MINDGRAPH: Faithful Concept-Graph Memory for Long-Context Reasoning

Anonymous Authors[1]

Abstract

Large language models (LLMs) struggle with long-context reasoning not because of limited context windows alone, but because they lack a working memory of accumulated knowledge that is at once *persistent*, *structured*, and *reasoning-friendly*. Prior approaches—long-context prompting, retrieval-augmented generation, summary-based generation, multi-agent memory, and recursive REPL agents—each fail at least one of these properties: they rebuild state from raw tokens on every query, compress it into unstructured summaries that drop contradictions and relationships, or impose structures that do not directly support multi-step inference. We isolate the regime where these failures compound—long inputs that demand both broad input understanding and deep multi-step inference—through LONGREASON-200, a 200-item benchmark filtered from LongBench-v2, NarrativeQA, and LooGLE. We then propose MINDGRAPH, a concept-graph memory framework: an incremental, question-aware build stage maintains a compact graph with character-level provenance to the original input, and an answer stage reasons over it via two tools that combine fine-grained verification and hierarchical navigation, choosing its access pattern from the graph's size. On LONGREASON-200, MINDGRAPH reaches 77% accuracy—above all five baselines—while passing only ∼7% of the original input to the answering LLM per query. The result suggests that effective long-context reasoning requires architectures that treat the working memory itself as a first-class design dimension.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the FoGen Workshop at ICML 2026. Do not distribute.

1. Introduction

Large language models (LLMs) are increasingly deployed in settings requiring reasoning over long and evolving contexts, including multi-document analysis, code repositories, extended dialogues, and interactive environments. In these regimes, models must integrate evidence across distant spans, maintain constraints over time, and update their understanding as new information arrives. Yet even with rapidly growing context windows, current LLMs remain unreliable: they lose constraints, overlook long-range dependencies, and hallucinate when relevant evidence is diffuse or indirectly connected (Liu et al., 2024).

A natural explanation is insufficient context length. However, simply scaling the window does not solve the underlying problem. Current systems still reconstruct state repeatedly from raw tokens using attention mechanisms that are diffuse, brittle, and sensitive to prompt formulation. As contexts grow longer and reasoning becomes more compositional, this repeated reconstruction becomes both inefficient and error-prone.

Existing context-management approaches only partially address this limitation. Long-context prompting improves coverage but dilutes attention and incurs high cost. Retrieval-augmented generation (RAG) (Lewis et al., 2020) retrieves relevant evidence but leaves integration and consistency tracking implicit. Summarization-based methods (Wu et al., 2021; Zhang et al., 2024) compress context but risk discarding critical details, while recent multi-agent and memory-control approaches (Yan et al., 2025; Yu et al., 2025) still rely largely on unstructured memory representations. Across these paradigms, the system lacks a persistent, structured working memory that explicitly captures accumulated knowledge. We make this concrete in § 2: existing benchmarks are thin in the regime that demands both broad input understanding and deep multi-step inference, and current methods that perform well on easier regimes fail there.

We propose MINDGRAPH (§ 3), a structured working-memory architecture based on an explicit concept graph. Instead of repeatedly reconstructing state from raw tokens or storing free-form summaries, MINDGRAPH incrementally builds a persistent graph whose nodes represent semantic

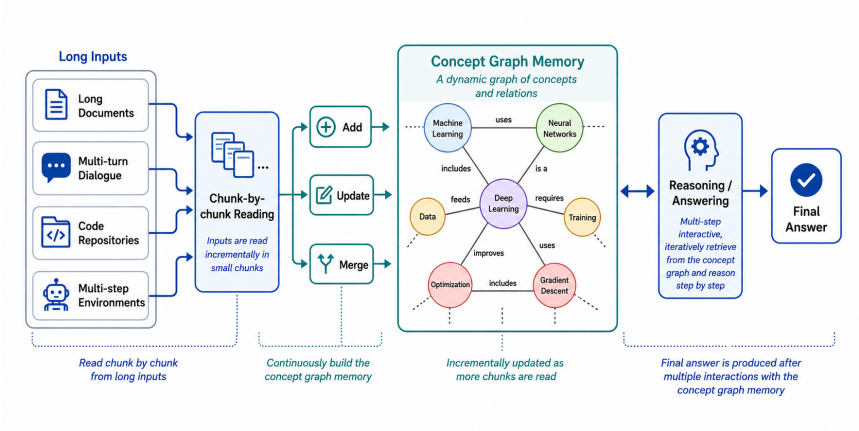

Figure 1. **Architecture of MindGraph.** The build stage streams over chunks and incrementally maintains a concept graph with character-level provenance to the original input; the answer stage reasons over the persistent graph using two tools, choosing its access pattern based on graph size.

units such as entities, claims, events, and constraints, while edges encode relationships such as support, contradiction, and causation. Each node is grounded to the original context through character-level provenance links, enabling faithful recovery and verification of supporting evidence.

MindGraph separates long-context reasoning into two stages. In the build stage, a question-aware builder streams over the input chunk by chunk and incrementally updates the graph, selectively preserving information relevant to downstream reasoning while maintaining explicit structure across distant spans. In the answer stage, the model reasons over the resulting graph using a tool loop that combines hierarchical graph navigation with fine-grained source verification. This allows reasoning to operate over a compact, persistent memory while still retaining access to the original evidence when needed. Both stages are implemented using prompted LLMs, making the framework model- and task-agnostic.

To evaluate this paradigm, we introduce LongReason-200 (§ 4.1), a benchmark of 200 long-context reasoning tasks filtered from LongBench-v2, NarrativeQA, and LooGLE to isolate settings requiring both broad evidence integration and multi-step inference. [1] We compare MindGraph against representative baselines spanning direct long-context prompting, RAG, summarization, multi-agent memory, and recursive retrieval. Existing methods struggle even with large context windows, highlighting the limitations of token-based state reconstruction. In contrast, MindGraph substantially improves reasoning reliability while maintaining compact and auditable working memory.

Overall, our results suggest that scaling context length alone is insufficient for reliable long-horizon reasoning. Instead, effective long-context systems require explicit, persistent, and structured working memory that supports global reasoning over accumulated knowledge.

[1] Released anonymously at https://anonymous.4open.science/r/LongReason-200/.

2. Motivation

This section motivates our work in two steps. We first analyze existing long-context benchmarks along two dimensions—reasoning breadth and reasoning depth (§ 2.1); we then characterize the failure modes of current methods on tasks that require both full-context understanding and intensive reasoning (§ 2.2). The benchmark we construct to study this regime directly is introduced in § 4.1.

2.1. Where Existing Benchmarks Live

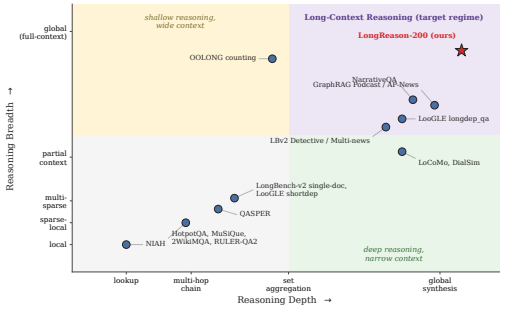

Figure 2. **Representative long-context benchmarks** situated in the (reasoning-depth, reasoning-breadth) plane. Existing benchmarks concentrate in the shallow-reasoning–wide-context (yellow) and deep-reasoning–narrow-context (green) regions; the upper-right *long-context reasoning* regime (purple)—global evidence integration coupled with multi-step inference—is only loosely covered. LongReason-200 (red star) targets this regime.

The difficulty of a long-context question arises from two largely independent demands: the *breadth* of input that must be understood, and the *depth* of inference required over it. A model can read the document fully and still misfire on

the inference chain; another can reason flawlessly over the wrong subset of evidence. We therefore treat the two as separate axes and use them as a per-item rubric throughout.

Reasoning breadth measures the fraction of the input the model must consult to answer correctly. We adopt a five-level scale: *local* (the answer lives in a single span of the input), *sparse-local* (a few nearby spans, typically within one section or paragraph cluster), *multi-sparse* (several spans scattered across distant parts of the input), *partial context reading* (a sizable but not whole-input slice must be read), and *global* (the input must be understood as a whole). Higher breadth increases the difficulty of locating and integrating relevant evidence, independently of inferential depth.

Reasoning depth measures the number of inferential steps that connect the gathered evidence to the answer. We use four levels: *lookup* (extract a verbatim span; no inference required), *multi-hop chain* (a small chain of inferential steps connects the evidence to the answer), *set aggregation* (count or aggregate over a set of items identified across the input), and *global synthesis* (combine many heterogeneous claims while weighing evidence and resolving contradictions). Higher depth increases the difficulty of deriving the answer, independently of reasoning breadth. Per-label diagnostics and worked toy examples for both axes are given in Appendix C.

Figure 2 situates representative long-context benchmarks in this plane. The bottom-left clusters at low depth and low breadth: needle-in-a-haystack, synthetic multi-hop QA (HotpotQA (Yang et al., 2018), RULER's QA2 (Hsieh et al., 2024), MuSiQue (Trivedi et al., 2022), 2WikiMultiHopQA (Ho et al., 2020)), QASPER (Dasigi et al., 2021), and LooGLE's `shortdep` (Li et al., 2024). Single-axis pockets push one dimension high: OOLONG counting (Bertsch et al., 2025) aggregates over the full input (high breadth, low depth); LoCoMo (Maharana et al., 2024) and DialSim (Kim et al., 2024) reason deeply over a partial slice (high depth, partial breadth). The upper-right is reached only by individual subsets of LongBench-v2 (Bai et al., 2025) (Detective, Multi-news) and LooGLE (Li et al., 2024) (`longdep_qa`), together with NarrativeQA (Kočiskỳ et al., 2018) and certain GraphRAG Podcast/AP-News items.

Takeaway 1: The upper-right corner (i.e., global evidence integration coupled with multi-step inference) is where existing benchmarks are thin: either absent, or diluted with easier items in the same split. This gap motivates both the failure-mode analysis in § 2.2 and the targeted benchmark we construct in § 4.1.

2.2. How Current Methods Fail

We hypothesize that existing long-context methods perform well when either reasoning breadth or reasoning depth remains limited, but degrade sharply once both become large simultaneously i.e., the upper-right corner of Figure 2. However, as stated in § 2.1, most existing long-context benchmarks lie outside this hardest regime. In particular, many tasks require either shallow reasoning over broad context or deeper reasoning over localized evidence. Under these settings, current approaches already achieve near-ceiling performance. For example, Table 1 shows that several long-context methods perform strongly on RULER's single-key needle-in-a-haystack benchmark at 128K context (Hsieh et al., 2024). We evaluate four representative approaches: long-context direct prompting (**Direct**, i.e., feeding the entire input to a vanilla LLM in a single call), summary-based generation (**Chain-of-Agents (CoA)** (Zhang et al., 2024)), multi-agent memory (**General-Agentic-Memory (GAM)** (Yan et al., 2025)), and recursive retrieval (**Recursive Language Model, RLM** (Zhang et al., 2025)). Despite differences in architecture and memory strategy, all four achieve near-ceiling accuracy in this setting, which primarily stresses reasoning breadth while requiring only shallow reasoning depth.

The picture changes in the upper-right corner. On the LongBench-v2 Detective sub-domain (Bai et al., 2025) (N=10 items, multiple-choice; whole-novel reasoning over distributed clues, alibis, and motives), the same four methods drop sharply (Table 1, lower row), well below their canonical-benchmark accuracy. We argue this is not a coincidence of any one method but a consequence of three failure modes shared across the design space. These are:

Failure mode 1: No persistent working memory. In some methods, no working memory of the input survives the query that produced it. Direct re-attends over raw tokens on every query; RLM operates on the input inside a Python REPL, so its working state is local variables and execution traces rather than an inspectable object that downstream queries can reuse. There is nothing to audit, reuse across queries, or improve incrementally: each query reconstructs the model's understanding from scratch, every call pays a cost linear in the full input length, and small reconstruction errors—an entity spelled inconsistently across chunks (the input segments a method processes one at a time), a contradiction not reconciled across documents—are paid again every time.

Failure mode 2: Unstructured working memory loses information and accumulates contradictions. A persistent working memory must also be *compact* to be cheaply reusable on every query, so summary-based methods such as Chain-of-Agents compress the input into a rolling prose summary. But plain text has no internal structure to distinguish a claim from an entity, a fact from an opinion, or

Table 1. **Accuracy of representative long-context methods on a low-depth and a high-depth task.** Top row (RULER NIAH-S1, 128K): Direct/GAM/RLM cells are taken from each method's published evaluation; the CoA cell is our measurement (CoA was not evaluated on NIAH in its original paper (Zhang et al., 2024)). Bottom row (LongBench-v2 Detective): all four cells are our runs on a random sample of 10 items from this sub-domain.

Task	Direct	CoA	GAM	RLM
RULER NIAH-S1, 128K context (Hsieh et al., 2024)	∼99	∼99	∼96 (Yan et al., 2025)	∼100 (Zhang et al., 2025)
LongBench-v2 Detective, N=10	20%	30%	50%	10%

an unresolved hypothesis from a confirmed conclusion; under a fixed length budget, compression therefore becomes destructive. When successive chunks introduce contradictory evidence, the summary silently drops one side or holds both as adjacent sentences with no flag of conflict. The loss is one-way: a fact compressed out at chunk N cannot be recovered when it becomes relevant at chunk $N{+}k$, even though it was once in the input.

Failure mode 3: Even structured working memory is often not reasoning-friendly. Methods that do construct a structured memory typically optimize for extraction breadth rather than reasoning fitness. Extraction-style entity graphs, for example, yield flat NER-style structures in which every literal string in the text becomes a node and edges encode within-chunk co-occurrence; the relations such graphs record (named-entity adjacency) do not directly support compositional inference, multi-hop chaining, or the weighing of competing evidence that the upper-right regime demands. The structure exists, but it is not the structure reasoning needs: claims, concepts, and their semantic relations have to be first-class in the graph, so that multi-hop chaining and conflict-weighing become operations on the structure itself rather than free-form inference over surface adjacency.

Takeaway 2: These failures share a single root: no current method holds a working memory that is simultaneously persistent (so reasoning need not start over), structured (so claims, entities, and their relationships are first-class primitives), and reasoning-friendly (so the operations the regime requires are well-supported). The architecture we present in § 3 is designed to provide exactly this.

3. Our Approach: MINDGRAPH

We consider the standard long-context QA setting, where the system receives a *(question, context)* pair with the question presented first. Most existing approaches can be viewed as instantiating the same two-stage abstraction: a *build stage* that converts the context into some working memory, followed by an *answer stage* that reasons over that memory. The two stages may be tightly coupled into a single pass (e.g., Direct), or explicitly separated (e.g., CoA, GAM, GraphRAG, and RLM).

Under this view, long-context QA reduces to two coupled design questions: **(RQ1)** *what working memory should the build stage produce?* and **(RQ2)** *how should the answer stage use it?* These questions are inseparable. A working memory that is theoretically sufficient is ineffective if the answer stage cannot navigate or verify it; conversely, even a sophisticated reasoning procedure cannot recover evidence that the memory failed to preserve.

The structure of the task itself constrains what an effective working memory must provide. The "breadth" feature, where relevant evidence is distributed across the full context, demands a working memory that is *compact* enough to reuse efficiently across queries and *persistent* enough to avoid repeatedly reprocessing the entire input. In contrast, the "depth" feature, where solving the task requires multi-step inference over dispersed evidence, demands memories that are *structured*, so that chaining, aggregation, and conflict resolution operate directly over the memory representation rather than emerging implicitly from free-form generation. [2] Finally, the ordering of the input provides an additional opportunity for compactness. Because the question is available before context processing begins, the build stage can construct a more selective, question-aware memory instead of encoding the entire document uniformly.

We propose MINDGRAPH, which answers RQ1 with a question-aware, *incrementally constructed concept graph* (§ 3.1–§ 3.2) carrying character-level provenance back to the original input, and answers RQ2 with an LLM-driven tool loop (§ 3.3) combining fine-grained verification (`lookup_source`) and hierarchical navigation (`subgraph_summary`). The graph is *persistent* (built once, queried repeatedly), *structured* (typed nodes, labeled relations, explicit provenance), and *reasoning-friendly* (lookup, multi-hop chaining, and verification are first-class operations)—directly resolving the three failure modes of § 2.2. The system is model- and task-agnostic: build and answer can use any LLMs (the same or different), no fine-tuning is required, and the only task-specific input is the question.

[2] These requirements closely mirror the failure modes identified in § 2.2.

3.1. Concept-Graph Working Memory

MINDGRAPH represents the input as a directed labeled graph $G = (V, E)$, built incrementally by a build LLM that streams over the input chunk by chunk (§ 3.2). Each node $v \in V$ carries (i) a *type* from a closed vocabulary $\{\text{entity}, \text{event}, \text{claim}, \text{concept}, \text{stat}\}$—an *entity* is a person, object, or location; an *event* an action or occurrence; a *claim* a proposition asserted or inferred from the text; a *concept* an abstract or aggregate theme; a *stat* a quantitative fact—(ii) *content*, a short natural-language description, (iii) *provenance*, a $(\text{start}, \text{end})$ pair indexing into the original input and derived by locating an LLM-supplied verbatim quote in the source, and (iv) the index of the chunk in which the node was introduced. Each edge $e = (v_i \xrightarrow{\text{relation}} v_j) \in E$ is a labeled directed relation with its own provenance and originating chunk; edge labels are open-vocabulary strings (e.g., *caused_by*, *contradicts*, *supports*) chosen by the build LLM rather than drawn from a closed schema.

3.2. Build: Incremental, Question-Aware Construction

The build stage processes the input as an ordered sequence of chunks $\{c_1, \ldots, c_T\}$ obtained by token-bounded splitting. Starting from the empty graph $G_0 = (\emptyset, \emptyset)$, the build LLM is invoked T times: at iteration t it sees the question, the current graph G_{t-1}, and chunk c_t, and returns a list of graph edits that are applied to obtain G_t. The exact build-stage prompt used in our runs is given in Appendix D.1. Concretely, the edit operations are $\{\texttt{add\_node}, \texttt{add\_edge}, \texttt{edit\_node}, \texttt{delete\_node}\}$, and every edit carries a verbatim quote from the current chunk, which is resolved to a character-level provenance offset on the original input. Two properties of this design matter.

Incremental. Because the build LLM, at iteration t, sees G_{t-1}, it can *merge* new evidence into existing nodes via `edit_node`, *retract* earlier mistakes via `delete_node`, and add cross-chunk edges that were impossible to state when either endpoint was first introduced. Coreference and inter-claim relationships are therefore resolved *as the graph is built*, not re-derived on every query. This is what gives the working memory its persistence: G_t always reflects the content seen so far, and is continuously updated as new chunks arrive.

Question-aware. The question is in the build prompt, biasing extraction toward question-relevant content. This conditioning is the reason a 300K-token novel can yield a graph of fewer than 50 nodes when the question targets a single character arc, and a much denser graph when the question requires whole-cast tracking; we report graph sizes against context length in § 5.

3.3. Answer: Reasoning over the Graph

At answer time, an answering LLM is given the question and G_T in JSON form, together with summary statistics (node count, edge count, build-chunk count). It has access to two tools (the full answer-stage prompt and the function-calling tool descriptions are reproduced verbatim in Appendix D.2–D.3):

- `lookup_source(node_id)` returns approximately 1K characters of the original input centered on the node's provenance span. This is the verification primitive: when the answer LLM has narrowed in on a candidate claim, it can read the surrounding source text rather than relying on the node's content string alone.
- `subgraph_summary(mode, subgraph_id)` returns a navigation aid for large graphs. Following GraphRAG (Edge et al., 2024), we partition G_T into densely-connected subgraphs by Leiden clustering and, on first access, generate one summary per subgraph with an LLM call. With `mode="index"`, the tool returns a one-line entry for every subgraph (subgraph id, title, impact rating, number of nodes); with `mode="detail"` and a specific id, it returns the full multi-finding summary for that subgraph. Partitions and reports are cached per graph hash, avoiding recomputation.

3.4. Compaction

The graph compresses the input substantially. Empirically the answer-stage prompt receives a graph that is on average a small fraction of the original-input token count, an order-of-magnitude reduction relative to feeding the full context (full numbers in § 5). Two properties of this compaction are worth flagging here. First, structure is preserved: unlike a flat textual summary, G_T retains explicit node types, edge labels, and provenance, so multi-step reasoning over the graph remains compositional. Second, compaction is lossy: a fact that the build LLM deemed off-topic and did not extract is genuinely gone. But the loss is bounded by `lookup_source`: a fact that lies in the source text near the provenance of any retained node remains recoverable, even if it never made it into the graph. The compaction ratio—the fraction of original-input tokens that the answer LLM sees on its first call (formal definition in § 4.2)—thus understates the effective coverage of the working memory, which we quantify against the original input in § 5.

4. Setup

We now describe our experimental setup. § 4.1 introduces LONGREASON-200, the targeted benchmark we construct for long-context reasoning, along with its filtering criteria and length statistics. § 4.2 describes the five baselines we compare against MINDGRAPH and our evaluation metrics.

4.1. Benchmark: LONGREASON-200

We refer to the upper-right quadrant of Figure 2 as *long-context reasoning*: tasks in which reaching the answer requires full-context understanding and composing, comparing, or weighing that evidence over multiple inferential steps. As §2.1 shows, existing long-context benchmarks either dilute hard items with easy ones or specialize in a single axis, leaving this regime only loosely covered. To study it directly, we construct LONGREASON-200 (released anonymously; see footnote in §1), a 200-item benchmark drawn from LongBench-v2 (Bai et al., 2025) (62 items), NarrativeQA (Kočiskỳ et al., 2018) (118), and LooGLE (Li et al., 2024) (20).

Filtering. For each source we keep only items at the global end of the reasoning-breadth axis—annotated *mostly global* (most of the input must be consulted) or *truly global* (the full input must be integrated)—and whose reasoning depth is *global synthesis* or *set aggregation*, with no code-related or summarization tasks, and a per-source length floor: $\geq$ 80,000 estimated tokens for LongBench-v2, $\geq$ 120,000 words of source story for NarrativeQA, and $\geq$ 50,000 estimated tokens for LooGLE. The metadata labels are produced by an LLM judge and post-processed for consistency; we keep only items whose labels are stable across two judging passes. Per-label definitions and toy examples for the two axes are given in Appendix C.

Statistics. By question format, 62 items are multiple-choice and 138 are free-text. Context length (i.e., estimated GPT tokens) has mean 179K, median 159K, range 56K–469K.

4.2. Evaluation

Methods. We compare MINDGRAPH to five baselines representing the major paradigms in §2.2: Direct, GraphRAG (Edge et al., 2024), Chain-of-Agents (Zhang et al., 2024), General-Agentic-Memory (Yan et al., 2025), and RLM (Zhang et al., 2025). All methods use `gpt-5.4` as the backbone LLM with high reasoning effort and, where applicable, a build/chunk size of 8,192 tokens; Leiden clustering, where used, runs with `max_cluster_size` = 10 and a fixed seed. Among them, GraphRAG uses one gleaning pass and `top_k_reports` = 8; General-Agentic-Memory uses `gpt-4.1-mini` for ingestion (whose call count per item is significantly higher than that of the other methods); RLM caps REPL iterations at 30; and MINDGRAPH caps tool rounds at 40 with `lookup_source` and `subgraph_summary` as the available tools, with subgraph reports generated lazily on first access and cached per graph.

Accuracy. For multiple-choice items we compare the predicted letter to the gold letter. For free-text items, an LLM judge (`gpt-5.4-mini`) issues a binary `correct`/`incorrect` verdict per item, given the gold answer, the predicted answer, and the question (full judge prompt in Appendix D.4). Judge is strict on list-type answers (rejects partial set matches) and is calibrated against a small human-graded subset. Accuracy is reported as the per-item correctness aggregated by mean.

Table 2. **Main results on LONGREASON-200**. Compaction ratio reported as a percentage of the original input.

Method	Acc. (%)	Comp. (%)
Direct	64	∼100
GraphRAG (Edge et al., 2024)	37	∼1
CoA (Zhang et al., 2024)	55	∼1
GAM (Yan et al., 2025)	62	∼8
RLM (Zhang et al., 2025)	20	∼3
MINDGRAPH (ours)	**77**	∼7

Compaction ratio. For an item with original-input token count L, let f_1 be the input-token count of the answering LLM's first call. We define compaction ratio $= f_1/L$, expressed as a percentage; smaller values indicate stronger compression. This captures the per-query cost in the multi-query setting where any build artifact is reused.

5. Results

We organize the section around four questions:

1. How does MINDGRAPH perform on LONGREASON-200?(§ 5.1)
2. How does the (compaction ratio, accuracy) relationship look across methods? (§ 5.2)
3. What graphs does MINDGRAPH build, and how do they scale with input length? (§ 5.3)
4. How does the answer stage leverage these graphs at query time? (§ 5.4)

5.1. Main Results

MINDGRAPH reaches 77% accuracy on LONGREASON-200, 13 points above Direct's 64% while passing only ∼7% of the input to the answering LLM on its first call—roughly 14× less per query (Table 2). Among the five baselines, the next strongest are General-Agentic-Memory (62%) and Chain-of-Agents (55%); GraphRAG and RLM trail at 37% and 20%. MINDGRAPH's margin over the strongest non-Direct baseline is 15 points. MINDGRAPH thus attains both the highest accuracy and a sub-10% per-query input cost—a pairing no baseline matches; we examine the full picture in § 5.2.

5.2. The Compaction–Accuracy Relationship

Figure 3 places each method in the (accuracy, compaction ratio) plane. Two endpoints anchor the plane: Direct passes

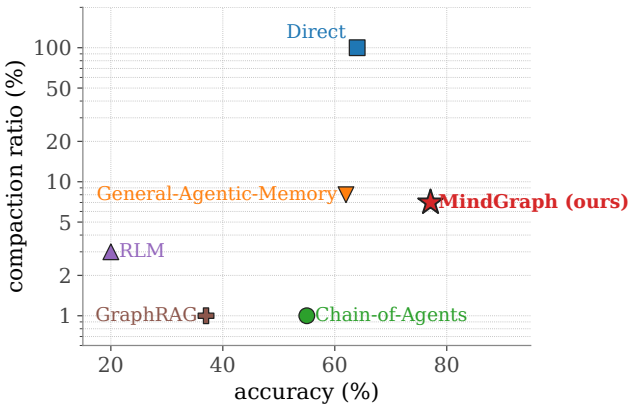

Figure 3. **Accuracy vs. compaction ratio on LONGREASON-200.** One point per method.

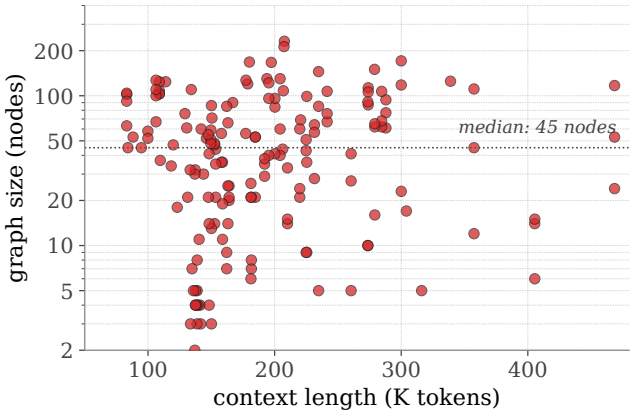

Figure 4. **MINDGRAPH graph size vs. context length on LONGREASON-200.** Pearson r=0.08; median 45 nodes.

the full context on every call (∼100% compaction ratio) and obtains the strongest *baseline* accuracy, while Chain-of-Agents compresses to ∼1% via a rolling summary and matches accuracy on items reducible to a single dominant narrative thread. MINDGRAPH sits to the upper-left of both: at ∼7% compaction ratio it exceeds Direct's accuracy by 13 points and Chain-of-Agents's by 22 points, and it dominates Direct on both axes simultaneously. MINDGRAPH and Chain-of-Agents jointly trace the Pareto frontier; the remaining methods sit below it.

Structure, not compaction ratio, drives the gap among memory-based methods. General-Agentic-Memory compresses to ∼8%, almost identical to MINDGRAPH's ∼7%, yet trails by 15 points in accuracy (62% vs. 77%). Chain-of-Agents and GraphRAG compress more aggressively (∼1%) but accuracy drops to 55% and 37%; RLM at ∼3% compaction ratio is the weakest in the table at 20%, because the REPL state carries no structured working memory of it. The compaction ratio alone is not the signal: methods that compress without preserving structure end up on or below the lower envelope of the figure. The methods on the Pareto frontier are precisely those whose compressed memory keeps the relations among claims first-class—a rolling summary that retains a narrative thread (Chain-of-Agents) at one end, a typed concept graph with provenance (MINDGRAPH) at the other.

5.3. Graph Properties

Figure 4 plots graph size against context length for every MINDGRAPH item. Graph size stays below ∼200 nodes (median 45) and is essentially uncorrelated with input length (Pearson r=0.08): a 469K-token novel yields a graph of roughly the same size as an 83K-token excerpt. This bounded growth is the property that makes the working memory practical: as the input scales from tens of thousands to hundreds of thousands of tokens, the object the answer-stage LLM has to read does *not* scale with it, so the long-context bottleneck the build stage was meant to relieve does not reappear inside the graph.

5.4. Leveraging the Graph at Answer Time

Routing between `lookup_source` and `subgraph_summary` is implemented entirely through prompt instructions (§ 3.3); in practice the LLM honors the size-based guidance without any code-level branching. On items with < 50 nodes (median graph size 20), the LLM never invokes `subgraph_summary` and verifies with a median of 4 `lookup_source` calls. On medium graphs (50–150 nodes, median 85), 42% of items invoke `subgraph_summary` at least once, and the median lookup count rises to 9. On large graphs ($\geq$150 nodes, median 170), 67% invoke `subgraph_summary`, with a median of 1 index call followed by 3 detail calls before drilling into specific nodes with ∼12 lookups. Two-tool access scales: small graphs are read directly, large graphs are navigated hierarchically, and the LLM picks between the two based on the graph statistics in the prompt rather than a hand-coded threshold.

6. Related Work

Long-context language models. Architectural and systems advances—efficient attention, position interpolation, and memory-augmented transformers (Dao et al., 2022; Press et al., 2022; Chen et al., 2023a; Wu et al., 2022)—have pushed commercial context windows to hundreds of thousands or millions of tokens (Singh et al., 2025). Yet models still fail to utilize distant or distributed information (Liu et al., 2024), motivating structured working memory rather than longer contexts alone.

Retrieval-augmented generation. RAG augments LLMs with external memory by retrieving documents at infer-

ence time (Lewis et al., 2020), with extensions for iterative or graph-based retrieval such as GraphRAG and HippoRAG (Edge et al., 2024; Gutiérrez et al., 2024). These methods improve recall but treat retrieved evidence as unstructured text, leaving integration to the model and risking redundancy or unresolved conflicts.

Summarization and context compression. Recursive summarization and chain-of-agents pipelines compress long inputs while attempting to preserve salient information (Wu et al., 2021; Zhang et al., 2024), but tend to discard fine-grained details, mask contradictions, and lose provenance to the source.

Memory-augmented and agent-based systems. Earlier memory mechanisms include key-value memories, retrieval-conditioned models, and learned controllers (Weston et al., 2015; Graves et al., 2016; Borgeaud et al., 2022); recent agent frameworks add memory modules to persist information across interactions (Park et al., 2023; Shinn et al., 2023; Yu et al., 2025). These systems improve long-horizon performance but typically rely on text buffers or embeddings, limiting their ability to enforce consistency or support global reasoning.

Structured representations for reasoning. Knowledge graphs offer explicit entities and relations and have been combined with LLMs for reasoning and retrieval (Nickel et al., 2016; Hogan et al., 2021; Sun et al., 2024; Luo et al., 2024), but usually over static or externally constructed graphs. A complementary line imposes structure on the reasoning trace itself, including chain-of-thought (Wei et al., 2022), program-of-thought (Chen et al., 2023b), and tool-augmented prompting (Schick et al., 2023), yet such traces are transient and not reused across inputs. MINDGRAPH differs from all of the above: it dynamically constructs a task-specific concept graph from unstructured input, maintaining character-level provenance and incremental updates under a bounded budget, thereby providing the persistent, structured, and auditable working memory that prior approaches lack.

7. Conclusion

We argued that the failure of current long-context methods on tasks that simultaneously require global access and multi-step inference can be located in a single object: the working memory that the method holds during reasoning. None of the representative paradigms we surveyed—long-context prompting, retrieval-augmented generation, summary-based generation, multi-agent memory, and recursive REPL agents—holds a working memory that is at once persistent, structured, and reasoning-friendly.

We proposed MINDGRAPH, a system designed to close this gap. It maintains an incrementally constructed concept graph with character-level provenance back to the original input, and exposes two complementary tools to the answer-stage LLM: fine-grained verification against the source text, and hierarchical navigation of large graphs. The system is model- and task-agnostic: any LLMs can be used at the build and answer stages, and the only task-specific input is the question itself. To evaluate this regime in a controlled way, we introduced LONGREASON-200, a 200-item benchmark filtered to require global evidence integration and multi-step reasoning.

Across the tasks studied, MINDGRAPH matches or improves on the strongest baselines while passing only a small fraction of the original input to the answering LLM on each query. More broadly, our findings suggest that scaling context length alone is insufficient: future long-context systems should treat the working memory—not just the model or the prompt—as a first-class design dimension, and should answer two coupled questions jointly: what working memory to maintain, and how the answer stage should leverage it. We discuss limitations in Appendix A and broader impacts in Appendix B.

References

Bai, Y., Tu, S., Zhang, J., Peng, H., Wang, X., Lv, X., Cao, S., Xu, J., Hou, L., Dong, Y., et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. pp. 3639–3664, 2025.

Bertsch, A., Pratapa, A., Mitamura, T., Neubig, G., and Gormley, M. R. Oolong: Evaluating long context reasoning and aggregation capabilities. *arXiv preprint arXiv:2511.02817*, 2025.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J.-B., Damoc, B., Clark, A., et al. Improving language models by retrieving from trillions of tokens. *Proceedings of the 39th International Conference on Machine Learning*, 2022.

Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023a.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023b.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35, 2022.

Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. 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 (NAACL)*, 2021.

Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., Metropolitansky, D., Ness, R. O., and Larson, J. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Gutiérrez, B. J., Shu, Y., Gu, Y., Yasunaga, M., and Su, Y. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569, 2024.

Ho, X., Nguyen, A.-K. D., Sugawara, S., and Aizawa, A. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, 2020.

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., Kirrane, S., Labra Gayo, J. E., Navigli, R., Neumaier, S., et al. Knowledge graphs. *ACM Computing Surveys*, 54(4), 2021.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Kim, J., Chay, W., Hwang, H., Kyung, D., Chung, H., Cho, E., Jo, Y., and Choi, E. DialSim: A real-time simulator for evaluating long-term multi-party dialogue understanding of conversational agents. *arXiv preprint arXiv:2406.13144*, 2024.

Kočiskỳ, T., Schwarz, J., Blunsom, P., Dyer, C., Hermann, K. M., Melis, G., and Grefenstette, E. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 2020.

Li, J., Wang, M., Zheng, Z., and Zhang, M. Loogle: Can long-context language models understand long contexts? pp. 16304–16333, 2024.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12, 2024. arXiv:2307.03172.

Luo, L., Li, Y.-F., Haffari, G., and Pan, S. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *ICLR 2024: The Twelfth International Conference on Learning Representations*. ICLR, 2024.

Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F., and Fang, Y. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Nickel, M., Murphy, K., Tresp, V., and Gabrilovich, E. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. *International Conference on Learning Representations*, 2022.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2023.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023.

Singh, A., Fry, A., Perelman, A., Tart, A., Ganesh, A., El-Kishky, A., McLaughlin, A., Low, A., Ostrow, A., Ananthram, A., et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Sun, J., Xu, C., Tang, L., Wang, S., Lin, C., Gong, Y., Ni, L. M., Shum, H.-Y., and Guo, J. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *International Conference on Learning Representations*, 2024. arXiv:2307.07697; commonly cited as "sun2023kg".

Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35, 2022.

Weston, J., Chopra, S., and Bordes, A. Memory networks. *International Conference on Learning Representations*, 2015. arXiv:1410.3916.

Wu, J., Ouyang, L., Ziegler, D. M., Stiennon, N., Lowe, R., Leike, J., and Christiano, P. Recursively summarizing books with human feedback. *arXiv preprint arXiv:2109.10862*, 2021.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

Yan, B., Li, C., Qian, H., Lu, S., and Liu, Z. General agentic memory via deep research. *arXiv preprint arXiv:2511.18423*, 2025.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Yu, H., Chen, T., Feng, J., Chen, J., Dai, W., Yu, Q., Zhang, Y.-Q., Ma, W.-Y., Liu, J., Wang, M., et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025.

Zhang, A. L., Kraska, T., and Khattab, O. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025.

Zhang, Y., Sun, R., Chen, Y., Pfister, T., Zhang, R., and Arık, S. Ö. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

A. Limitations

The graph inherits whatever the build LLM extracts: missing claims, mis-merged entities, or noisy edges all propagate to the answer stage, and accuracy degrades on items whose graphs grow to several hundred nodes even with `subgraph_summary` navigation. All stages of the system are prompted rather than learned, so the obvious next step—learned policies for build operations, pruning, and tool routing—remains future work. Conditioning the build on the query also keeps graphs small but ties each graph to a specific question, so multi-query workloads require either rebuilding or a query-distribution prompt; and while first-call compaction is small, the cumulative cost across many tool calls can approach the original input on tool-heavy items, with which cost matters depending on the deployment regime.

B. Broader Impacts

MINDGRAPH is a building block for systems that reason over long inputs. Two classes of impact are worth flagging.

Positive impact. A persistent, auditable graph memory makes the path from a system's answer back to its source evidence inspectable: every node carries a character-level pointer to the input span that licensed it, and the answer stage can ground each claim in the surrounding source text via `lookup_source`. This property is useful in high-stakes settings where downstream readers must verify the system's reasoning, such as legal review, scientific synthesis, and long-form fact-checking. The compaction MINDGRAPH achieves also lowers per-query inference cost in multi-query workloads, reducing energy use and broadening access to long-context reasoning capabilities.

Risks and mitigations. The graph is built by an LLM and is no more reliable than the underlying extraction. Hallucinated claim nodes, mis-merged entities, and unfaithful aggregate summaries propagate to downstream answers, with the additional risk that a structured presentation can lend unwarranted credibility to incorrect content. Provenance pointers partially mitigate this, but they do not detect hallucinations automatically; we recommend that deployments expose source spans alongside generated answers and treat unverifiable claims with appropriate caution. Systems built on MINDGRAPH that operate on sensitive inputs (medical records, personal communications) inherit the privacy considerations of those inputs: the graph encodes the same content the input does, and any sharing of the graph carries the same considerations as sharing the source text.

C. Detailed Taxonomy of Reasoning Breadth and Depth

§2.1 introduces the two axes we use to characterize a long-context QA item—reasoning *breadth* (how much of the input must be consulted) and reasoning *depth* (how many inferential steps connect the gathered evidence to the answer). This appendix expands each axis: we restate every label using the same name as in the main text, give a diagnostic for when an item should be assigned that label, and accompany it with a minimal toy example. Examples are deliberately schematic so that the level—not the domain—is what distinguishes them; real items typically combine multiple breadth or depth signals at once.

C.1. Reasoning Breadth

Breadth is a property of the *evidence support* of a question: the smallest subset of the input whose removal would change the answer. We use a five-level scale, ordered from least to most demanding.

local. The answer lives in a single span of the input; once that span is located, no further reading is required. Locating it can still be hard in a very long document, but the inferential surface area is one span. Length-independent of the context.

> **Context (excerpt).** "... Alice is Bob's father. ..."
> **Question.** Who is Alice's son?
> **Answer.** Bob.
> **Why local.** A single sentence licenses the answer; the rest of the document is irrelevant.

sparse-local. A small number of supporting spans must be consulted, but they sit close together—typically within the same paragraph, scene, or short section—so a competent skim of one locality recovers all of them. The evidence is multi-span but *spatially clustered*.

> **Context (excerpt, single paragraph).** "Alice is Bob's father. Bob is Carol's father. ..."
> **Question.** Who is Carol's grandfather?
> **Answer.** Alice.
> **Why sparse-local.** Two supporting facts are required, but they appear adjacent; locating one localises the other.

multi-sparse. Several supporting spans are required and they are scattered across distant parts of the input (e.g. different chapters, sections, or documents). The model must locate multiple non-co-located evidence pieces before any inference can begin. This is the regime that classic multi-hop QA stresses.

> **Context (long document).** Chapter 2 introduces "Alice is the mother of the village's blacksmith." Chapter 11, hundreds of pages later, names "the blacksmith's son Daniel."
> **Question.** What is Daniel's relationship to Alice?
> **Answer.** Alice is Daniel's grandmother.
> **Why multi-sparse.** The two facts that license the answer are far apart; both must be retrieved, but most of the document is still irrelevant.

partial context reading. A sizable but not whole-input slice must be read coherently—an entire section, sub-document, or thematically connected region—rather than a few spans. Skimming for keywords is not sufficient; the model needs to follow the slice's internal structure (e.g. the methods section of a paper, or one character's chapters in a multi-POV novel).

> **Context.** A long scientific paper.
> **Question.** Which data-collection methods does the study use?
> **Answer.** (Whatever the Methods section reports.)
> **Why partial.** The answer is concentrated in one section, but that section is itself substantial and must be read in full—no single sentence captures the methods, and the rest of the paper (intro, related work, results, discussion) is largely irrelevant.

global. The input must be understood as a whole: the answer depends on integrating evidence drawn from the full document, or on a property of the document at the document level (consistency, overall theme, set membership over the entire corpus). No proper subset of the input is sufficient.

Context. A grammar and lexicon defining a new constructed language, distributed across an entire document.
Question. Translate the English sentence "the bird flew over the river" into the new language.
Answer. (A sentence whose lexicon entries and grammar rules each come from different parts of the document.)
Why global. Almost every word and rule in the document contributes to forming a correct translation; one cannot answer without holding the whole language in working memory.

The filtering step in §4.1 keeps only items annotated as *mostly global* (most of the input must be consulted, i.e. *partial context reading* or *global*) or *truly global* (the full input must be integrated, i.e. *global*).

C.2. Reasoning Depth

Depth is a property of the *inference chain* that connects the gathered evidence to the final answer, conditional on having the evidence already in hand. It is largely independent of breadth: an item can require deep inference over a single span (high depth, low breadth) or shallow aggregation over the whole input (low depth, high breadth).

lookup. The answer is a verbatim or near-verbatim span of the gathered evidence. No inferential step is required beyond pointing at the right substring.

Evidence. "Alice is Bob's father."
Question. Who is Alice's son?
Answer. Bob.
Why lookup. The answer can be read directly off the evidence span—no transformation, no chaining.

multi-hop chain. A small chain of inferential steps—typically two or three—connects the evidence to the answer. Each step is itself a lookup-grade transformation, but the chain has to be assembled.

Evidence. "Alice is Bob's father." "Carol is Bob's daughter."
Question. Who is Carol's grandfather?
Answer. Alice.
Why multi-hop chain. *father(·, Bob)=Alice* and *daughter(·, Bob)=Carol* must be composed via the intermediate entity Bob; the answer is one short inferential chain away.

set aggregation. The answer is a count, sum, majority, or other aggregate over a *set of items* that must first be identified across the input. Each individual identification is shallow, but the answer requires producing a quantity or label defined over the whole set, not over any one item.

Context. A collection of email messages, each tagged or described with a category.
Question. Which category appears most often in the collection?
Answer. (The mode of the per-message categories.)
Why set aggregation. Per-email classification is shallow, but the final answer is an argmax over a set; it cannot be extracted from any single item.

global synthesis. The answer requires combining many heterogeneous claims drawn from across the input while *weighing* evidence and *resolving* contradictions (e.g. unreliable narrators, conflicting witness statements, hypotheses to be confirmed or rejected). Unlike multi-hop chain, the graph of relevant evidence is not a short path but a web; unlike set aggregation, the items combined are not homogeneous.

Context. A detective novel: scattered clues, witness testimony, alibis, motives, red herrings, and authorial misdirection. The text never names the murderer outright.
Question. Who is the murderer?
Answer. (The character whose combined motive, opportunity, and physical evidence is consistent with the full set of clues, after discounting unreliable testimony.)
Why global synthesis. The answer depends on integrating heterogeneous evidence from many parts of the text and on resolving explicit conflicts; no chain of length two or three suffices, and no single aggregate captures it.

The filtering step in §4.1 keeps only items at the upper end of this scale—*set aggregation* or *global synthesis*—which, combined with the breadth filter, isolates the upper-right regime of Figure 2 that motivates LONGREASON-200.

D. Prompts Used by MINDGRAPH

This appendix lists, verbatim, the prompts used by MINDGRAPH for the runs reported in §5. The same prompts are used across all backbones; only the model identifier changes. Placeholders in curly braces (e.g., {question}, {graph_state}) are filled by the harness at call time.

D.1. Build-Stage Prompt

The build LLM is invoked once per chunk with the prompt below. At iteration t, {graph_state} holds the JSON serialisation of G_{t-1} (or the literal string (empty) when t=1), {block_text} is chunk c_t, and {block_idx} / {total_blocks} expose the chunk position so the LLM can budget edits over the remaining context.

```
You are a mind-graph builder. You read text blocks one at a time and maintain
a concept graph that captures information needed to answer the question below.

## Question to answer later
{question}

## Current graph state
{graph_state}

## New text block (block {block_idx}/{total_blocks})
"""
{block_text}
"""

## Your task
Propose updates to the graph. Output **only** a JSON object:

{"operations": [
  {"op": "add_node", "id": "short_id", "type": "entity|event|claim|concept|stat",
   "content": "description", "src": "exact quote from text"},
  {"op": "add_edge", "source": "id1", "target": "id2",
   "relation": "relationship", "src": "exact quote"},
  {"op": "edit_node", "id": "existing_id", "content": "updated description"},
  {"op": "delete_node", "id": "old_id"}
]}

Rules:
- Focus on information relevant to the question.
- Merge / update existing nodes when new text refines them.
- The `src` field must be an **exact, verbatim** substring copied from the text
  block (used to locate the source passage).
- Use readable snake_case ids (e.g. "matt_mercer", "total_nat20s").
- Output ONLY valid JSON. No commentary.
```

D.2. Answer-Stage Prompt

The answer LLM is given the prompt below on its first turn, together with the function-calling schemas for lookup_source and subgraph_summary (§D.3). {graph_state} is the JSON serialisation of G_T; {graph_stats} is a short human-readable string of node count, edge count, and build-chunk count (e.g. "72 nodes, 154 edges, built from 21 chunks").

```
Answer the question using the mind graph below. Work through four phases.

## Mind Graph
{graph_state}

## Graph stats
{graph_stats}

## Question
{question}
```

```
## Tools available
You have two tools. Choose based on graph size and question type.

- `lookup_source(node_id)` -- retrieve the original passage (~1000 chars)
  around a node's source quote. Best for verifying specific facts and
  resolving ambiguity at the level of individual nodes.

- `subgraph_summary(mode, subgraph_id?)` -- get a structural overview of
  the graph organized into Leiden subgraphs. Call it with
  `mode="index"` first to see a compact list of
  `(subgraph_id, title, impact_rating, n_nodes)` for every subgraph.
  Then call with `mode="detail"` and a specific `subgraph_id` to get
  that subgraph's full executive summary and findings. Best when the
  graph is too large to survey node-by-node and you need a global map
  before drilling in.

## How to choose
Use the graph stats above to decide your strategy:
- **< 50 nodes**: skip `subgraph_summary`; the whole graph fits in your
  attention. Go straight to Phase 1 and use `lookup_source` when you need
  to verify.
- **50 - 150 nodes**: use judgment. If the question targets a specific
  entity you can locate by eye, `lookup_source` is enough. If the question
  requires integrating evidence across multiple regions, call
  `subgraph_summary(mode="index")` first to map the terrain.
- **> 150 nodes**: start with `subgraph_summary(mode="index")`, pick the
  1-3 subgraphs most relevant to the question, fetch each with
  `mode="detail"`, then use `lookup_source` on specific nodes inside those
  subgraphs to verify key quotes.

## Phase 1 -- Global understanding
Before doing any lookups, build a picture of the graph as a whole:
- For small graphs, read the nodes and edges directly. Start from hub
  nodes (many edges) -- they usually represent central entities, themes,
  or events that anchor the narrative.
- For large graphs, rely on `subgraph_summary(mode="index")` and,
  selectively, `mode="detail"`. Treat the subgraph titles and ratings as
  a table of contents.
- Trace the main storylines, argument chains, or causal sequences that
  connect hubs/subgraphs.
- Build a mental summary: what is this context fundamentally about? What
  are the key relationships, conflicts, or conclusions?

## Phase 2 -- Locate relevant subgraph
Now relate the question back to the graph:
- Identify which region(s) of the graph contain the information needed to
  answer the question. This may be a single cluster of nodes, multiple
  scattered regions, or even the entire graph if the question requires
  holistic understanding.
- Use `lookup_source` on the most relevant nodes to read the original
  text and ground your understanding in primary evidence. Prioritize
  nodes that sit at the intersection of the question's key concepts.
- If the question requires integrating information across distant parts
  of the graph, look up nodes from each relevant region.

## Phase 3 -- Answer
Synthesize the evidence gathered from the relevant subgraph(s).
- Ground your answer in the looked-up source passages, not just node
  summaries.
- If information from multiple regions is needed, explain how they connect.
- Be precise and concise.

## Phase 4 -- Reflect
Before finalizing, verify your answer against the graph:
```

```
- Does your answer contradict any node or edge in the graph?
- Are there nodes with evidence that conflicts with your conclusion? If
  so, address the conflict explicitly.
- Did you overlook any region/subgraph of the graph that the question
  might depend on?

If reflection reveals a gap or conflict, do more lookups before answering.

## Final output
When ready, respond in JSON:
{"answer": "<your concise answer>", "cited_nodes": ["node_id1", ...],
 "confidence": "high|medium|low"}
```

D.3. Tool Descriptions Exposed via Function-Calling

The two tools are registered with the answer-stage LLM via the function-calling interface (OpenAI-style `tools` parameter, or the equivalent `FunctionDeclaration` for Gemini). Their `description` strings are part of the prompt the model sees.

`lookup_source`.

```
Retrieve the original text passage that a mind-graph node was derived from.
Returns ~1000 chars of context around the source quote.

Parameters:
  node_id (string, required) -- The id of the node to look up.
```

`subgraph_summary`.

```
Get a structural overview of the mind graph organized into Leiden subgraphs.
Call mode='index' first to see a compact list of
(subgraph_id, title, impact_rating, n_nodes) for every subgraph. Then call
mode='detail' with a specific subgraph_id to get that subgraph's full
executive summary and 5-10 detailed findings. Use when the graph is large
and you need a global map before drilling in with lookup_source.

Parameters:
  mode (string, required, one of {"index", "detail"}) --
    'index' lists all subgraphs at a glance; 'detail' returns the full
    report for one subgraph.
  subgraph_id (integer) --
    Required when mode='detail'. The integer id shown in the index.
```

D.4. LLM-Judge Prompt

Free-text answers that fail the deterministic scoring cascade (exact match, normalised number, containment, first-sentence containment, key-phrase token-F1 ≥ 0.5) fall through to an LLM judge. The judge model is `gpt-5.4-mini` for every method, including MINDGRAPH, so the judging step is held constant across baselines. The prompt is fixed across runs and is shown below.

```
You are a strict but fair grading judge. Given a question, a gold (reference)
answer, and a predicted answer, decide if the predicted answer is essentially
correct -- i.e. it conveys the same core meaning as the gold answer, even if
phrased differently, more verbose, or using synonyms.

Important: For list-type questions (e.g. "what animals...", "what
techniques..."), the predicted answer must include ALL key items from the
gold answer. A partial list that misses important items, or a list with
substantially different items, should be marked incorrect.

Question: {question}
Gold answer: {gold}
Predicted answer: {predicted}
```

```
Reply with ONLY a JSON object:
{"correct": true/false, "reason": "brief explanation"}
```