`)	356	0.45	0.34	0.29	0.24
		[0.40, 0.50]	[0.29, 0.40]	[0.25, 0.35]	[0.18, 0.30]
Paraphrase (`meaning`)	248	0.77	0.63	0.65	0.58
		[0.68, 0.84]	[0.54, 0.72]	[0.59, 0.72]	[0.49, 0.68]
Span (`span`)	22	0.91	0.64	0.59	0.73
		[0.79, 1.00]	[0.43, 0.82]	[0.38, 0.77]	[0.52, 0.91]
Character	241	0.89	0.79	0.80	0.73
		[0.82, 0.94]	[0.71, 0.85]	[0.72, 0.86]	[0.64, 0.80]
Setting	130	0.91	0.81	0.84	0.74
		[0.78, 0.99]	[0.68, 0.90]	[0.71, 0.93]	[0.61, 0.84]
Relational (`relat`)	115	0.84	0.80	0.78	0.82
		[0.76, 0.92]	[0.71, 0.88]	[0.69, 0.87]	[0.73, 0.90]
Plot	269	0.91	0.83	0.82	0.80
		[0.83, 0.95]	[0.75, 0.88]	[0.74, 0.89]	[0.72, 0.85]

Table A6: NovelQA per-method accuracy by the dataset's own complexity labels (single-hop, multi-hop, detail). Same CI method as Table A5; n is the number of questions in the subset. Accuracies are rounded to two decimals; gaps quoted in the text are computed from the unrounded values.

subset	n	Flat	Naive RAG	RAPTOR	GraphRAG
Single-hop (`sh`)	476	0.94	0.86	0.85	0.83
		[0.90, 0.96]	[0.82, 0.89]	[0.80, 0.89]	[0.77, 0.87]
Multi-hop (`mh`)	567	0.59	0.49	0.46	0.44
		[0.54, 0.65]	[0.43, 0.55]	[0.40, 0.52]	[0.37, 0.50]
Detail (`dtl`)	338	0.78	0.64	0.67	0.55
		[0.70, 0.84]	[0.57, 0.72]	[0.61, 0.73]	[0.49, 0.62]

Table A7: QASPER per-method Answer-F1 by answer type (full scored pool). Brackets give 95% clustered-bootstrap confidence intervals (percentile method, 10 000 resamples, resampling papers with replacement); n is the number of questions in the subset.

subset	n	Flat	Naive RAG	RAPTOR	GraphRAG
yes/no	111	0.79	0.75	0.72	0.68
		[0.70, 0.87]	[0.66, 0.83]	[0.64, 0.80]	[0.60, 0.76]
extractive	509	0.48	0.47	0.44	0.43
		[0.45, 0.50]	[0.45, 0.50]	[0.41, 0.47]	[0.40, 0.46]
abstractive	244	0.36	0.35	0.32	0.31
		[0.33, 0.40]	[0.32, 0.39]	[0.28, 0.35]	[0.28, 0.35]
unanswerable	91	0.19	0.16	0.15	0.14
		[0.12, 0.27]	[0.09, 0.23]	[0.09, 0.22]	[0.08, 0.21]