Information Hunting: Agentic Retrieval as Native Navigation over Coherent Document Units

Anonymous ACL submission

Abstract

Agentic retrieval-augmented generation has grown more active — agents that reflect on retrieved spans, decide whether to retrieve again, and interleave reasoning with retrieval calls — while the substrate they operate on has remained essentially constant: fixed-size text chunks recovered by similarity over an embedding index. We argue this asymmetry is the ceiling of agentic RAG. We assume a document substrate of *Coherent Document Units* (CDUs) — typed, addressable, atomically-bounded objects derived from a document's natural structure — and study how an agent should act over such a substrate. We introduce **Information Hunting**: a goal-directed mode of operation in which the agent discovers, inspects, commits, and composes typed evidence units rather than ranking and reading text spans. A 1,200-line reference implementation exposes nine navigation primitives over an in-memory typed object store, with no vector database in the critical path. On a 78-query probe with a stronger reasoning backbone we reach **90.1%** accuracy (**83.3%** with a smaller backbone); a 200-query probe at 2.6× scale on 179 documents reaches **80.0%** (**+7.0pp** over the strongest chunk baseline). A substrate-isolation control — chunk baseline given a *matched* action surface — still loses by +13 to +25 percentage points on structural query types where typed units most decisively differ from chunks; a 4-primitive CDU ablation already exceeds the strongest chunk baseline. The substrate, not the tool count, is what wins.

1 Introduction

Code agents have files. Document agents have chunks. That is the problem.

Software-engineering agents work because their substrate has structure. A codebase exposes files, paths, functions, classes, and symbols, each with stable addresses and well-defined operations: the agent opens a file, jumps to a definition, reads a function body, edits a line. Every action targets an object the agent can name and the user can verify.

Document-grounded agents are asked to behave similarly, but are given a substrate of opaque text spans. A "chunk" — a 512- or 1024-token slice of the source — is an artifact of vector-database packaging, not a property of the document. It cuts equations from their labels, tables from their headers, and code functions in half. The agent's only tool is `retrieve(query, k)`: throw a string at a similarity index and hope the answer ranks in the top-k window. When it doesn't, the recourse is to retrieve again with a different string. This is the design space of contemporary agentic retrieval-augmented generation (RAG): more search rounds over the same opaque substrate, more reflection on returned snippets, more reranking. The agent gets more *active*, but the documents remain *passive*.

We argue this asymmetry is the ceiling of agentic RAG. The fix is not to give the agent better search; it is to give it something structured to navigate. We assume a document substrate of **Coherent Document Units** (CDUs) — typed, addressable, atomically-bounded objects derived from a document's natural structure (sections, equations, tables, code blocks, figure captions, lists) — and study how an agent should *act* over such a substrate.[1] Once units are typed and addressable, *retrieval* gives way to *navigation*: the agent issues typed queries (`find_units(..., type=equation)`), commits to specific units by stable identifier (`expand_unit(uid)`), and traces structural relations between them. We call this mode **Information Hunting**: a goal-directed process that discovers, inspects, commits, and com-

[1] CDU construction is the subject of a parallel manuscript; a minimal self-contained interface (six fields: `unit_id`, `element_type`, `content`, `heading`, `neighbors`, `source_ref`) is given in Appendix A. Our reference parser is intentionally minimal; replacing it with MinerU (Wang et al., 2024) or Nougat (Blecher et al., 2023) can only improve results.

poses evidence objects rather than ranking and reading text spans. The two paradigms differ in what they *terminate on* — search terminates on retrieval saturation; hunting terminates on evidence completeness — and consequently in their failure modes: search fails by ranking the wrong span (invisible in the index); hunting fails by failing to commit (visible in the trajectory).

The substrate change also dissolves much of the engineering complexity contemporary RAG accretes. Rerankers, query rewriters, hypothetical-document expansions, and fusion strategies are largely compensations for a missing layer — documents were never represented as navigable evidence objects in the first place. Our reference implementation is approximately 1,200 lines of core code, with no vector database in the critical path, and supports nine navigation primitives over a typed in-memory CDU store. On a 78-query probe, the system reaches **90.1%** accuracy with a stronger reasoning backbone (**83.3%** with a smaller one); on a disjoint 200-query probe at 2.6× scale, **80.0%** — **+7.0pp** over the strongest chunk baseline (vs +3.8pp at 78q). A *substrate-isolation control* — chunk baseline with an action surface matched to ours — still loses by +13 to +25pp on the structural query types where typed units decisively differ from chunks; a 4-primitive CDU ablation already exceeds the strongest chunk baseline. The substrate, not the tool count, is what wins.

Contributions. **(i)** *Information Hunting* — an agentic retrieval architecture over typed, addressable document units, characterised by typed primitives, addressable commitment of evidence units, and structural attribution, with embedding similarity demoted to one optional lookup tool. **(ii)** *A nine-primitive navigation action space* with a ∼1,200-line reference controller and no vector database in the critical path; a 4-primitive ablation already exceeds the strongest chunk baseline. **(iii)** *Empirical evidence across two probe sets and two model tiers*, including a substrate-isolation control with a matched action surface that still loses by 20.5pp overall and 13–25pp on structural query types.

2 Background: From Passive Retrieval to Agentic Search

Modern retrieval-augmented generation has evolved along a single axis: making the retrieval *process* increasingly active. The substrate it operates on — text chunks recovered by similarity over an embedding index — has remained essentially constant. We trace four lines of work that occupy this design space and identify the layer none of them touch.

Passive RAG. Dense retrieval (Karpukhin et al., 2020; Guu et al., 2020; Izacard et al., 2023) and hybrid variants (Khattab and Zaharia, 2020; Robertson and Zaragoza, 2009) established the chunk-rank-stuff-generate paradigm. A document is split into fixed-size token windows, each window is embedded, queries are matched against the index, and the top-k windows are concatenated into the generator's context. The chunk is treated as the natural retrieval unit despite being a token-budget artifact: it is the unit at which embeddings are computed, not a unit the document itself recognises. Errors at this layer — a table cleaved across two chunks, an equation orphaned from its label, a code function spliced mid-body — are absorbed silently.

Agentic RAG. A second wave (Asai et al., 2024; Yan et al., 2024; Trivedi et al., 2023; Jiang et al., 2023; Yao et al., 2023; Khattab et al., 2024) added agency on top of the chunk substrate: the agent reflects on retrieved spans and decides whether to retrieve again, with what query, and how to interleave reasoning with retrieval. These systems improve multi-hop and decomposable queries by replacing a single retrieve-then-generate decision with a loop, but leave the retrieval *object* unchanged — the agent still issues `retrieve(query, k)`, still receives ranked text spans, and still has no addressable handle to commit to. Their algorithmic contributions (self-reflection, interleaved reasoning, forward-looking generation, declarative pipelines) are composable with ours: a future system that runs, e.g., Self-RAG-style reflection over CDU-typed primitives is a natural follow-up. We do not report head-to-head numbers because a naïve port would conflate substrate change with algorithm change.

Structure-aware, hierarchical, and generative retrieval. RAPTOR (Sarthi et al., 2024) summarises sections at multiple levels of abstraction to improve ranking; PageIndex (VectifyAI, 2025) builds a hierarchical tree and lets the LM reason over the tree to locate relevant pages; DSI (Tay et al., 2022) encodes document identifiers as decoder targets, treating the retrieval target as a token sequence rather than a span. This is the closest body of work to ours and shares the diagnosis that flat chunking discards information. The differen-

tiation is what structure is *for*: prior work uses structure to improve ranking or LM-side reasoning over a textual hierarchy; we use it to define a *typed-object action space* on which an agent navigates. Their unit is still a span, a textual node with structural metadata, or a document-level identifier; ours is a typed object with operations (find / peek / expand / compare / neighbours / trace) defined on it.

Differentiator. Long-context LMs sidestep retrieval for single-document tasks without changing the substrate question; we treat them as orthogonal. Across all four lines, existing systems agentise the retrieval *process*; we change the retrieval *object*. The intuition is borrowed not from RAG but from software-engineering agents: code agents work because their substrate exposes typed objects (files, functions, symbols) at addressable locations with stable identifiers. When the substrate exposes structure, the agent needs the right primitives, not more capacity. Document agents have not had such a substrate. We propose one. Prior agentic-RAG work asks how the agent should search; we ask what the document must become for the agent to navigate it.

3 Information Hunting: Agentic Retrieval as Navigation

The reframing: hunt vs search. A search-based retrieval agent operates on a model of the world in which evidence is *somewhere in the index* and the question is which spans most resemble the query. Each retrieval call returns a ranked list; the agent reads, decides whether to issue another query, and eventually commits to an answer based on the spans it has read. Termination is governed by *retrieval saturation* and citation, when offered, points to spans of approximate relevance.

A hunting agent operates on a different model. Evidence is *located in specific document objects*, and the question is which objects to commit to. Each navigation call yields candidate CDUs by stable identifier; the agent inspects, commits to specific units, and composes evidence across committed units. Termination is governed by *evidence completeness* — every subgoal of the question grounded in at least one committed CDU — and citation is structural: the answer references the `unit_ids` of the units that ground it.

> *Search finds candidates; hunting commits evidence.*

Group	Primitive	Purpose
Orient	`open_document(doc)`	structural tree
Discover	`find_units(q, type?)`	typed semantic finder
	`grep_units(kw)`	exact-match locator
Inspect	`peek_unit(uid)`	cheap glance
	`read_unit(uid, o, l)`	page-read of long unit
Commit	`expand_unit(uid)`	full content (committed)
	`compare_units(uids)`	multiple committed units
Navigate	`get_neighbors(uid)`	sibling/order-adjacent
	`trace_source(uid)`	resolve refs

Table 1: Nine navigation primitives. Full I/O contracts in Appendix B.

The two paradigms differ in failure mode. Search fails by ranking the wrong span — a similar-looking chunk recovered from a wrong context. Hunting fails by failing to commit — the agent navigates poorly and terminates without grounded evidence. The latter is a navigation-policy error visible in the trajectory; the former is a similarity error invisible in the index. The shift is itself a contribution: navigation-policy errors are debuggable and amenable to controller-level intervention in a way similarity errors are not.

The primitive set. We expose nine navigation primitives, grouped by hunting affordance (Table 1). Compare to chunk-RAG's single `retrieve(query, k)`: less expressive, less faithful to document reality, with no addressable target to commit to. Each primitive realises a specific navigation move possible *only because* CDUs are typed and addressable; the kit is the action space the substrate makes available, not a layer added on top.

The agent loop. Two properties matter. First, *evidence is committed CDU identifiers, not retrieved snippets*: the agent's working memory is a set of `unit_ids` with their content, not a buffer of accumulated retrieval text. Second, *termination is bounded by evidence completeness, not by top-k or step budget*: the loop ends when every subgoal of the query is grounded in at least one committed CDU. The step budget exists only to bound runaway behaviour. The reference loop is in Appendix C; specific controller details are not part of the paper's contribution.

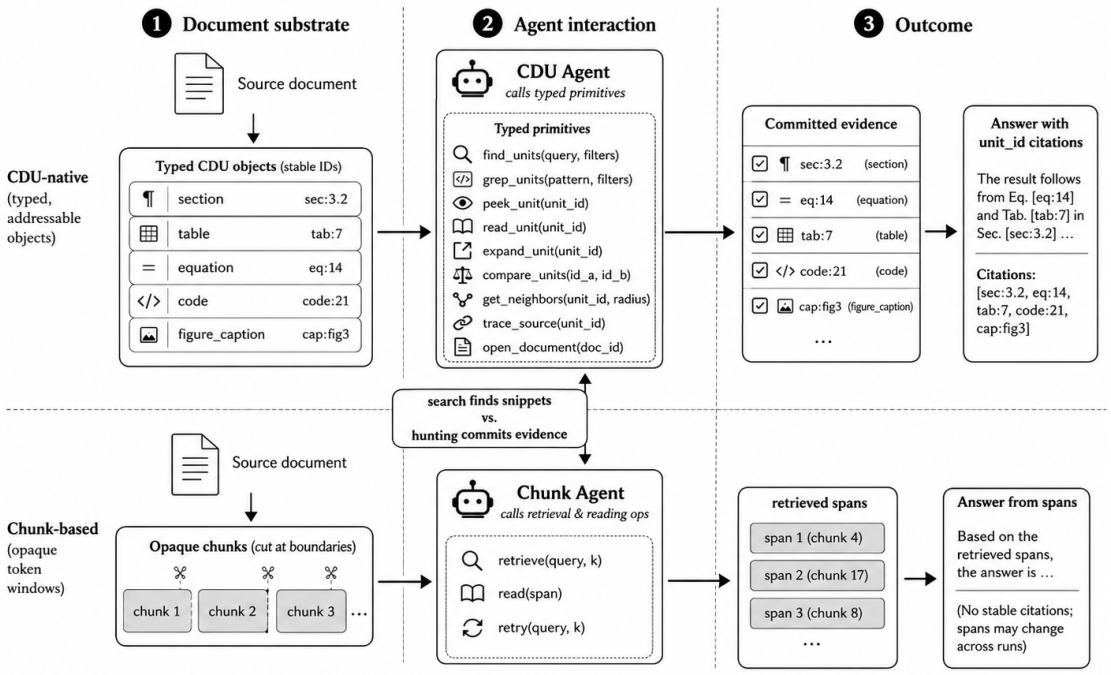

Figure 1: CDU agent with primitives over structured document units (top), contrasted with a chunk-based baseline that reads unstructured text spans (bottom). The CDU agent navigates a typed CDU store using explicit primitives, commits stable unit-level evidence, and composes grounded answers with structural citations.

4 Reference Implementation

The reference implementation is approximately 1,200 lines of core code: typed object store, nine primitives, agent controller, system prompt. There is no vector database in the critical path; the in-memory typed object store is sufficient for the corpus scales evaluated here. We deliberately resisted adding orchestration machinery that other RAG systems accumulate — no reranker, no query rewriter, no hypothetical-document generator, no retrieval-fusion stage, no separate planning module — because the hypothesis being tested is that capability emerges from the substrate and its action space, not from orchestration complexity. We do not claim that this particular controller, prompt, or parser is optimal; each is one instantiation, deliberately minimal, included as supplementary material so the substrate change can be inspected directly and the claims reproduced.

5 Experimental Setup

Research questions. (RQ1) Can a minimal CDU-native agent perform grounded document navigation across a diverse query set? (RQ2) Does CDU-native navigation outperform chunk-substrate retrieval on agentic tasks? (RQ3) Where does the substrate change matter most: structural lookup, cross-element synthesis, or prose QA? (RQ4) What are the cost (token, step) and attribution trade-offs?

Probe sets. We use two stratified probes drawn from a multi-domain corpus (scientific papers, financial filings, medical literature, legal contracts, developer documentation). The 78-query probe is a high-resolution set on 6 documents; the 200-query probe is a 2.6× scale set on 179 documents disjoint from 78q. Per-type counts (section/table/equation/code/cross_element): 78q = 32/23/8/5/10; 200q = 40/45/40/35/40. Documents are sourced from established public benchmarks; we contribute unit-level gold annotations on top of an unchanged document distribution. Large-scale evaluation against (Yang et al., 2018; Tang and Yang, 2024; Ma et al., 2024; Islam et al., 2023) is deferred to follow-up work, which requires unit-level annotation on benchmarks not designed under the substrate-aware paradigm. Construction details: Appendix D.

Systems compared. Three chunk baselines (varying in agency and action surface) plus CDU+hunt and a CDU ablation. `sp_chunk` is non-agentic single-pass retrieve-then-generate (top-k=5). `chunk_agent` is the agentic baseline with one primitive (`search(query, k)` returning ranked full-text chunks). `chunk_rich`

is the *substrate-isolation control*: an action surface matched to ours (lookup + peek + paged read; `search_chunks` returns a preview only, `peek_chunk` returns the first 300 chars, `read_chunk` returns the bounded chunk content), so the only remaining variable between it and CDU+hunt is the *unit* the primitives operate on. `CDU 4-tool` is the converse ablation: `open_document`, `find_units`, `expand_unit`, `read_unit` over CDUs, testing how much headroom comes from the additional five primitives versus from the substrate itself. **CDU+hunt** is the full nine-primitive system.

All systems use the same agent backbone and the same gpt-5.4 judge with a two-level rubric (correct/wrong with rationale). Agent backbones evaluated: gpt-4o-mini (matched-backbone comparison) and gpt-5.4-mini (stronger-backbone for CDU+hunt on 78q to test robustness). max_steps = 12; temperature 0; chunk_size = 1000; chunk_overlap = 200; BGE-small-en-v1.5 embeddings.

Metrics. Outcome: `answer_correctness` from the LLM judge. CDU-native diagnostics (computed only on queries with gold unit-id annotations): `evidence_recall`, `attribution_hit`, `trajectory_valid`. Efficiency: `mean_steps`, `max_steps`, `mean_tokens`, `max_tokens`, `wandering_rate` (fraction of queries with $\geq$8 steps).

6 Results

We report results under a **two-tier model + two-probe** evaluation. The 78q probe is run with two agent backbones (gpt-4o-mini, gpt-5.4-mini) to test that the pattern is not specific to a single instruction-following capacity; the 200q probe is run with gpt-4o-mini to verify the pattern at 2.6$\times$ scale.

6.1 Overall 78q accuracy, two model tiers, three chunk baselines

Table 2 reports headline accuracy and efficiency on 78q. Four observations frame the rest of the section.

(i) CDU+hunt is the best system at both model tiers: +3.8pp over the strongest chunk baseline at gpt-4o-mini and reaching 90.1% at gpt-5.4-mini. The gap widens at the stronger tier — CDU+hunt scales with backbone capacity in a way chunk baselines do not.

(ii) The substrate, not the tool count, is what wins: the CDU 4-tool ablation drops only -1.2pp below CDU+hunt full while still beating the best chunk baseline by +2.6pp and the substrate-matched chunk_rich control by +19.3pp. The substrate's value survives a 5-of-9 primitive removal; the chunk substrate fails even with a 3-of-3 matched action surface.

(iii) Giving the chunk substrate a richer action surface does not help: chunk_rich (3 matched primitives) under-performs chunk_agent (1 primitive) by -16.7pp, because chunks lack the structural anchors that would let the agent productively use peek/read. When there is no addressable typed unit to commit to, additional primitives become coordination overhead rather than navigation tools.

(iv) The headline gap concentrates on structural query types (table, equation), where the answer *is* a typed unit; on prose-heavy queries the chunk substrate remains competitive (§6.3). Attribution is structural: every CDU answer cites `unit_ids`, a property no chunk-substrate baseline can offer.

6.2 Where object identity matters

Gains concentrate where the answer *is* an object (Table 3). The CDU advantage is **structurally-attributable**: it shows up where the gold-truth answer *is* a typed unit (table +13.0pp, equation +25.0pp), and disappears or inverts on prose-heavy types (section_qa, code_qa) where the answer is a short factual statement embedded in continuous text and the chunk substrate's locality penalty is small. Cross-element queries are a tie at the smaller backbone but show a +28.9pp gap (88.9% vs 60.0%) at the stronger backbone — the substrate's value compounds with the controller's ability to chain typed primitives.

> *When the answer is an object, retrieving text is an approximation. Navigating to the object is the direct solution.*

6.3 Prose QA: chunk baselines compete and sometimes win

Section_qa is the query type where the chunk substrate has its strongest claim — the answer often lives in a span of continuous prose. On 78q, sp_chunk reaches 90.6% on section_qa and chunk_agent reaches 87.5%, both above CDU+hunt at gpt-4o-mini (81.2%). With the stronger gpt-5.4-mini backbone CDU+hunt rises to

System	Tools	Acc (4o-mini)	Acc (5.4-mini)	Mean steps	Mean tokens	Attribution
sp_chunk (top-k=5)	1 passive	76.9%	–	n/a	1.6K	text span only
chunk_agent	1 active	79.5%	–	1.5	4.3K	text span only
chunk_rich	3 matched	62.8%	–	5.1	8.4K	text span only
CDU 4-tool (ablation)	4 minimal	82.1%	–	4.8	17.7K	`unit_id`
CDU+hunt (ours)	9 native	**83.3%**	**90.1%**	6.5 / 4.4	30.7K / **18.1K**	`unit_id`

Table 2: 78q main results. All systems judged by gpt-5.4 with a two-level rubric. CDU+hunt + 5.4-mini is on 71 rows after dropping 7 queries flagged for parser-side gold issues.

Type	n	CDU+hunt	best chunk	Δ
section_qa	32	81.2%	90.6%	−9.4
table_qa	23	**91.3%**	78.3%	**+13.0**
equation_qa	8	**100%**	75.0%	**+25.0**
code_qa	5	80.0%	100%	−20.0
cross_element_qa	10	60.0%	60.0%	0.0

Table 3: 78q per-type accuracy at gpt-4o-mini, gpt-5.4 judge. "best chunk" = max over sp_chunk / chunk_agent / chunk_rich.

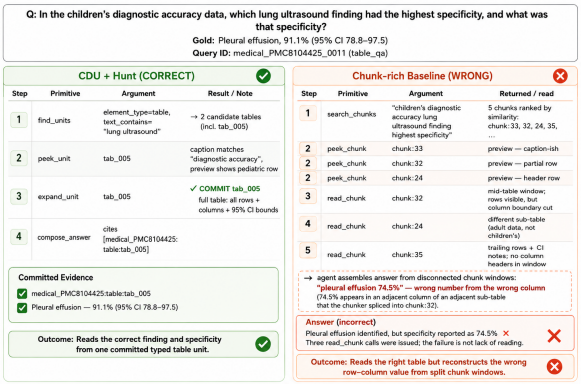

Figure 2: Paired trajectory on a table query where the source table is split across three chunks. chunk_rich reads all three chunks but assembles "74.5%" from the wrong column. CDU+hunt commits the whole typed table and reads "91.1%" atomically.

88.9%, closing the gap. We do not claim CDU navigation strictly dominates chunk retrieval on every query type. Section-level prose QA with a short factual answer is exactly where the chunk substrate's flat similarity index is a strong approximation. The framework is intended to support high-fidelity navigation where structure matters — where the typed unit ontology is doing real work — not to replace cheap shallow lookup where it does not.

6.4 Cross-element queries: navigation pays the most at the stronger tier

Cross-element queries require composing evidence from two or more units. With the smaller backbone the result is a flat tie (CDU+hunt at gpt-4o-mini and all three chunk baselines score 60.0% on the cross-element subset). With the stronger backbone the picture changes: CDU+hunt at gpt-5.4-mini reaches 88.9% on cross-element — a +28.9pp gap over the best chunk baseline. The substrate's value compounds with the controller's ability to chain typed primitives across sub-targets. Figure 2 illustrates the mechanism with a paired trajectory: chunk_rich, on a table query whose source table is split across three chunks, reads all three chunks but assembles a numerically-wrong answer from disconnected windows; CDU+hunt commits the whole typed table via `expand_unit` and reads the correct value atomically.

6.5 Attribution: a structural property

Every CDU+hunt answer is accompanied by the `unit_ids` it commits to as evidence. Chunk baselines, at best, return text spans of retrieved chunks; they cannot return addressable identifiers because their substrate has no addressable units. The asymmetry is structural, not engineering: a richer chunk-RAG cannot patch this because the object it would need (a stable unit identifier) does not exist in the chunk substrate.

> *A chunk can support an answer; a CDU can be cited as evidence.*

6.6 Efficiency — a substrate-honest accounting

CDU+hunt is not the cheap option in absolute tokens. With gpt-4o-mini it uses 30.7K mean tokens per query versus 4.3K for chunk_agent and 1.6K for sp_chunk; chunk_rich sits between at 8.4K. The premium reflects what `expand_unit` does: it returns a full structural unit (a complete table including header and totals, an entire labelled equation derivation, a whole code function) rather than a bounded 1000-char window. The premium buys

the accuracy at the places that matter (+13pp tables, +25pp equations, +28.9pp cross-element at the stronger backbone). Across the two CDU+hunt backbones the picture sharpens: under gpt-5.4-mini, mean tokens drop to 18.1K and mean steps to 4.4 while accuracy rises to 90.1%. The CDU substrate's value is amplified by a stronger controller. Token distribution per system: Figure 3.

6.7 200-query scale verification

Table 4 reports the 200q probe results with gpt-4o-mini. CDU+hunt leads by **+7.0pp** overall — a larger margin than the +3.8pp gap on 78q, indicating that the CDU advantage holds and slightly amplifies on the wider, more diverse 200q distribution. The 200q per-type breakdown (Appendix E) verifies the structural-query advantage observed on 78q (equation +32.5pp, table +4.5pp, section +10.0pp); code_qa and cross_element_qa flip on the smaller backbone — most demanding regimes in the entire evaluation, with cross-element expected to invert again with a stronger backbone (78q's gpt-5.4-mini result took cross-element from a flat 60% tie to 88.9%). The headline finding is robustness: the CDU+hunt advantage is not a 78q artifact.

7 Analysis: Why CDUs Change the Failure Mode

Representation debt. Many of the components that contemporary RAG pipelines accrete — rerankers, query rewriters, hypothetical-document expansions, retrieval-fusion strategies, multi-vector encoders — are compensations for a single missing layer. Documents were never represented as navigable evidence objects in the first place. Each compensating component patches a particular symptom of that absence: rerankers correct for noisy first-stage ranking, query rewriters work around literal-embedding brittleness, hypothetical-document expansion synthesises text the index might match. None of them address the underlying issue: the retrieval object is a token-budget artifact rather than a document object. Once the substrate provides typed addressable units, a substantial fraction of the compensating machinery becomes unnecessary.

The failure-mode shift. A failure in chunk-substrate retrieval lives inside the embedding manifold and is not directly inspectable: cosine similarity does not tell you why the wrong chunk ranked high, only that it did. A failure in CDU-substrate navigation lives in the trajectory and is fully visible: every primitive call, its arguments, and its result are recorded, and the wrong decision can be located precisely. Failures become locally identifiable units of work that controller-level intervention can address.

Boundary of navigation. Trajectory inspection reveals a category of failures with a consistent signature: the agent navigates correctly, commits to the right CDU, and *still* produces a wrong answer. The clearest examples come from finance cross-element queries — the agent locates and commits the table containing category sales, but the question requires multiplying net sales by category mix percentages to derive category-specific revenue, and arithmetic on the committed table fails. No additional searching, no tighter `find_units`, no smarter `compare_units` would help — the right CDU is already committed. What is missing is the ability to perform structured operations *on* a committed CDU: typed, semantic, reproducible operations beyond reading text. We flag this as a complementary research direction outside the navigation framework, naturally targeted at the CDU substrate (typed addressable units are a natural target for typed operators).

8 Conclusion

Retrieval should be navigation, not ranking, when documents have structure to navigate. Chunks are an artifact of vector-database packaging rather than a property of documents themselves; agentic systems built on the chunk substrate inherit that artifact and spend much of their machinery patching its consequences. We propose **Information Hunting** as an agentic retrieval architecture that operates over typed, addressable document units — discovers, inspects, commits, composes, with structural attribution as a first-class property. A 1,200-line reference implementation with no vector database in the critical path reaches 90.1% accuracy on a 78q probe with a mid-tier reasoning model, 83.3% with a smaller backbone, and 80.0% (vs 73.0% for the strongest chunk baseline; +7.0pp) on a disjoint 200q probe at 2.6× scale. A substrate-isolation control gives the chunk baseline a matched action surface and shows the substrate is the locus of the gain. A 4-primitive ablation of our own system still exceeds the strongest chunk baseline — the substrate, not the tool count, is what wins. The future of agentic retrieval-augmented generation is not more search over chunks; it is native navigation

System	n	Acc	Mean steps	Mean tokens	Max tokens	Wander (≥8 steps)
chunk_agent	200	73.0%	1.47	4.2K	113.2K	0.5%
CDU+hunt (ours)	200	**80.0%**	5.50	21.1K	70.6K	19.0%

Table 4: 200q main results, gpt-4o-mini agent, gpt-5.4 judge.

over document objects.

Limitations

Scale and statistical power. The 78q and 200q probes do not rise to the scale of public benchmarks like HotpotQA (Yang et al., 2018), MultiHop-RAG (Tang and Yang, 2024), MMLongBench-Doc (Ma et al., 2024), or FinanceBench (Islam et al., 2023); per-cell n on 78q is too small for confidence intervals, so we report magnitudes rather than intervals. Adapting these benchmarks requires unit-level gold annotations, a community gap we plan to address.

Parser-dependence and CDU-in-CDU. CDU quality is upstream of every result. Our reference parser does not recursively segment a section that *contains* a sub-table or sub-code-block, visible in the 200q table_qa drop (77.8% vs 91.3% on 78q): on documents whose tables are syntactically embedded inside a section, the parser emits the entire section as one CDU. A recursive parser emitting *CDU-in-CDU* (the substrate already supports the addressing scheme) or replacement with MinerU (Wang et al., 2024) or Nougat (Blecher et al., 2023) can only improve reported numbers.

Scope and comparators. Each query is single-document scoped; cross-document hunting (where the identifier scheme extends with corpus-level addressing and `find_units` scopes across stores) is a natural extension we defer. Head-to-head comparisons against Self-RAG (Asai et al., 2024), IRCoT (Trivedi et al., 2023), RAPTOR (Sarthi et al., 2024), or PageIndex (VectifyAI, 2025) on identical documents are not reported: a naïve port to CDUs conflates substrate change with algorithm change. The principled comparison — running each algorithm over the CDU substrate — is a natural follow-up.

Operations over committed CDUs. §8 identified a class of failures the navigation primitives cannot solve: the right CDU is committed, but the answer requires a structured operation on it (arithmetic across a table, joining a chart caption with a downstream paragraph). Extending the action space from navigation primitives to typed operations on committed units is a complementary direction; typed addressable units are a natural target for typed operators.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of ICLR 2024*.

Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. 2023. Nougat: Neural optical understanding for academic documents. *arXiv preprint arXiv:2308.13418*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: Retrieval-augmented language model pre-training. In *Proceedings of ICML 2020*.

Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. 2023. FinanceBench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of EMNLP 2023*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of EMNLP 2020*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *Proceedings of ICLR 2024*.

Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of SIGIR 2020*.

Yubo Ma, Yuhang Zang, Liangyu Chen, Meiqi Chen, Yizhu Jiao, Xinze Li, Xinyuan Lu, Ziyu Liu, Yan Ma, Xiaoyi Dong, Pan Zhang, Liangming Pan, Yu-Gang Jiang, Jiaqi Wang, Yixin Cao, and Aixin Sun. 2024. MMLongBench-Doc: Benchmarking long-context document understanding with visualizations. In *Proceedings of NeurIPS 2024 Datasets and Benchmarks Track*.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *Proceedings of ICLR 2024*.

Yixuan Tang and Yi Yang. 2024. MultiHop-RAG: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391*.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer memory as a differentiable search index. In *Proceedings of NeurIPS 2022*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of ACL 2023*.

VectifyAI. 2025. PageIndex: Vectorless, reasoning-based document indexing for RAG. https://github.com/VectifyAI/PageIndex and https://pageindex.ai/blog/pageindex-intro. Open-sourced 2025-04-01.

Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, and 1 others. 2024. MinerU: An open-source solution for precise document content extraction. *arXiv preprint arXiv:2409.18839*. Author list abbreviated.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. Corrective retrieval augmented generation. In *Proceedings of ICLR 2024*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of EMNLP 2018*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of ICLR 2023*.

A CDU schema

The minimal CDU interface assumed throughout the paper. CDU construction is the subject of a parallel manuscript; we reproduce the interface here for self-containment.

Interface (Table 5). A CDU is never split mid-element: the unit of indexing is the unit of evidence. Three properties of the interface are load-bearing for the method: (i) *type-filterability* (`find_units(query, element_type=table)` requires `element_type`); (ii) *addressable commitment* (the agent commits to evidence by `unit_id`; without stable identifiers, neither `expand_unit` nor `compare_units` exists, and final answers cannot cite anything an external verifier can re-resolve); (iii) *atomic content* (`content` is the whole element, so a single committed unit suffices for a structural answer).

Field	Role
`unit_id`	stable address `doc_id:type:elem_id`
`element_type`	one of `{section, paragraph, equation, table, code, list, figure_caption}`
`content`	natural body of the element — never split
`heading`	element's title or label, where applicable
`neighbors`	parent, sibling document-order, child links
`source_ref`	pointer back to source span for verification

Table 5: The minimal CDU interface.

Reference implementation. The full schema (used by our reference parser and store) fits in approximately ten lines of typed Python:

```
@dataclass
class CDU:
    unit_id: str   # "doc:type:elem"
    doc_id: str
    element_type: str
    heading: str | None
    content: str   # never split
    section_path: list[str]
    document_order: int
    neighbors: dict     # prev/next/
                        # parent/children
    source_ref: dict    # page, span,...
```

Embedding's role. `find_units` may use a dense embedding index as one mechanism for semantic lookup, but embeddings are *demoted from*

substrate to tool: the primary index is the typed object store keyed by `unit_id`; an embedding index is one of several lookup mechanisms (alongside exact-match grep, type-filtered enumeration, structural traversal). The object the agent receives is always a typed CDU with a stable identifier, not a span of text.

B Hunting primitives — I/O contracts

The nine primitives, with input and output shapes:

- `open_document(doc_id)` → {`counts_by_type`, `section_tree`, `element_captions`} — full TOC.
- `find_units(query, element_type?, heading_contains?, section_path?)` → ranked list of {`unit_id`, `score`, `type`, `heading`}.
- `grep_units(pattern, element_type?, section_scope?)` → list of {`unit_id`, `match_offset`, `snippet`}.
- `peek_unit(unit_id)` → element-type-aware preview (eq: full LaTeX; table: caption + 1st row; section: heading + first 500 chars).
- `read_unit(unit_id, offset, limit)` → paged content of one CDU.
- `expand_unit(unit_id)` → full `content` — marks as committed evidence.
- `compare_units(unit_ids)` → side-by-side `content` of multiple CDUs in one call — marks all as committed.
- `get_neighbors(unit_id, scope)` → list of structurally adjacent CDUs.
- `trace_source(unit_id)` → resolved in-text references (e.g. "see Eq. 3" → `doc:equation:eq_003`).

Full JSON schemas (as passed to the OpenAI Chat Completions API in the `tools=` parameter) are reproduced in the supplementary code.

C Reference controller

```
def run(query, store, model,
        max_steps=12):
    history = [system_prompt(),
               user_query(query)]
    committed = []
    for step in range(max_steps):
        msg = llm(model, history,
                  tools=SCHEMAS)
        if not msg.tool_calls:
            return compose(
              msg.content, committed)
        for tc in msg.tool_calls:
            obs = execute(tc, store)
            if tc.name in {
              "expand_unit",
              "compare_units"}:
                committed.extend(
                  extract_uids(tc.args))
            history.append((tc, obs))
    return compose(
      history[-1].content, committed)
```

System prompt and prompt evolution are documented in supplementary code. Specific prompt-tuning methodology is outside this paper's scope.

D Evaluation set construction

78q held-out set. Six documents across four domain families (scientific = 2 NLP papers; medical = 1 PMC article; financial = 1 earnings press release; legal = 1 contract excerpt; developer = 1 Python docs page). Query construction by type: section_qa has gold-truth answer in one element_type=section CDU; table_qa in one element_type=table CDU; equation_qa is the LaTeX or symbolic content of one element_type=equation CDU; code_qa is a value or behaviour of one element_type=code CDU; cross_element_qa requires composition across $\geq$2 CDUs of any types.

200q probe set. Stratified sample (40/45/40/35/40 for section/table/equation/code/cross_element) drawn from `llm_debiased` and `llm_native` query files (gpt-5.4-generated queries with post-hoc adversarial debiasing). 179 distinct documents across four domain families, disjoint from 78q. Sampling code: `sample_200q.py` (deterministic with `seed=20260526`).

Judge rubric. Two-level: *correct* (essential information present and accurate, formatting may differ) / *wrong* (essential information absent, incorrect, or contradictory). Each judgment carries a rationale. Judge model: gpt-5.4. We do not report inter-annotator agreement with a human judge.

Type	n	CDU+hunt	chunk_agent	Δ
section_qa	40	**95.0%**	85.0%	**+10.0**
table_qa	45	77.8%	73.3%	+4.5
equation_qa	40	**92.5%**	60.0%	**+32.5**
code_qa	35	91.4%	**97.1%**	−5.7
cross_element_qa	40	45.0%	**52.5%**	−7.5

Table 6: 200q per-type accuracy, gpt-4o-mini agent, gpt-5.4 judge. The structural-query advantage observed on 78q amplifies (equation +32.5pp, section +10.0pp). Code and cross-element flip on the smaller backbone; we expect inversion with a stronger backbone, as 78q's gpt-5.4-mini result took cross-element from a flat 60% tie to 88.9%.

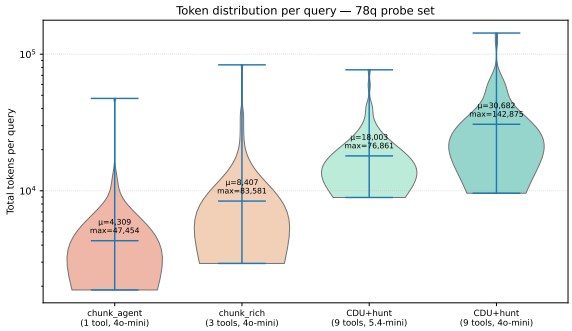

Figure 3: Token consumption per query, 78q probe, log scale. chunk_agent is cheapest at 4.3K mean; CDU+hunt with gpt-4o-mini is the most expensive at 30.7K mean; CDU+hunt with gpt-5.4-mini drops to 18.1K mean while raising accuracy to 90.1%.

E 200q per-type results

F Token distribution: F5

G Parser pipeline

The reference parser is a hand-written rule-based pipeline that (i) loads PDF via `pdfplumber`, HTML via `beautifulsoup`, Markdown natively; (ii) walks the document tree and emits typed units for section / paragraph / equation / table / code / list / figure_caption elements; (iii) populates the CDU schema fields; (iv) computes `neighbors` by document-order traversal and explicit parent–child links.

The pipeline is intentionally minimal and does *not*: recursively segment a section containing a sub-table or sub-code-block (the "CDU-in-CDU" limitation discussed in §Limitations); use a learned layout-detection model (MinerU (Wang et al., 2024), Nougat (Blecher et al., 2023), LayoutLMv3, etc.); validate cross-references at parse time (resolved at query time by `trace_source`).

Replacing this parser with any of the above can only improve reported numbers, so all results in this paper are *lower bounds* with respect to parser sophistication.