Corpus2Skill: Distilling Enterprise Knowledge into Navigable Agent Skills for QA and RAG

Yiqun Sun **Pengfei Wei*** **Lawrence B. Hsieh**
Magellan Technology Research Institute (MTRI)
{duke.sun, pengfei.wei, lawrence.hsieh}@mtri.co.jp

Abstract

Retrieval-Augmented Generation (RAG) grounds LLM responses in external evidence but treats the model as a passive consumer of search results, with no view of how the corpus is organized or what it has not yet seen. We present CORPUS2SKILL, a system-level retrieval architecture for bounded, structurally coherent corpora such as enterprise knowledge bases: an offline compiler distills the corpus into a hierarchical skill directory, and at serve time an LLM agent navigates it, drilling from a bird's-eye view through progressively finer summaries down to documents and backtracking when a branch is unproductive. On an enterprise customer-support benchmark, CORPUS2SKILL improves both answer quality and grounding over single-shot dense, hybrid, hierarchical-retrieval, and agentic RAG baselines at a moderate cost tradeoff, and the lead persists under encoder-matched controls and paired significance tests. An eleven-dataset study shows that corpus navigation is *not* a universal replacement for retrieval: it significantly wins on five datasets, ties on three, and loses on three. It helps on single-domain corpora with a recoverable topical taxonomy, but flat retrieval remains preferable on open-domain factoid pools or homogeneous-tabular corpora that defeat top-level clustering. We characterize this scope distinction as a design guideline for knowledge-grounded systems. Code is available at https://github.com/dukesun99/Corpus2Skill.

1 Introduction

Enterprise knowledge bases contain thousands of heterogeneous support articles, product guides, and policy documents (Gao et al., 2023). Retrieval-Augmented Generation (RAG; Lewis et al., 2020) grounds LLM responses in such corpora by embedding the query, fetching similar passages from a vector index, and conditioning the answer on the retrieved context. This is now the dominant paradigm for enterprise QA, customer support, and compliance workflows (Gao et al., 2023; Singh et al., 2025). Yet for queries that touch several topics in the knowledge base, **the LLM never sees the forest for the trees**: it receives the top-k passages but has no view of what else is in the corpus, no way to tell whether the best evidence was even in the retrieval window, and no map to plan a more thorough search (Sun et al., 2026b; Tang et al., 2025).

Agentic RAG (Yao et al., 2023; Schick et al., 2023) partially closes this gap by letting the agent issue multiple search queries, but each query is still a shot in the dark because the agent must guess productive search terms with no view of the corpus structure. Hierarchical approaches such as RAPTOR (Sarthi et al., 2024), GraphRAG (Edge et al., 2024), StructRAG (Li et al., 2025), and BookRAG (Wang et al., 2026b) organize documents into trees

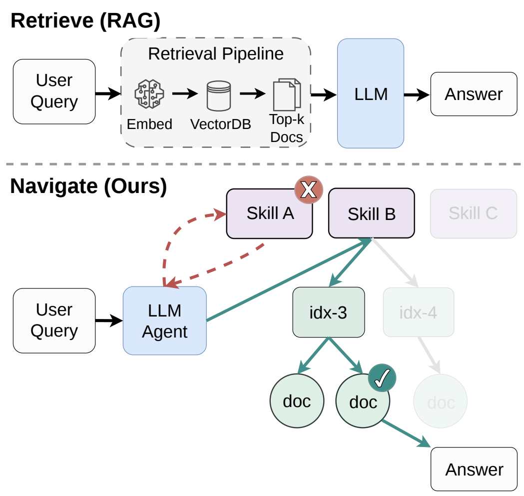

Figure 1: **Retrieve vs. Navigate.** Traditional RAG passively feeds fixed retrieved passages to the LLM. The *navigation* paradigm instead exposes the corpus as a structured hierarchy that the agent actively explores, backtracks through, and drills into to locate evidence.

* Corresponding author.

or graphs with LLM-generated summaries, but the hierarchy is still consumed indirectly through embedding-based search at query time, and the model receives results without seeing the organizational structure that produced them.

Meanwhile, a parallel line of work on LLM agent skills (Wang et al., 2024; Jiang et al., 2026) has shown that agents perform complex tasks more effectively when given structured, reusable knowledge modules that they can selectively load and follow. Agent platforms now support filesystem-based skill packages[1] that use *progressive disclosure* (Section 2) to keep context costs low: the agent sees only lightweight metadata at startup and loads full content on demand, so the hierarchy can grow with the corpus without hitting context-window limits. Because skills are plain directories of files, they require no special infrastructure, which makes them a natural substrate for corpus navigation: rather than searching for passages, one can *distill* the corpus into a navigable skill hierarchy and let the agent explore it.

Figure 1 contrasts these two paradigms. We propose exactly this with CORPUS2SKILL: a one-time offline compilation transforms the corpus into a navigable skill tree, and at serve time the LLM agent browses the tree to locate evidence. The hierarchy itself, not an embedding index, becomes the primary interface between the model and the corpus, giving the agent a bird's-eye view it can reason over before committing to any document. CORPUS2SKILL is a *system-level* retrieval architecture: the compiled hierarchy, its enrichments, and the navigating agent operate as one pipeline, so we both evaluate the complete system against strong retrieval baselines and decompose which components carry the gains. Our contributions are:

- We identify corpus *navigability* as an underexplored axis in RAG and recast retrieval as agentic file browsing over a compiled skill tree.
- We present a compile-then-navigate framework that turns a raw corpus into a hierarchical skill directory requiring no embedding index or vector DB at serve time.
- We provide a quality- and grounding-aligned evaluation on WixQA (Cohen et al., 2025) against five baselines (BM25, Dense, Hybrid, RAPTOR, Agentic), with ablations plus encoder-matched, component, compiler, and 50k-document scale controls that trace the gains to the navigable structure.
- We characterize *when* corpus navigation helps through a 10-subset generalization sweep on RAGBench (Friel et al., 2024), which with WixQA gives 11 datasets: under per-dataset paired significance tests CORPUS2SKILL wins on 5, ties on 3, and loses on 3. The split tracks corpus structure, and flat retrieval remains preferable on open-domain or homogeneous-tabular corpora (HAGRID, TatQA, CUAD).

2 Background: Skills in LLM Agents

From tools to skills. Modern LLM agents act on external systems through *tools*: atomic, stateless functions such as web search or database queries (Schick et al., 2023; Yao et al., 2023). *Agent skills* (Jiang et al., 2026) fill the gap between acting and knowing *how*: reusable modules that package procedural knowledge or multi-step workflows into a form the agent can load and follow. Jiang et al. (2026) formalize a skill as a tuple $S = (C, \pi, T, R)$ of an applicability condition, an execution policy, a termination condition, and a reusable interface, distinguishing it from one-off plans, episodic memories, and atomic tools.

Filesystem-based skills. The current standard skill representation, adopted by Anthropic's Skills API[2] and recent open agent platforms, is a *filesystem-based package*: a directory containing a `SKILL.md` alongside optional scripts and data assets. This format is favored because it cleanly supports *progressive disclosure*: *metadata*, specifically each skill's name and one-line description (typically under 100 tokens), is exposed at startup, the full `SKILL.md` *instructions* are read only when the agent selects the skill, and *resources* stay on disk until accessed. CORPUS2SKILL builds directly on this representation, mapping the corpus onto a forest of such skill directories.

From procedural to informational skills. In prior work, skills encode *how to perform a task*: workflows, decision trees, or code patterns. CORPUS2SKILL repurposes the same mechanism to encode *what a corpus contains*: each skill directory represents a topical partition with `SKILL.md` and `INDEX.md` files at increasing detail, and the agent

[1] https://docs.anthropic.com/en/docs/agents-and-tools/agent-skills

[2] https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills

follows the same progressive-disclosure workflow (description → SKILL.md → INDEX.md → document) but navigates an information hierarchy rather than executing procedural instructions. This shift from procedural to informational skills is the central design insight of CORPUS2SKILL.

3 Related Work

Cluster-based and structure-aware retrieval. Scatter/Gather (Cutting et al., 1992) pioneered cluster-based navigation (Liu and Croft, 2004) but required human selection. CORPUS2SKILL delegates this to an LLM agent. RAPTOR (Sarthi et al., 2024) recursively clusters and summarizes chunks into a tree retrieved by traversal or collapsed-tree vector search. We share the compile-time embed-cluster-summarize loop but materialize the hierarchy as navigable files and replace fixed traversal with agentic navigation. GraphRAG (Edge et al., 2024) and HiRAG (Huang et al., 2025) build knowledge graphs with community-level summaries but rely on graph retrieval at query time. RAG (Lewis et al., 2020; Karpukhin et al., 2020; Santhanam et al., 2022) and its structure-aware extensions, including Self-RAG (Asai et al., 2024), IRCoT (Trivedi et al., 2023), StructRAG (Li et al., 2025), BookRAG (Wang et al., 2026b), A-RAG (Du et al., 2026), SPD-RAG (Akay et al., 2026), NaviRAG (Dai et al., 2026), and HCAG (Wu and Deng, 2026), restructure, rerank (Sun et al., 2026c), or re-query retrieved evidence but still depend on retrieval or search infrastructure at serve time (Gao et al., 2023; Singh et al., 2025).

Tool-use agents and skill construction. Toolformer (Schick et al., 2023) and ReAct (Yao et al., 2023) establish LLMs' ability to select tools. We extend that paradigm with read-only file-browsing and document-lookup tools over a static hierarchy. Voyager (Wang et al., 2024), SkillX (Wang et al., 2026a), and EvoSkill (Alzubi et al., 2026) build skills from agent *trajectories*. CORPUS2SKILL builds them from *document corpora* via clustering and summarization, a fundamentally different input modality.

Positioning. CORPUS2SKILL is distinguished from prior active-navigation work (NaviRAG, HCAG) by its filesystem-based skill representation with progressive disclosure via SKILL.md/INDEX.md files, and by its reliance on standard code-execution tools rather than custom retrieval APIs (Appendix F provides a feature comparison).

4 CORPUS2SKILL Framework

CORPUS2SKILL operates in two phases: a one-time offline *compile phase* that transforms a corpus $\mathcal{D} = \{d_1, \ldots, d_N\}$ into a hierarchical skill forest $\mathcal{S} = \{s_1, \ldots, s_K\}$, and a per-query online *serve phase* where an LLM agent navigates this tree with full visibility into the corpus structure, as shown in Figure 2. Standard RAG selects $\mathcal{D}_q \subseteq \mathcal{D}$ via a black-box retrieval function $f(q, \mathcal{D}) \to \mathcal{D}_q$. CORPUS2SKILL replaces this with agent-driven navigation, where each skill s_k is a navigable directory tree of summaries covering a topical partition of $\mathcal{D}$: the skill's summary serves as a routing condition, the agent descends through progressively finer summaries before retrieving documents, and the SKILL.md/INDEX.md files together with the get_document tool form the reusable navigation interface.

4.1 Compile Phase

The compilation pipeline turns a raw corpus into a navigable forest of skills through a sequence of LLM-driven stages. Implementation details and full prompts are provided in Appendices A and C.

Document representation. The compiler accepts standard document formats (.md, .txt, .json, .jsonl) and assigns each document a content-hash identifier. Every document is passed through a lightweight LLM call that returns a compact *summary card* (title, one-line description, distinctive phrases). The card is concatenated with the truncated raw text and embedded with a sentence model (Reimers and Gurevych, 2019), so the vector reflects both literal content and an LLM-distilled gist, and the same cards reappear as INDEX.md row entries.

Building the hierarchy. The hierarchy is built bottom-up under two parameters: the branching ratio p and the maximum number of top-level clusters K. At each level the current n embeddings are partitioned into $\lceil n/p \rceil$ groups via K-Means (Lloyd, 1982) on L2-normalized vectors. Each document is assigned a single home cluster, but documents whose top-1 versus top-2 cosine gap falls below a margin τ are additionally cross-referenced in the runner-up cluster's index, so the agent can discover cross-cutting content from multiple navigation paths. Each cluster is then summarized by

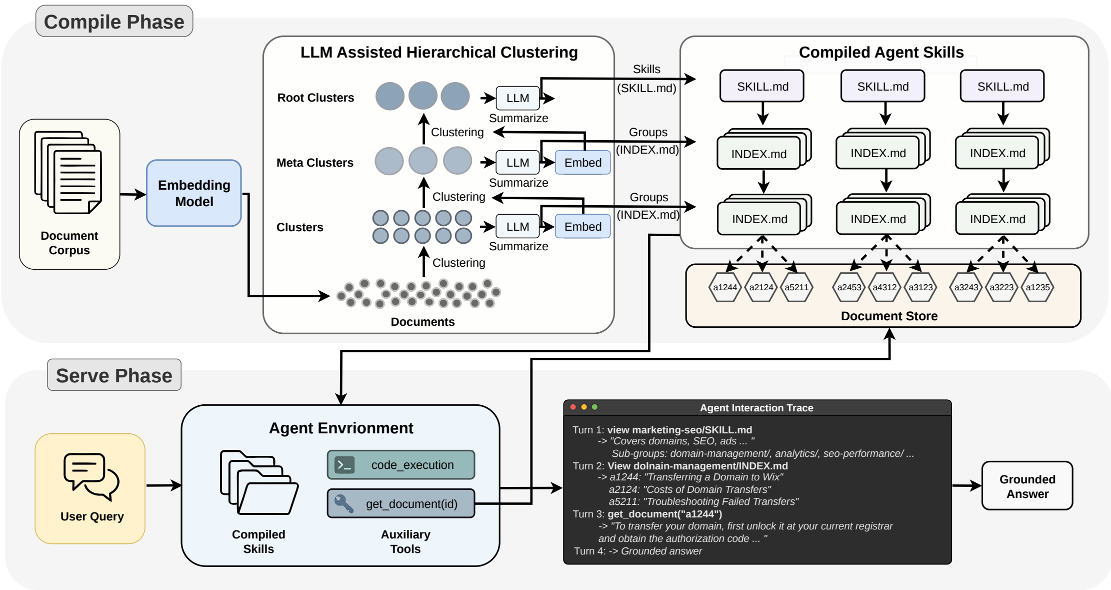

Figure 2: CORPUS2SKILL system architecture. The compile phase (top) embeds documents, builds a multi-level cluster hierarchy with LLM summarization at each level, and materializes the result as a forest of navigable skills. The serve phase (bottom) uses an LLM agent with two tools: file navigation (`code_execution`) and document lookup (`get_document`).

an LLM, and a *verify-and-repartition* pass examines all sibling summaries together to flag pairs that cover the same topic (*confusable*) or single clusters that visibly span multiple topics (*mixed*). Flagged clusters are repartitioned at the ID level and re-summarized. The cluster summaries, together with a small sample of member excerpts, are re-embedded and become the input to the next round of K-Means. The loop repeats until the number of remaining clusters falls below K. This embed-cluster-summarize recursion is inspired by RAPTOR (Sarthi et al., 2024), but materializes the hierarchy as browsable files rather than as a flat vector store.

Enabling cross-branch navigation. Each non-leaf node receives a short, filesystem-safe label (e.g., `wix-commerce-operations`) so directory names alone often identify the relevant branch. Once the hierarchy stabilizes, two enrichments are added. First, for every non-leaf cluster the top-k documents closest to the centroid are surfaced as *exemplars* in the navigation file, giving the agent concrete evidence of what each branch contains. Second, a batched LLM call tags every cluster summary with named entities and document types, aggregated into a corpus-wide `entity_index.json`. Each skill's navigation file includes a `## Related skills` block derived from entity-Jaccard overlap, so the agent can jump laterally to any other skill that mentions the same entity.

Skill tree construction. The hierarchy is materialized as a forest of K skill directories: each root cluster becomes a `SKILL.md` and each sub-cluster an `INDEX.md` listing further subgroups or leaf document rows. Full document text lives separately in a `documents.json` accessed via `get_document`, and a corpus-level `entity_index.json` maps each named entity to the skill paths that mention it (full layout in Appendix B). This separation keeps navigation files small (typically under 2 KB each) so the agent can survey 20 documents per `INDEX.md` cheaply before incurring the token cost of any full document.[3] Each `SKILL.md`/`INDEX.md` follows a uniform template with YAML frontmatter (`name`, `description`, `level`, `num_documents`) consumed by the Skills API for progressive disclosure, and the compilation prompts produce *routing-oriented* summaries (topic area, question types, key terms) so that each file functions as a decision point for the navigating agent.

Navigation complexity. The hierarchy depth grows as $O(\log_p N)$, so the agent traverses at most $L = \lceil \log_p N \rceil$ levels and inspects $O(p \cdot \log_p N)$ summaries rather than $O(N)$. For WixQA

[3] Anthropic's current Skills API imposes limits of 8 skills per request, 200 files and 30 MB per skill. These constraints informed our default parameters but are not fundamental to the framework design.

(N=6,221, p=10), this gives L=3 and roughly 30 summaries per query; the framework therefore scales to 100k-document corpora with one additional level (Appendix A).

4.2 Serve Phase

At serve time, the pre-compiled skill directories are uploaded to Anthropic's Skills API. The API implements progressive disclosure: skill names and one-line descriptions are pre-loaded into the agent's context, but full file content is loaded only when the agent explicitly reads a file. This design is critical for token efficiency. Pre-loading all navigation files would flood the context with thousands of tokens of irrelevant summaries. Loading nothing would leave the agent blind to the corpus. Progressive disclosure strikes a balance: the agent sees just enough (skill names and descriptions, approximately 200 tokens total) to make an informed first decision, then incrementally loads detail as it descends.

Tools. The agent receives two tools: (1) `code_execution`, which allows browsing `SKILL.md` and `INDEX.md` files (as well as the corpus-level `entity_index.json`) through standard shell utilities such as `ls`, `cat`, and `grep`, providing the agent with full visibility into the hierarchy, and (2) `get_document(doc_id)`, a custom tool that retrieves the full text of a document from the document store given its identifier. The two-tool design reflects the two types of information the agent needs: *navigational* (where to look) and *evidential* (what to cite). Keeping them separate means the agent can survey many document summaries cheaply before committing to the token cost of reading full documents.

Navigation workflow. Because skill descriptions are pre-loaded, the agent begins each query with a bird's-eye view of the corpus, knowing what topics exist and roughly how many documents each topic covers. The system prompt instructs the agent to keep at least two candidate skills alive before committing to a path, to consult `entity_index.json` when the query references a specific named entity, and to cross-jump via the `## Related skills` block when its initial branch turns out to be thin. A typical query proceeds in 2 to 3 turns:

1. The agent identifies the most relevant skill(s) from the pre-loaded descriptions (and from `entity_index.json` if the query mentions a specific entity) and reads the corresponding `SKILL.md`(s).
2. The agent navigates into the most relevant subgroup's `INDEX.md`, which lists document rows with titles, one-line descriptions, and distinctive phrases.
3. The agent calls `get_document` to retrieve the full text of one or more promising documents and synthesizes a grounded answer.

At each step, the agent makes an informed decision guided by summaries describing each branch. If a `SKILL.md` lists ten subgroups and the agent has only explored one, it knows nine remain, an awareness that flat retrieval cannot provide. This structural visibility enables two behaviors: *targeted backtracking* (abandoning an unpromising branch for a more relevant one) and *cross-branch synthesis* (combining evidence from multiple subgroups within or across skills), the latter aided by the related-skills cross-references and the entity index. The system prompt enforces a *grounding rule* (every claim must trace to a retrieved document, not a summary file) and a *concise-answer format* (word limits, no preamble). See Appendix D.

5 Experiments

5.1 Setup

Dataset and compilation. We evaluate on WixQA (Cohen et al., 2025), a customer-support QA benchmark with 6,221 support articles and 200 expert-written test queries with gold answers and gold article IDs. We compile WixQA with branching ratio p=10 and maximum top-level clusters K=10, producing a 3-level hierarchy with 6 top-level skills and 665 navigation files. Compilation takes 19.5 minutes on an RTX 4090 server.

Baselines. All methods use the same answer LLM and the same evaluation protocol. We compare five baselines spanning three retrieval paradigms: **BM25** (sparse keyword retrieval (Robertson and Zaragoza, 2009)), **Dense** (Qwen3-Embedding-0.6B (Zhang et al., 2025) with a FAISS index (Johnson et al., 2019)), **Hybrid** (RRF (Cormack et al., 2009) of BM25 and Dense), **RAPTOR** (hierarchical tree retrieval (Sarthi et al., 2024) with UMAP+GMM+BIC clustering), and **Agentic** (an LLM agent with iterative access to BM25, Dense, and Hybrid tools). The first three are single-shot (top-5 passages, one LLM call). RAPTOR uses collapsed tree retrieval with the *same* summarization LLM as our compiler (Claude Sonnet 4.6), and

Table 1: Main results on WixQA (200 queries). CORPUS2SKILL wins on F1, BERT, Fact., CtxR, CtxP and has the lowest Hallucination Rate among multi-turn methods; the F1 lead is bootstrap-significant (p<0.0001) with serve-time variance ±0.002 over three runs (Appendix K). Single-shot retrievers win Faith.$_{grd}$ / Halluc. because they always feed back a small grounded passage. Best in **bold**; ↑ cells green (darker=better), ↓ cells (Halluc., $/q) red (darker=worse). Multi-turn methods use Anthropic ephemeral prompt caching.

Method	Quality Metrics							Cost	
	F1↑	BERT↑	Fact.↑	Faith.$_{grd}$ ↑	CtxR↑	CtxP↑	Halluc.↓	Turns	$/q↓
BM25	0.345	0.830	0.472	0.831	0.406	0.549	0.005	1.00	**0.007**
Dense	0.364	0.833	0.537	0.889	0.448	0.659	**0.000**	1.00	0.007
Hybrid	0.361	0.832	0.524	0.851	0.416	0.637	0.005	1.00	0.007
RAPTOR	0.398	0.839	0.704	**0.909**	0.618	0.659	0.005	1.00	0.012
Agentic	0.378	0.843	0.705	0.528	0.452	0.649	0.500	5.06	0.082
CORPUS2SKILL	**0.456**	**0.862**	**0.767**	0.859	**0.708**	**0.829**	0.045	2.38	0.153

Agentic is allowed up to 10 rounds. Section 5.4 additionally controls for the compile-time encoder in both directions. Full implementation in Appendix H.

Metrics. We use a seven-metric suite. Quality: *Token F1* (token precision/recall harmonic mean against gold), *BERTScore-F1* (Zhang et al., 2020), and *Factuality* (LLM-as-judge (Zheng et al., 2023) 1–5 vs. gold, normalized to 0–1). Grounding: three RAGAS-style (Es et al., 2024) LLM-judge metrics (*Faithfulness$_{grd}$*, *Context Recall* (CtxR), *Context Precision* (CtxP)) plus *Hallucination Rate*, the fraction of queries with Faithfulness$_{grd}$ < 0.6. We also report turn count and $/query. Cross-method fairness details are in Appendix I. Every win/tie/loss statement follows one rule: a dataset is a *win* if CORPUS2SKILL's Token F1 exceeds the best baseline's under a per-query paired bootstrap (10,000 resamples) at p<0.05, a *loss* in reverse, and a *tie* otherwise; per-dataset deltas and p-values are in Appendix K.

5.2 Main Results

Answer quality. CORPUS2SKILL achieves the highest score on every answer-quality metric and on both retrieval-coverage metrics (Table 1). Token F1 reaches 0.456 (a 21% relative gain over Agentic 0.378 and 25% over Dense 0.364). Three independent serve passes yield Token F1 0.458 ± 0.002, run-to-run noise roughly 30× smaller than the gap to the strongest baseline, and the lead over RAPTOR is significant (p<0.0001; Appendix K). BERTScore-F1 is 0.862, ahead of Agentic (0.843), RAPTOR (0.839), and the flat retrievers (0.830–0.833). Factuality 0.767 exceeds the next-best methods Agentic (0.705) and RAPTOR (0.704) by 6 absolute points. CtxR/CtxP lead by clear margins (0.708/0.829 vs. 0.618/0.659 for RAPTOR and 0.452/0.649 for Agentic). Both hierarchical methods outperform the flat baselines, but CORPUS2SKILL extends RAPTOR's gains by a further 9% on Factuality and 15% on CtxR: cluster summaries surface thematic connections flat embeddings miss (Sun et al., 2026a), letting the agent match queries to relevant document *groups* rather than passages.

Faithfulness and hallucination. Single-shot retrievers score Faithfulness$_{grd}$ near 0.83–0.91 with Hallucination ≈0% by construction (one passage in, little room to drift). Multi-turn agents face a strictly harder grounding setting because any claim not exactly supported by an accumulated document is flagged, and Agentic suffers severely (0.528 / 50%). CORPUS2SKILL closes most of this gap (0.859 / 4.5%, within 0.05 of the best single-shot baseline) while keeping the answer-quality lead.

Cost. With Anthropic ephemeral prompt caching on the navigation context, CORPUS2SKILL costs **$0.153/query**, roughly 1.9× Agentic ($0.082) and 13–22× the single-shot retrievers ($0.007–$0.012); removing the cache discount raises this to $0.302 without changing the ordering (Appendix K). The Skills API loads navigation files in every call, so cached input tokens dominate. Typical queries reuse ∼70% of the prompt across consecutive turns and 100% across queries that target the same top-level branch (billed at one-tenth the base rate once cached). The ablations (Section 5.4) show that Haiku-4.5 or a tighter exploration budget cut cost further while keeping most of the quality lead. Two qualitative navigation patterns recur in the traces: *direct descent* (one `SKILL.md`, one `INDEX.md`, one document, stop) and *cross-*

Table 2: Generalization on 10 RAGBench subsets (200 queries each, or all available) over the *expanded retrieval pool*, grouped by the structural pattern that determines whether navigation helps. †/‡ marks a significant CORPUS2SKILL win/loss in Token F1 vs. the best baseline (paired bootstrap, $p<0.05$; Appendix K); unmarked CORPUS2SKILL rows are ties. Per-dataset best in **bold**; ↑ cells green, ↓ cells (Halluc., $/q) red.

Dataset	Method	Quality Metrics							Cost	
		F1↑	BERT↑	Fact.↑	Faith.$_{grd}$ ↑	CtxR↑	CtxP↑	Halluc.↓	Turns	$/q↓
(a) Single-domain, atomic-document corpora: CORPUS2SKILL leads (significantly on all but TechQA)										
Emanual	BM25	0.412	0.855	0.661	0.874	0.556	0.586	**0.000**	1.00	0.006
	Dense	0.421	0.855	0.691	0.894	0.650	0.668	**0.000**	1.00	0.006
	Hybrid	0.424	0.854	0.721	**0.895**	0.647	0.644	**0.000**	1.00	0.006
	Agentic	0.455	0.871	**0.841**	0.696	0.611	0.633	0.220	3.39	0.046
	CORPUS2SKILL†	**0.497**	**0.898**	0.817	0.835	**0.768**	**0.832**	0.091	2.57	0.060
DelucionQA	BM25	0.412	0.858	0.730	0.946	0.681	0.664	0.005	1.00	0.006
	Dense	0.440	0.862	0.821	**0.969**	0.750	0.753	**0.000**	1.00	0.006
	Hybrid	0.429	0.861	0.775	0.957	0.714	0.733	**0.000**	1.00	0.006
	Agentic	0.438	0.874	**0.841**	0.719	0.711	0.725	0.206	2.91	0.039
	CORPUS2SKILL†	**0.478**	**0.893**	0.821	0.921	**0.824**	**0.872**	0.022	2.64	0.110
TechQA	BM25	0.377	0.840	0.537	**0.781**	0.295	0.432	0.040	1.00	**0.009**
	Dense	0.382	0.840	0.560	0.774	0.297	0.499	0.050	1.00	0.009
	Hybrid	0.383	0.840	0.562	0.779	0.267	0.466	**0.040**	1.00	0.009
	Agentic	0.309	0.836	0.605	0.414	0.271	0.476	0.635	6.33	0.151
	CORPUS2SKILL	**0.396**	**0.854**	**0.709**	0.672	**0.496**	**0.649**	0.280	3.75	0.507
ExpertQA	BM25	0.312	0.825	0.486	**0.913**	0.439	0.408	**0.005**	1.00	**0.007**
	Dense	0.330	0.827	0.573	0.909	0.529	0.507	0.010	1.00	0.007
	Hybrid	0.326	0.827	0.552	0.898	0.478	0.463	0.015	1.00	0.007
	Agentic	0.288	0.838	0.709	0.517	0.442	0.457	0.508	4.39	0.085
	CORPUS2SKILL†	**0.356**	**0.854**	**0.764**	0.666	**0.565**	**0.589**	0.225	3.88	0.183
ClapNQ	BM25	0.325	0.849	0.735	0.940	0.766	0.406	**0.000**	1.00	**0.005**
	Dense	0.333	0.850	0.774	0.934	0.788	0.420	0.005	1.00	0.005
	Hybrid	0.333	0.850	0.756	0.938	0.782	0.411	0.005	1.00	0.005
	Agentic	0.308	0.855	0.756	0.736	0.710	0.421	0.161	2.52	0.027
	CORPUS2SKILL†	**0.468**	**0.893**	**0.924**	**0.911**	**0.937**	**0.878**	0.015	2.03	0.080
(b) Heterogeneous or numerical-table corpora: statistical ties										
CovidQA	BM25	0.204	0.785	0.546	0.905	0.542	0.517	0.050	1.00	**0.006**
	Dense	0.215	0.804	0.635	**0.927**	0.634	**0.602**	**0.030**	1.00	0.006
	Hybrid	**0.216**	0.795	0.613	0.924	0.608	0.602	0.035	1.00	0.006
	Agentic	0.206	**0.849**	**0.651**	0.612	0.549	0.617	0.363	4.13	0.058
	CORPUS2SKILL	0.208	0.847	0.456	0.735	0.405	0.426	0.255	6.85	0.436
FinQA	BM25	0.359	0.847	0.340	**0.940**	0.358	0.413	0.015	1.00	**0.005**
	Dense	**0.400**	0.852	0.520	0.924	0.514	0.479	0.020	1.00	0.006
	Hybrid	0.389	0.853	0.440	0.936	0.438	0.481	**0.005**	1.00	0.006
	Agentic	0.368	0.861	**0.576**	0.618	0.512	0.506	0.420	4.36	0.076
	CORPUS2SKILL	0.392	**0.867**	0.574	0.770	**0.550**	**0.652**	0.175	4.51	0.495
(c) Open-domain factoid or homogeneous-tabular corpora: CORPUS2SKILL loses										
HAGRID	BM25	0.320	0.862	0.815	0.967	0.825	0.613	0.010	1.00	**0.005**
	Dense	0.322	0.862	0.855	**0.974**	**0.881**	0.627	**0.000**	1.00	0.005
	Hybrid	0.323	0.862	0.831	0.970	0.859	**0.641**	0.005	1.00	0.005
	Agentic	**0.356**	**0.879**	0.826	0.881	0.829	0.623	0.071	2.29	0.022
	CORPUS2SKILL‡	0.188	0.838	0.358	0.787	0.220	0.200	0.260	1.05	0.021
TatQA	BM25	0.307	0.847	0.468	**0.965**	0.471	0.489	**0.005**	1.00	**0.005**
	Dense	0.312	0.846	0.527	0.935	0.508	0.498	0.020	1.00	0.006
	Hybrid	0.315	0.848	0.536	0.951	0.522	0.537	0.010	1.00	0.005
	Agentic	**0.355**	**0.866**	**0.612**	0.638	**0.537**	0.510	0.390	3.02	0.045
	CORPUS2SKILL‡	0.240	0.847	0.388	0.706	0.367	0.499	0.285	6.92	1.475
CUAD	BM25	**0.314**	0.824	0.277	0.962	0.181	0.477	**0.000**	1.00	**0.007**
	Dense	0.304	0.819	**0.279**	**0.963**	0.167	**0.742**	0.005	1.00	0.008
	Hybrid	0.308	0.821	0.277	0.945	0.195	0.685	**0.000**	1.00	0.007
	Agentic	0.249	0.819	0.270	0.402	0.177	0.706	0.695	4.27	0.080
	CORPUS2SKILL‡	0.272	**0.828**	0.237	0.643	**0.217**	0.439	0.430	5.80	0.502

branch exploration via `entity_index.json` or the `## Related skills` block (full traces in Appendix E).

5.3 Generalization to RAGBench

To probe how CORPUS2SKILL behaves outside its target domain, we run the same five methods on ten RAGBench (Friel et al., 2024) subsets (CovidQA (Möller et al., 2020), DelucionQA (Sadat et al., 2023), Emanual (Nandy et al., 2021), ExpertQA (Malaviya et al., 2024), TechQA (Castelli et al., 2020), FinQA (Chen et al., 2021), CUAD (Hendrycks et al., 2021), HAGRID (Kamalloo et al., 2023), TatQA (Zhu et al., 2021), ClapNQ (Rosenthal et al., 2025)), sampling

Table 3: Ablations on WixQA: *(a)* cluster structure, *(b)* agent exploration budget, *(c)* serving LLM (all 200 queries), *(d)* compiler LLM (50-query subset; compile cost \$9.5 vs. \$1.97). Per-panel best in **bold**.

Variant	F1↑	BERT↑	Fact.↑	CtxR↑	Halluc.↓	Turns	\$/q↓
(a) Cluster structure (p/skills/depth)							
Narrow (5/2/5)	**0.462**	**0.864**	0.758	0.690	0.080	2.38	**0.133**
Default (10/6/3)	0.456	0.862	**0.767**	**0.708**	**0.045**	2.38	0.153
Wide (20/4/3)	0.382	0.847	0.552	0.414	0.245	3.69	0.565
(b) Agent exploration budget (max rounds)							
5 rounds	0.455	0.863	**0.774**	0.702	0.065	2.35	0.145
10 rounds	**0.459**	**0.863**	0.754	0.693	0.065	2.36	**0.142**
20 rounds	0.456	0.862	0.767	**0.708**	**0.045**	2.38	0.153
(c) Serving LLM (default tree, max_turns=20)							
Sonnet 4.6	**0.456**	**0.862**	**0.767**	0.708	**0.045**	2.38	0.153
Haiku 4.5	0.420	0.856	0.671	**0.722**	0.180	2.92	**0.093**
(d) Compiler LLM (same serve setup, 50-query subset)							
Sonnet 4.6	**0.433**	**0.864**	–	0.665	0.082	–	–
Haiku 4.5	0.421	**0.864**	–	**0.688**	**0.020**	–	–

Table 4: Compile-enrichment decomposition on WixQA (200 queries per variant); top group: compile-resource components, bottom: navigational structure.

Variant	F1↑	Fact.↑	CtxR↑	CtxP↑
Full	0.456	0.767	0.708	0.829
− verify-and-repartition	0.456	0.760	0.677	0.821
− doc summary cards	0.463	0.780	0.695	0.811
− 8B encoder (use 0.6B)	0.461	0.751	0.702	0.840
− exemplars	0.401	0.609	0.514	0.676
− soft assignment	0.333	0.397	0.336	0.413
− entity index	0.312	0.359	0.278	0.344

200 test queries per subset (or all available). RAGBench is published as a generator-evaluation benchmark with per-row gold contexts. To convert it into a retrieval benchmark we build the *expanded retrieval pool* as the union of all train+val+test documents (deduplicated by content hash, pool sizes 221–12,727). This is intentionally harder than the original RAGBench protocol: every test query competes against thousands of distractors.

Corpus navigation is not one-size-fits-all. Under the paired-bootstrap criterion of Section 5.1, CORPUS2SKILL wins on 5 of the 11 datasets (WixQA +0.059, Emanual +0.061, DelucionQA +0.034, ExpertQA +0.027, ClapNQ +0.135; all $p \leq 0.010$), ties on 3, and loses on 3 (Table 2, per-dataset statistics in Appendix K). The split is not random but instead tracks the *structure* of the underlying corpus, and three regimes separate wins from losses. (i) *Single-domain, topically coherent corpora* where each document covers one feature or topic (WixQA, Emanual, DelucionQA, TechQA, ExpertQA, ClapNQ): CORPUS2SKILL leads every retrieval baseline, significantly so on all but TechQA (+0.013, p=0.14), with ClapNQ the most striking case (0.468 vs. Dense 0.333, +0.135 absolute). (ii) *Heterogeneous or numerical-table corpora* (CovidQA, FinQA): all methods sit within 0.01 F1 of the best baseline ($p \geq 0.33$), and these are statistical ties. (iii) *Open-domain factoid or homogeneous-tabular corpora* (HAGRID, TatQA, CUAD): CORPUS2SKILL loses by 0.05–0.17 F1 ($p < 0.001$) because the level-1 cluster summaries collapse onto generic labels (HAGRID's open-domain mix), describe near-identical units (TatQA's repeated SEC tables), or partition a single long document across multiple clause types (CUAD).

Grounding and positioning. On grounding, CORPUS2SKILL's per-dataset Faithfulness$_{\text{grd}}$ / Hallucination Rate sit between single-shot retrievers (which feed back one grounded passage by construction) and the Agentic baseline, consistently halving Agentic's hallucination rate while staying competitive with single-shot grounding on the datasets where it leads on F1. We therefore position CORPUS2SKILL as a complementary primitive rather than a universal replacement for retrieval: preferred for curated, single-domain knowledge bases (this paper's deployment target), with flat retrieval or RAPTOR retained as the default for open-domain or homogeneous-table pools. The HAGRID/CUAD/TatQA losses are structural, and motivate the follow-on directions in the Limitations.

5.4 Ablation Studies

We ablate compile-time and serve-time design choices on WixQA: cluster structure, exploration budget, serving and compiler LLMs (Table 3), encoder choice, and the compile-time enrichments (Table 4). All variants share the two-tool agent configuration and are directly comparable to Table 1.

Cluster structure (panel a). Varying p at K=10 produces narrow (p=5), default (p=10), and wide (p=20) trees. Narrow and default land within ≈1 F1 point (0.462 vs. 0.456) on every quality metric, and narrow is slightly cheaper (\$0.133 vs. \$0.153). The *wide* shape collapses to F1 0.382 (−0.07), CtxR 0.414 (−0.29), Hallucination 24.5%, 3.7 turns / \$0.57 per query: **once a level-1 cluster must summarize hundreds of topically-mixed documents its summary becomes too generic to route on**, reproducing the TatQA failure mode of Section 5.3 in miniature.

Agent exploration budget (panel b). Varying max rounds (5/10/20) has minimal impact on quality (within 0.005 F1, avg. 2.4 turns regardless of budget) because the hierarchy is well-organized enough that the agent rarely needs extended exploration. A larger budget mainly helps grounding: Hallucination drops monotonically from 6.5% (budget 5) to 4.5% (budget 20), suggesting that the extra rounds are used to verify rather than research.

Serving LLM choice (panel c). Swapping Sonnet 4.6 for Haiku 4.5 (same compiled tree) cuts cost \$0.153 $\rightarrow$ \$0.093 (-39%) while retaining 92% of Sonnet's F1 (0.420) and 87% of its Factuality (0.671). CtxR actually *increases* (0.708 $\rightarrow$ 0.722) because Haiku tends to read one more document before answering. The tradeoff is grounding: Haiku's Hallucination Rate is $4\times$ Sonnet's, so Sonnet remains the right choice for compliance-sensitive deployments. **The Haiku variant is the cheapest CORPUS2SKILL configuration we measure and still substantially beats every flat retrieval baseline on F1 by 2–6 absolute points**, evidence that the compiled tree, not the navigator's sophistication, drives retrieval performance.

Compiler LLM (panel d). Swapping the *entire* compile LLM from Sonnet 4.6 to Haiku 4.5 retains 97% of F1 at one-fifth the compile cost: neither side of the pipeline needs a frontier model. Compile-time summarization errors are also contained: summaries only *route* the agent, while answers are grounded in the retrieved raw documents, so a weak summary can cost recall but cannot inject unsupported content (cf. Appendix G).

Encoder choices. CORPUS2SKILL compiles with Qwen3-Embedding-8B; the baselines index with the 0.6B variant (Appendix H). A 2×2 encoder cross rules this asymmetry out: compiled with the baselines' 0.6B encoder, CORPUS2SKILL scores 0.461 F1 vs. 0.458 ± 0.002 for the 8B compile (p=0.37), while the $13\times$ larger encoder helps the baselines by at most $+0.023$ F1. Against RAPTOR, whose recursive summarization already uses the same Sonnet 4.6 summarizer as our compiler, CORPUS2SKILL leads by $+0.064$ F1 matched at 0.6B and $+0.035$ at 8B (both p<0.0001; full tables in Appendix K).

Component decomposition. Removing one compile-time enrichment at a time (Table 4) separates the two groups cleanly: dropping the entity index takes F1 from 0.456 to 0.312, soft assignment to 0.333, and exemplars to 0.401, while verify-and-repartition is F1-neutral and removing the summary cards or the 8B encoder shifts F1 by at most 0.007 (n.s.). The gains come from the navigable structure; compile-side resources contribute little.

Table 5: Token F1 on CovidQA as its pool grows 5k $\rightarrow$ 50,000 documents (same 200 queries and compiler). The 50k compile costs \$54.1 / 76 min (\$1.08/1k docs vs. \$3.33 at 6k) and adds one hierarchy level.

Method	5k pool	50k pool	Δ
CORPUS2SKILL	0.208	0.191	-0.017
Dense	0.215	0.187	-0.028
Hybrid	0.216	0.175	-0.041
BM25	0.204	0.169	-0.035

5.5 Scaling to 50k Documents

Growing CovidQA's pool to 50,000 documents with in-domain CORD-19 abstracts ($8\times$ our largest corpus, same compiler and protocol) adds exactly one hierarchy level, matching the $O(\log_p N)$ prediction, at sub-linear compile cost (Table 5); all 200 queries navigate the larger tree without error, and the compact-mode output fits production Skills API limits. With 45,000 same-domain distractors in the pool, every method degrades; navigation degrades least, leaving CORPUS2SKILL highest in F1, tied with Dense (p=0.74) and above BM25 (p=0.04), so CovidQA remains in the tie regime at $10\times$ scale.

6 Conclusion

We introduced CORPUS2SKILL, a system-level retrieval architecture that compiles a document corpus into a hierarchical skill directory and replaces embedding-based retrieval with agent-driven navigation. On WixQA, it improves answer quality and grounding over flat, hierarchical, and agentic baselines. An eleven-dataset study bounds the scope: navigation significantly wins on five datasets, ties on three, and loses on three. It suits single-domain collections with a recoverable taxonomy, while flat retrieval remains preferable where top-level clustering collapses. Controlled ablations trace the gains to the compiled tree and its enrichments, and these conclusions hold at 50,000 documents. Because the output is plain skill directories requiring no vector database, it integrates directly into existing agent deployments. For bounded corpora with recoverable topical structure, making a corpus *navigable* complements making it *searchable*.

Limitations

Cost. Navigation file content is included in every API call, so input tokens accumulate across turns. With Anthropic ephemeral prompt caching (used throughout our results), CORPUS2SKILL costs \$0.153/query, about 1.9× the Agentic baseline; without caching this roughly doubles to \$0.302/query, though the ordering against the baselines is unchanged (Appendix K). Swapping to Haiku 4.5 reduces the cached cost to \$0.093 while retaining most of the quality lead (Section 5.4). The method therefore targets high-value queries where improved grounding justifies the cost premium. For high-volume, low-stakes queries, single-shot retrieval remains more cost-effective.

Routing errors. Although soft assignment and entity cross-linking (Section 4.1) let the agent discover cross-cutting content, our failure analysis (Appendix G) shows that top-level routing errors still account for the majority of failures. Documents spanning multiple topics can be cross-referenced in secondary branches, but the primary routing decision remains a bottleneck when the query targets the less obvious topic.

Compilation and corpus churn. The one-time compilation takes ∼20 minutes of wall-clock time for WixQA (6,221 documents), negligible for enterprise deployments handling thousands of daily queries, and per-document cost falls sub-linearly with corpus size (Section 5.5). The system assumes a mostly static corpus: incremental insertion, deletion, and hierarchy repair are not supported, so any change requires recompilation, and frequently changing collections face recurring compile costs. The cheaper Haiku 4.5 compiler (97% of F1 at one-fifth the cost, Section 5.4) lowers the cost of each recompilation.

Corpus scope. Our RAGBench study (Section 5.3) identifies three corpus shapes where navigation does not help: long extractive documents where answers are verbatim spans (CUAD), homogeneous tabular corpora whose cluster summaries collapse onto near-duplicate labels (TatQA), and open-domain pools where dense retrieval already surfaces sufficient evidence in a single shot (HAGRID). These are structural scope limits rather than implementation issues, and motivate hybrid per-query routing between navigation and retrieval.

References

Yagiz Can Akay, Muhammed Yusuf Kartal, Esra Alparslan, Faruk Ortakoyluoglu, and Arda Akpinar. 2026. SPD-RAG: Sub-agent per document retrieval-augmented generation. *arXiv preprint arXiv:2603.08329*.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. EvoSkill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*.

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

Vittorio Castelli, Rishav Chakravarti, Saswati Dana, Anthony Ferritto, Radu Florian, Martin Franz, Dinesh Garg, Dinesh Khandelwal, Scott McCarley, Mike McCawley, Mohamed Nasr, Lin Pan, Cezar Pendus, John Pitrelli, Saurabh Pujar, Salim Roukos, Andrzej Sakrajda, Avirup Sil, Rosario Uceda-Sosa, and 2 others. 2020. The TechQA dataset. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1269–1278.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711.

Dvir Cohen, Lin Burg, Sviatoslav Pykhnivskyi, Hagit Gur, Stanislav Kovynov, Olga Atzmon, and Gilad Barkan. 2025. WixQA: A multi-dataset benchmark for enterprise retrieval-augmented generation. *arXiv preprint arXiv:2505.08643*.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. *Proceedings of the 32nd International ACM SIGIR Conference*, pages 758–759.

Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. 1992. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM SIGIR Conference*, pages 318–329.

Jihao Dai, Dingjun Wu, Yuxuan Chen, Zheni Zeng, Yukun Yan, Zhenghao Liu, and Maosong Sun. 2026. NaviRAG: Towards active knowledge navigation for retrieval-augmented generation. *arXiv preprint arXiv:2604.12766*.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Shaohan Wang, Pengyu Wang, Xiaorui Wang, and Zhendong Mao. 2026. A-RAG: Scaling agentic retrieval-augmented generation via hierarchical retrieval interfaces. *arXiv preprint arXiv:2602.03442*.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A Graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158.

Robert Friel, Masha Belyi, and Atindriyo Sanyal. 2024. RAGBench: Explainable benchmark for retrieval-augmented generation systems. *arXiv preprint arXiv:2407.11005*.

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

Dan Hendrycks, Collin Burns, Anya Chen, and Spencer Ball. 2021. CUAD: An expert-annotated NLP dataset for legal contract review. In *Advances in Neural Information Processing Systems Track on Datasets and Benchmarks*.

Haoyu Huang, Yongfeng Huang, Junjie Yang, Zhenyu Pan, Yongqiang Chen, Kaili Ma, Hongzhi Chen, and James Cheng. 2025. Retrieval-augmented generation with hierarchical knowledge. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 6044–6060.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. 2026. SoK: Agentic skills – beyond tool use in LLM agents. *arXiv preprint arXiv:2602.20867*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Ehsan Kamalloo, Aref Jafari, Xinyu Zhang, Nandan Thakur, and Jimmy Lin. 2023. HAGRID: A human-LLM collaborative dataset for generative information-seeking with attribution. *arXiv preprint arXiv:2307.16883*.

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. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Zhuoqun Li, Xuanang Chen, Haiyang Yu, Hongyu Lin, Yaojie Lu, Qiaoyu Tang, Fei Huang, Xianpei Han, Le Sun, and Yongbin Li. 2025. StructRAG: Boosting knowledge intensive reasoning of LLMs via inference-time hybrid information structurization. In *International Conference on Learning Representations*.

Xiaoyong Liu and W. Bruce Croft. 2004. Cluster-based retrieval using language models. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 186–193.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

Chaitanya Malaviya, Subin Lee, Sihao Chen, Elizabeth Sieber, Mark Yatskar, and Dan Roth. 2024. ExpertQA: Expert-curated questions and attributed answers. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3025–3045.

Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Geoffrey J. McLachlan and David Peel. 2000. *Finite Mixture Models*. Wiley.

Timo Möller, Anthony Reina, Raghavan Jayakumar, and Malte Pietsch. 2020. COVID-QA: A question answering dataset for COVID-19. In *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*.

Abhilash Nandy, Soumya Sharma, Shubham Maddhashiya, Kapil Sachdeva, Pawan Goyal, and Niloy Ganguly. 2021. Question answering over electronic devices: A new benchmark dataset and a multi-task learning based QA framework. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4600–4609.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992.

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

Sara Rosenthal, Avirup Sil, Radu Florian, and Salim Roukos. 2025. CLAPnq: Cohesive long-form answers from passages in natural questions for RAG systems. *Transactions of the Association for Computational Linguistics*, 13:53–72.

Mobashir Sadat, Zhengyu Zhou, Lukas Lange, Jun Araki, Arsalan Gundroo, Bingqing Wang, Rakesh Menon, Md Rizwan Parvez, and Zhe Feng. 2023. DelucionQA: Detecting hallucinations in domain-specific question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 822–835.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 3715–3734.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Gideon Schwarz. 1978. Estimating the dimension of a model. *The annals of statistics*, pages 461–464.

Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V. Vasilakos. 2025. Agentic retrieval-augmented generation: A survey on agentic RAG. *arXiv preprint arXiv:2501.09136*.

Yiqun Sun, Qiang Huang, Anthony Kum Hoe Tung, and Jun Yu. 2026a. Position: Text embeddings should capture implicit semantics, not just surface meaning. In *Forty-third International Conference on Machine Learning Position Paper Track*.

Yiqun Sun, Pengfei Wei, and Lawrence B. Hsieh. 2026b. Information retrieval misses the mark for LLM agents. *Available at SSRN 6903579*.

Yiqun Sun, Pengfei Wei, and Lawrence B. Hsieh. 2026c. Very efficient listwise multimodal reranking for long documents. In *Forty-third International Conference on Machine Learning*.

Yixuan Tang, Yuanyuan Shi, Yiqun Sun, and Anthony Kum Hoe Tung. 2025. Uncovering the bigger picture: Comprehensive event understanding via diverse news retrieval. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33939–33957.

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 the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. 2026a. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Shu Wang, Yingli Zhou, and Yixiang Fang. 2026b. BookRAG: A hierarchical structure-aware index-based approach for retrieval-augmented generation on complex documents. *Proceedings of the VLDB Endowment*, 19(9):2358–2371.

Yusen Wu and Xiaotie Deng. 2026. HCAG: Hierarchical abstraction and retrieval-augmented generation on theoretical repositories with LLMs. *arXiv preprint arXiv:2603.20299*.

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 *International Conference on Learning Representations*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations (ICLR)*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287.

A Compilation Details

Top-level skill labels. The 6 skills produced by the default configuration (p=10, K=10) on the WixQA corpus are shown in Table 6.

Table 6: Top-level skills produced by CORPUS2SKILL on WixQA.

Skill Label	Docs	Coverage
commerce-operations	1,682	Commerce, POS, payments
platform-overview	1,770	Editors, CMS, design
platform-ecosystem	1,797	Domains, billing, lifecycle
members-community	256	Members, gated access
bookings-platform	283	Bookings, scheduling
media-display	433	Galleries, storefront

Embedding. Documents and cluster summaries are embedded using Qwen/Qwen3-Embedding-8B, a 7.6B-parameter sentence encoder producing 4,096-dimensional embeddings. We use the document prompt template with batch size 32 on a single GPU.

Document summary cards. Before embedding, each document is processed by a lightweight *summary-card* LLM call (Claude Haiku 4.5, $\leq$250 output tokens, single API call per document, async with a semaphore of 20). The card is a JSON object with fields `title`, `one_line`, and `phrases` (up to four 1–4-word distinctive phrases such as product names or feature flags). The card is concatenated with the (truncated) raw text as a single string and embedded as one vector per document; the card is also retained on the document record and reused later as INDEX.md row metadata, so the per-document summarization cost is amortized across embedding, navigation, and serving.

Cluster summarization. Cluster summaries are produced by Claude Sonnet 4.6 with direct API calls (no agent loop). Level-1 summaries condense up to 15 document texts (600 characters each); deeper-level summaries condense up to 20 child summaries (300 characters each). Summarization is performed concurrently using async calls with a semaphore of 20. Internal-level clusters are re-embedded by concatenating their summary with a sampled set of member excerpts before passing the result to the next round of K-Means, so each higher-level vector reflects both the abstractive summary and surface-lexical signal.

Soft assignment. K-Means assignment uses a margin threshold τ=0.05 on cosine similarity: items whose top-1 versus top-2 centroid similarity gap is below τ retain their primary parent and additionally register a secondary parent. Secondary assignments are not duplicated as full documents; the runner-up parent's `INDEX.md` lists them as `@see` stubs pointing back to the canonical location. On the WixQA compile this produces 2,068 secondary stubs spread across 665 navigation files.

LLM verify-and-repartition. After each level's clusters are summarized, a verify-and-repartition LLM pass examines all sibling cluster summaries together (in batches of up to 60 clusters per call) and proposes ID-level reassignments where two summaries cover the same topic (*confusable*) or one summary spans multiple topics (*mixed*). The pass operates at levels 1 and 2 only and visits each cluster at most once per compile; affected clusters are re-summarized after reassignment. This adds one extra Sonnet call per affected sibling group and a re-summarization call per repartitioned cluster.

Exemplars. For every non-leaf cluster, the top-3 documents closest (by cosine similarity to the cluster centroid) are surfaced under an `## Example documents` section in the rendered `SKILL.md`/`INDEX.md`. Exemplars carry the document title and one-line description from the summary card.

Entity extraction and related skills. Once the cluster hierarchy is stable, a single batched Sonnet call set tags each cluster summary with up to 12 named entities (proper nouns, product names, error codes, file formats) and up to 4 document types. The entity tags are aggregated into a corpus-level `entity_index.json` that maps each entity to the skill paths that mention it (sorted by mention count). For every top-level skill, the top-3 sibling skills by entity-Jaccard overlap are recorded as `## Related skills`, giving the agent explicit cross-jump targets. On the WixQA compile this produces 2,328 distinct entries in `entity_index.json`.

File counts. The compilation produces 665 navigation files (`SKILL.md + INDEX.md`), a `documents.json` file storing the full text of all 6,221 documents (13 MB), and a `entity_index.json` mapping 2,328 entities to skill paths. Total output size is 16 MB.

Navigation complexity. The hierarchy depth grows as $O(\log_p N)$: with branching ratio p and N documents, the agent traverses at most $L = \lceil \log_p N \rceil$ levels to reach a leaf. At each level ℓ, the agent reads one navigation file summarizing $O(p)$ children. A single decision at level ℓ reduces the candidate set from $N/p^{\ell-1}$ to N/p^{ℓ}, yielding a multiplicative information funnel:

$$N \xrightarrow{\div p} \frac{N}{p} \xrightarrow{\div p} \frac{N}{p^2} \cdots \xrightarrow{\div p} \frac{N}{p^L} \approx 1 \quad (1)$$

The total summaries inspected scale as $O(p \cdot \log_p N)$ rather than $O(N)$. For WixQA (N=6,221, p=10), this gives L=3 levels and approximately 30 summaries to locate a single document among thousands. Note that this bound counts summaries inspected under an idealized single-path traversal; in practice, the actual token cost is higher because the Skills API includes previously loaded navigation files in each subsequent call, and the agent may explore multiple branches. The per-query costs reported in Section 5 reflect these real-world overheads. A 100,000-document corpus with p=10 would require only one additional level (L=5).

Document-level summarization prompt. Used at the lowest level to summarize a cluster of raw documents.

```
You are summarizing a cluster of related
documents from a knowledge base. Write a
2-3 sentence summary that captures:
1. The common TOPIC area these documents
   cover
2. The types of QUESTIONS these documents
   answer
3. Key TERMS or features mentioned across
   documents

Be specific and concrete. Name actual
features, products, or processes.

Documents ({N} total, showing up to 15):

--- Document 1 ---
{doc_text[:600]}

--- Document 2 ---
{doc_text[:600]}
[...]

Summary:
```

Cluster-level summarization prompt. Used at higher levels to summarize a group of sub-cluster summaries into a broader overview.

```
You are summarizing a level-{L} grouping
of {N} sub-groups from a knowledge base.
Each sub-group already has a summary below.

Write a 2-3 sentence overview that captures:
1. The broad DOMAIN these sub-groups cover
2. The range of TOPICS within this domain
3. What types of user QUESTIONS this group
   can answer

Be specific -- name the main product areas,
features, or workflows.

Sub-group summaries:

- Sub-group 1: {summary[:300]}
- Sub-group 2: {summary[:300]}
[...]

Overview:
```

Labeling prompt. Used to generate a filesystem-safe directory name from a cluster summary. The response is post-processed: lowercased, non-alphanumeric characters replaced with hyphens, truncated to 50 characters.

```
Generate a short (2-5 word) filesystem-safe
label for this cluster. Use lowercase,
hyphens instead of spaces. No quotes.

Summary: {summary[:500]}
```

B Skill Structure Examples

We show the full content of representative navigation files from the WixQA compilation (default configuration: p=10, K=10). These files are what the agent reads during navigation.

Root-level `SKILL.md`. This file sits at the top of the `skill-00-wix-commerce-operations` skill directory, the largest skill covering 1,682 documents across 11 subgroups. The agent reads this file in a single tool call and uses the subgroup summaries to decide which branch to explore.

```
---
name: skill-00-wix-commerce-operations
description: >
  This group covers the Wix commerce and
  business operations ecosystem, spanning
  the full range of tools for running
  transactions, managing services, and
  scaling a business on Wix. Topics include
  e-commerce store management, restaurant
  and hotel operations, event ticketing,
  point-of-sale systems, bookings and
  scheduling, pricing plans, payments and
  payment providers, tax and invoicing,
  promotions and discounts...
level: 3
num_documents: 1682
---

## Overview

This group covers the Wix commerce and
business operations ecosystem, spanning
the full range of tools for running
transactions, managing services, and
scaling a business on the Wix platform.
Topics include e-commerce, restaurant and
hotel operations, event ticketing, POS,
bookings, pricing plans, payments, tax,
invoicing, promotions, and multilingual
localization. Users can get answers about
setting up and configuring any Wix commerce
```

```
product and processing payments across
global providers.

## Contents

### Sub-groups (directories)

Read the INDEX.md in each sub-group to
understand what it covers.

- **group-00-wix-events-platform/**
  (138 docs): Wix Events platform: event
  creation, ticketing, registration, guest
  management, and promotion.
- **group-01-wix-restaurants-platform/**
  (232 docs): Wix Restaurants: menus,
  ordering, reservations, delivery, POS
  integrations, and fulfillment.
- **group-02-wix-hotels-bookings/**
  (109 docs): Wix Hotels and Bookings:
  room configuration, pricing, calendar
  sync, and OTA channel management.
- **group-03-wix-pos-ecosystem/**
  (167 docs): Wix POS ecosystem: hardware
  setup, payment processing, staff
  management, and inventory.
- **group-04-wix-payments-processing/**
  (113 docs): Payments lifecycle: account
  setup, chargebacks, fraud prevention,
  payouts, and subscription billing.
- **group-05-wix-billing-tax-management/**
  (110 docs): Financial and tax management:
  invoicing, quotes, Avalara, VAT/GST.
[... 5 more sub-groups omitted ...]
```

Mid-level `INDEX.md`. This file sits one level below the root, inside `group-04-wix-payments-processing/`, covering 113 documents across 10 leaf groups. The agent reads this after deciding from the `SKILL.md` above that payments are relevant.

```
---
name: wix-payments-processing
description: >
  Covers the broad domain of payments on
  the Wix platform, spanning the full
  lifecycle from account setup to
  transaction management, payouts, and
  compliance.
level: 2
num_documents: 113
---

## Overview

Topics range from setting up and verifying
Wix Payments accounts, enabling third-party
providers (PayPal, Stripe, Apple Pay,
Google Pay), and managing chargebacks and
fraud prevention, to payout schedules,
prohibited product restrictions,
subscription billing, and unsupported
payment features.

## Contents

### Sub-groups (directories)

- **group-00-wix-payments-chargebacks/**
  (12 docs): Chargeback management,
  transaction fees, and merchant practices.
- **group-01-wix-payment-setup/**
  (22 docs): Setup and troubleshooting of
  payment methods across commerce products.
- **group-02-wix-payments-payouts/**
  (15 docs): Payout management, settlement
  reports, and fund transfer timelines.
- **group-03-payment-flexibility-requests/**
  (9 docs): Feature requests for split,
  installment, and deferred payments.
- **group-04-wix-paypal-button-setup/**
  (7 docs): PayPal Buy Now and Donate
  button configuration.
- **group-05-wix-payments-account-setup/**
  (13 docs): Account setup, verification,
  and EIN/identity documents.
- **group-06-wix-payments-processing/**
  (10 docs): Payment processing issues,
  fraud prevention, and declined payments.
- **group-07-wix-payments-management/**
  (9 docs): Transaction dashboards, payment
  history, and settlement reports.
- **group-08-prohibited-payment-products/**
  (6 docs): Payment provider compliance and
  restricted product categories.
- **group-09-wix-subscription-billing/**
  (10 docs): Premium plan billing, renewal
  payments, and payment method updates.
```

Leaf-level `INDEX.md`. This file sits at the bottom of the hierarchy, inside `group-06-wix-payments-processing/` under the payments processing branch. It lists individual document IDs with titles and summaries. The agent reads this to select which documents to retrieve via `get_document`.

```
---
name: wix-payments-processing
description: >
  Payment processing issues, fraud
  prevention, and transaction management
  within Wix Payments.
level: 1
num_documents: 10
---

## Overview

These documents cover payment processing
issues, fraud prevention, and transaction
management within Wix Payments. They answer
questions about why payments are declined
(AVS mismatches, CVV errors, 3-D Secure
failures, suspected fraud), how to resolve
double charges, and how EMV chip liability
shift rules affect merchants.

## Contents

### Documents (10 items)

Use get_document with the doc_id to
read full content.

- `56ae7642b2a8c2ea` -- Best Practices
  for Accepting Card Payments
- `88f6b5848f6f662d` -- Troubleshooting
  Payment Failures and Declined
  Transactions
- `d22b8cf6e599e0b8` -- Double Charge on
  Bank Statement Explanation
- `448d7947c5a22489` -- Wix Payments:
  Liability Shift and Chip Card
  Requirements
- `bb73d9cb90385f12` -- Wix Payments
  Declined Payment Suspected Fraud
[... 5 more documents omitted ...]
```

C Compilation Prompts

The compilation pipeline uses three LLM prompts, each issued as a single-turn API call (no agent loop or tool use).

Document-level summarization. Used at the lowest level to summarize a cluster of raw documents.

```
You are summarizing a cluster of related
documents from a knowledge base. Write a
```

```
2-3 sentence summary that captures:
1. The common TOPIC area these documents
   cover
2. The types of QUESTIONS these documents
   answer
3. Key TERMS or features mentioned across
   documents

Be specific and concrete. Name actual
features, products, or processes.

Documents ({N} total, showing up to 15):

--- Document 1 ---
{doc_text[:600]}

--- Document 2 ---
{doc_text[:600]}
[...]

Summary:
```

Cluster-level summarization. Used at higher levels to summarize a group of sub-cluster summaries into a broader overview.

```
You are summarizing a level-{L} grouping
of {N} sub-groups from a knowledge base.
Each sub-group already has a summary below.

Write a 2-3 sentence overview that captures:
1. The broad DOMAIN these sub-groups cover
2. The range of TOPICS within this domain
3. What types of user QUESTIONS this group
   can answer

Be specific -- name the main product areas,
features, or workflows.

Sub-group summaries:

- Sub-group 1: {summary[:300]}
- Sub-group 2: {summary[:300]}
[...]

Overview:
```

Labeling. Used to generate a filesystem-safe directory name from a cluster summary. The response is post-processed: lowercased, non-alphanumeric characters replaced with hyphens, truncated to 50 characters.

```
Generate a short (2-5 word) filesystem-safe
label for this cluster. Use lowercase,
hyphens instead of spaces. No quotes.

Summary: {summary[:500]}
```

Document summary card. Issued once per document at the start of compilation. The returned JSON card is concatenated with the truncated raw text for embedding and is reused as INDEX.md row metadata.

```
You are summarizing a single document from
a knowledge base. Produce a compact JSON
'card' describing the document so an agent
can scan many cards quickly and pick the
relevant one.

Return ONLY a JSON object with exactly
these keys:
  title:    short descriptive title
            (5-12 words). Reuse the
            document's own title if it has
            one, otherwise infer.
  one_line: a single sentence (max 25
            words) describing what the
            document covers and what kind
            of question it answers.
  phrases:  list of 2-4 short distinctive
            phrases (2-6 words each) that
            are characteristic of this
            document and unlikely to appear
            in unrelated documents (e.g.
            product names, schema names,
            specific entities, code
            identifiers, section headers).

Be specific and concrete. Do not include
generic phrases like 'knowledge base
article' or 'support document'.

Document:
{document}

JSON card:
```

Verify-and-repartition. Issued once per hierarchy level (skipped beyond 60 sibling clusters per call). The model is asked to flag confusable cluster pairs and mixed individual clusters, and to return ID-level reassignments. Changes are applied verbatim; affected clusters are re-summarized.

```
You are auditing a clustering of items at
level {level} of a navigable knowledge
hierarchy. Each cluster has a label, a
short summary, and a list of member item
IDs with brief titles.

Two failure modes to look for:
  (a) CONFUSABLE: two or more clusters
      whose labels/summaries describe
      essentially the same topic.
  (b) MIXED: a single cluster whose
      members visibly belong to multiple
      distinct sub-topics that should be
      split.

For each problematic GROUP (a set of
confusable cluster IDs, or a single mixed
cluster ID), output a fresh partition of
its members by ID. Use ONLY the IDs already
listed in that group; do not invent IDs or
move items across non-flagged boundaries.

Return ONLY a JSON object with this shape:
{ "changes": [
  { "old_cluster_ids": ["L1-C2", "L1-C7"],
    "new_partition": {
      "label_a": ["id1","id2"],
      "label_b": ["id3","id4"]
    }
  }
] }
If no changes are needed, return
{ "changes": [] }.

Clusters at level {level}:
{clusters_block}

JSON:
```

Entity and doc-type extraction. Issued once per cluster, batched concurrently, after the hierarchy stabilizes. The returned named entities are aggregated into `entity_index.json` and drive the per-skill `Related skills` cross-references.

```
You are tagging a topic cluster from a
knowledge base with the entities and
document types it contains.

Extract ONLY two categories:
  named_entities: proper-noun product
    names, feature names, person names,
    place names, organization names,
    schema names, specific identifiers.
    Exclude generic concepts (e.g.
```

```
    'authentication', 'payments' are
    generic; 'Stripe Connect',
    'Wix Bookings' are named entities).
  doc_types: kinds of documents present
    in the cluster, e.g. 'FAQ', 'release
    notes', 'balance sheet', 'contract
    clause', 'API reference', 'tutorial',
    'changelog'.

Return ONLY a JSON object:
{ "named_entities": ["..."],
  "doc_types": ["..."] }
Aim for 3-10 named entities and 1-3 doc
types. Use lowercase, no duplicates.

Cluster label: {label}
Cluster summary: {summary}
Example item titles:
{titles_block}

JSON:
```

D Serving System Prompt

The serving agent receives the following system prompt:

```
# Corpus-Grounded Support Agent
#   (Hierarchical Navigation)

You are a knowledge agent that answers
questions by navigating a hierarchical
skill directory. You explore a structured
file tree where documents are organized
into topic clusters, and you have access
to an entity cross-index and a leaf-level
search tool to triangulate evidence.

## Hard Rules

- Every factual claim must trace to a
  document you retrieved via get_document.
- SKILL.md / INDEX.md files are NAVIGATION
  AIDS -- they tell you where to look,
  not what to say.
- Never fabricate steps, URLs, prices, or
  specifics not found in documents.
- Never guess. If you cannot find relevant
  content after thorough exploration,
  say so.

## Skill directory layout

skill-XX-topic/
  SKILL.md          <- top-level summary,
                       exemplar docs,
                       related skills,
                       index
  group-YY-subtopic/
    INDEX.md        <- sub-group summary
                       + child index
    group-ZZ/
      INDEX.md      <- leaf summary +
                       document rows
                       (id + title +
                       phrases)

Each SKILL.md / INDEX.md may contain:
- ## Overview      -- what this skill
                       covers
- ## Related skills -- sibling skills that
                       share entities;
                       cross-jump here when
                       the current branch
                       is thin
- ## Entities & document types
                    -- quick scan of named
                       entities present
- ## Example documents in this skill
                    -- representative items
                       (read one of these
                       to learn what kind
                       of content lives
                       here)
- ## Contents       -- sub-groups and/or
                       document rows
- ### See also      -- @see stubs
                       (documents whose
                       primary entry is
                       elsewhere)

A corpus-level entity_index.json is also
present at the corpus root. It maps each
named entity to the skill paths that
mention it; use it to find ALL skills
relevant to a query entity rather than
only the first match.

## Navigation Strategy (multi-candidate)

1. SCAN at least 2 candidate top-level
   skills before committing to a path.
   - If the query mentions a specific
     entity, first
     `cat entity_index.json | grep -i
       <entity>`
     to find every skill that references
     it, and read the SKILL.md of the top
     2 by mention count.
   - Otherwise read the SKILL.md of the 2
     most-plausible top-level skills.
2. For each candidate, descend ONE level
   (read its first relevant INDEX.md)
   before pruning. Write down which
   candidates you keep and why.
3. At the leaf, use `cat ... | grep ...`
   against the INDEX.md doc rows to pick
   the most promising doc_ids -- the rows
   include the document title, one-liner,
   and key phrases, which is usually
   enough to triangulate without opening
   anything.
4. If two candidate paths return
   comparable evidence, retrieve documents
   from BOTH and explicitly compare. Pick
   the best answer; do NOT just take the
   first match.
5. If a candidate skill has a `## Related
   skills` block and your initial path is
   thin, cross-jump to a related skill
   before giving up.
6. Read at least one full document via
   get_document before answering.

## Tools

- Code execution: Use `ls`, `cat`,
  `cat ... | grep ...` to navigate the
  skills hierarchy and the
  entity_index.json.
- get_document(doc_id): Retrieve the full
  text of a document by its ID.

## Answer Format

- First sentence = direct answer.
  No preamble.
- Factual questions: 1-3 sentences
  (~80 words max).
- Procedural questions: numbered steps
  only (~150 words max).
- Plain text. No bold, headers, or
  dividers.
- One approach only. Do not present
  alternatives.
- Never add "contact support" or closing
  remarks.
```

E Full Navigation Traces

We present complete navigation traces with raw tool calls for four queries from the WixQA evaluation set (200 expert-written test queries, Claude Sonnet 4.6 serving agent).

Trace 1: Direct descent (Factuality 1.0, CtxR 1.0). Query: *"I need to switch my business type from sole prop to LLC in order to use an EIN."*

```
[Context: agent sees 6 skill descriptions]

--- Turn 1 ---
Agent thought: The query is about changing
business type for Wix Payments. The skill
"wix-commerce-operations" covers payments,
taxes, and business operations.

Tool: code_execution
  cat skill-00-wix-commerce-operations/
    SKILL.md

Result: 11 subgroups listed, including:
  - group-04-wix-payments-processing/
    (113 docs): "payments lifecycle: account
    setup, chargebacks, fraud prevention,
    payouts, and subscription billing"

Tool: code_execution
  cat group-04-wix-payments-processing/
    INDEX.md

Result: 10 leaf groups. Agent identifies
  group-05-wix-payments-account-setup
  covering account setup and verification.

Tool: code_execution
  cat group-05-wix-payments-account-setup/
    INDEX.md

Result: 13 documents listed, including:
  - d56cc797742e: "Switching Between
    Individual and Business Account Types
    in Wix Payments"

--- Turn 2 ---
Tool: get_document
  {"doc_id": "d56cc797742e7d0f"}

Result: Full article text (1,247 chars)
  describing that account type cannot be
  changed directly; must contact Wix
  Customer Care.

--- Answer ---
"You cannot change your account type
directly within Wix Payments. To switch
from sole proprietorship to LLC, contact
Wix Customer Care with your EIN and LLC
formation documents ready."

Navigation: 6221 -> 1682 -> 113 -> 13 -> 1
```

Trace 2: Multi-document retrieval (Factuality 1.0, CtxR 1.0). Query: *"How do I sync the hotel app with my calendars iCal link to allow visitors to book available dates?"*

```
[Context: agent sees 6 skill descriptions]

--- Turn 1 ---
Agent thought: Hotels and calendar sync
fall under "wix-commerce-operations" which
covers hotel operations and bookings.

Tool: code_execution
  cat skill-00-wix-commerce-operations/
    SKILL.md

Result: 11 subgroups listed, including:
  - group-02-wix-hotels-bookings/
    (109 docs): "Wix Hotels and Bookings:
    room configuration, pricing, calendar
    sync, and OTA channel management"

Tool: code_execution
  cat group-02-wix-hotels-bookings/
    INDEX.md

Result: 8 leaf groups. Agent identifies
  group-07-hotel-channel-manager-setup
  covering external calendar integration.

Tool: code_execution
  cat group-07-hotel-channel-manager-setup/
    INDEX.md

Result: 17 documents listed, including:
  - 5f02fc2f8937: "Importing Reservations
    from External Calendar (iCal)"
  - 741830b917b4: "Connecting Airbnb iCal
    to Your Site"

--- Turn 2 ---
Tool: get_document
  {"doc_id": "5f02fc2f8937d16a"}

Result: Step-by-step iCal import guide
  for original Wix Hotels.

Tool: get_document
  {"doc_id": "741830b917b47d07"}

Result: Airbnb iCal connection guide for
  Wix Hotels by HotelRunner.

--- Answer ---
"For original Wix Hotels: Go to Hotels
tab > Reservations > iCal, select room
type, paste iCal URL, click Save. For
HotelRunner: go to dashboard, navigate
to channel connections, add Airbnb iCal
link for each property..."

Navigation: 6221 -> 1682 -> 109 -> 17 -> 2
```

Trace 3: Mobile quick action (Factuality 1.0, CtxR 1.0). Query: *"I need to temporarily remove the quick action button from the mobile view."*

```
--- Turn 1 ---
Agent thought: Mobile site editing falls
under "wix-platform-overview" which covers
editors, CMS, and design.

Tool: code_execution
  cat skill-01-wix-platform-overview/
    SKILL.md

Result: 14 subgroups. Agent picks
  group-13-wix-mobile-apps covering
  mobile editor features.

Tool: code_execution
  cat group-13-wix-mobile-apps/INDEX.md

Result: Leaf groups covering mobile UI
  elements. Agent finds group-00 with docs
  on Quick Action Bar (6 docs).

Tool: code_execution
  cat group-00-mobile-quick-action-bar/
    INDEX.md

Result: 6 documents including:
  - bce5e3d829fe: "Adding and Setting Up
    Your Quick Action Bar"

--- Turn 2 ---
Tool: get_document
  {"doc_id": "bce5e3d829fe6eeb"}

--- Answer ---
"There is no hide option, only deletion.
Right-click the Quick Action Bar in the
mobile Editor and select Delete. To re-add
later, go to Mobile Tools and click Add."

Navigation: 6221 -> 1770 -> 60 -> 6 -> 1
```

Trace 4: Cross-branch exploration (Factuality 1.0, CtxR 1.0). Query: *"I want to change the currency for my course."*

```
[Context: agent sees 6 skill descriptions]

--- Turn 1 ---
Agent thought: "Course" suggests Online
Programs under "wix-platform-ecosystem",
```

```
and "currency" is a payments topic under
"wix-commerce-operations".

Tool: code_execution
  cat skill-02-wix-platform-ecosystem/
    SKILL.md

Result: 11 subgroups listed, including:
  - group-07-wix-programs-partnerships/
    (119 docs): "online programs, courses,
    pricing, and partner integrations"

Tool: code_execution
  cat group-07-wix-programs-partnerships/
    INDEX.md

Result: 6 leaf groups. Picks group-04
  covering online program setup (9 docs).

Tool: code_execution
  cat group-04-wix-online-programs-setup/
    INDEX.md

Result: 9 documents including:
  - 3b2e9b0c4273: "Wix Online Programs:
    Setting Course Pricing"

Tool: get_document
  {"doc_id": "3b2e9b0c4273f8c9"}

Result: Course pricing article. Mentions
  that currency is configured at the site
  level via Language & Region settings,
  not per-program.

--- Turn 2 ---
Agent thought: Need the actual currency
configuration steps. Navigate to payments.

Tool: code_execution
  cat skill-00-wix-commerce-operations/
    SKILL.md | grep -i currency

Result: group-09-payment-provider-
  integration mentions currency settings.

Tool: code_execution
  cat group-09-payment-provider-integration/
    group-03-wix-currency-settings/INDEX.md

Result: 12 documents including:
  - 29139bd7c379: "Setting Your Currency
    for Accepting Payments"

Tool: get_document
  {"doc_id": "29139bd7c379ff77"}

Result: Step-by-step currency configuration
  guide via Settings > Language & Region.

--- Answer ---
"To change the currency for your course,
update the site-level currency setting:
1. Go to Settings in your dashboard.
2. Click Language & Region.
3. Scroll to Currency and select your
   desired currency.
4. Click Save.
This applies to all Wix products including
Online Programs. Your currency must match
your payment provider's currency."

Navigation paths:
  6221 -> 1797 -> 119 ->  9 -> 1 (programs)
  6221 -> 1682 -> 268 -> 12 -> 1 (currency)
```

This trace demonstrates cross-branch navigation: the agent first explored the online-programs branch in one skill to understand the relationship between courses and currency, then cross-jumped to the payment-provider branch in a different skill to find the authoritative currency configuration steps. The two branches across two skills provided complementary information that neither alone would have answered the query fully.

F Positioning Comparison

Table 7 highlights three axes on which CORPUS2SKILL differs from prior work.

Input modality. Scatter/Gather, RAPTOR, GraphRAG, and CORPUS2SKILL all start from documents or text chunks, but Voyager and SkillX build skills from agent *trajectories*: successful action sequences or distilled experience. CORPUS2SKILL is the only system that converts a static document corpus into the skill format typically reserved for procedural knowledge.

Navigation mechanism. Scatter/Gather relies on human users to iteratively select clusters; RAPTOR and GraphRAG delegate navigation to fixed algorithms (vector similarity or graph traversal). CORPUS2SKILL is unique in delegating navigation to an LLM agent that can reason about which branch to explore, backtrack when a path is unproductive, and combine evidence across branches. This agent-driven navigation is what enables the trial-and-error behavior illustrated in Appendix E.

Serve-time infrastructure. All prior systems require dedicated infrastructure at query time: a cluster index, a vector database, or a graph database. CORPUS2SKILL eliminates this dependency entirely: the only runtime component is the LLM itself, which receives pre-compiled skill files via the API. This "LLM only" property simplifies deployment but shifts the cost burden to input tokens, as discussed in Section 6.

G Failure Analysis

We examine the 45 of the 200 WixQA test queries (22%) on which CORPUS2SKILL scores Factuality$\leq$0.4 or Context Recall$\leq$0.2, and categorize them into four failure modes.

Navigation miss (19 queries). The agent retrieves documents but none overlap with the gold context (Context Recall=0), indicating it descended into the wrong branch. For example, a query about connecting PayPal is routed to a general payments group instead of the payment-setup leaf; a query about changing a header background on scroll navigates to a layout group rather than the header-effects cluster. These misroutes typically

Table 7: Positioning of CORPUS2SKILL relative to prior work.

Aspect	Scatter/Gather	RAPTOR	GraphRAG	Voyager/SkillX	CORPUS2SKILL
Input	Documents	Text chunks	Documents	Trajectories	Documents
Organization	Flat clusters	Hier. tree	Knowledge graph	Skill library	Hier. skill tree
Navigation	Human user	Algorithm	Graph search	Vector sim.	LLM agent
Serve infra	Cluster index	Vector index	Graph DB	Vector DB	**LLM only**

occur at the *top level*, where broad skill summaries fail to distinguish closely related topics.

Partial navigation (16 queries). The agent's retrieved set partially overlaps with the gold ($0 <$Context Recall< 0.6) but includes too many non-relevant documents, diluting the context and producing a low-quality answer. The agent identified the correct general topic area but over-retrieved from neighboring leaf groups.

Synthesis error (5 queries). The agent retrieves contextually relevant documents (Context Recall$\geq$0.6) but produces a factually incorrect answer, typically by over-generalizing across multiple articles or misinterpreting conditional instructions as universal.

No documents retrieved (5 queries). Five queries resulted in zero or near-zero retrieval. These include off-domain queries ("How do I clear my browser cache?", a Gmail password reset question, Google Ads billing) where the agent could not locate a matching skill branch and declined to answer rather than hallucinating.

Takeaway. The dominant bottleneck remains initial topic routing: 19 of 45 failures stem from the agent choosing the wrong top-level branch. The narrow-tree ablation (Section 5.4), which produces finer-grained top-level clusters, partially mitigates this by providing sharper topic separation at the first navigation step.

H Baseline Implementation Details

All baselines use the same LLM (Claude Sonnet) for answer generation and the same evaluation prompts for fair comparison.

BM25. We implement BM25 sparse keyword retrieval using the `rank_bm25` library (Robertson and Zaragoza, 2009). Documents are tokenized and indexed at the full-article level. At query time, we retrieve the top-5 articles by BM25 score and pass them as context to a single LLM call.

Dense. We encode all documents using Qwen3-Embedding-0.6B (Zhang et al., 2025), producing 1024-dimensional embeddings stored in a FAISS flat index (Johnson et al., 2019). We use the document prompt template for encoding articles and the query prompt template for encoding questions. At query time, we retrieve the top-5 nearest neighbors by cosine similarity and pass them as context to a single LLM call.

Hybrid. We combine BM25 and Dense scores using Reciprocal Rank Fusion (Cormack et al., 2009) with k=60 and equal weighting (α=0.5). The fused ranking selects the top-5 passages, which are passed to a single LLM call.

RAPTOR. We faithfully implement the RAPTOR algorithm (Sarthi et al., 2024): UMAP (McInnes et al., 2018) for dimensionality reduction, Gaussian Mixture Model (GMM) soft clustering (McLachlan and Peel, 2000) with BIC-based cluster count selection (Schwarz, 1978), and recursive summarization of clusters. The recursive cluster summarization uses Claude Sonnet 4.6, the same summarization model as the CORPUS2SKILL compiler, so the two embed-cluster-summarize pipelines are matched on the summarizer. We use the same embedding model (Qwen3-Embedding-0.6B) as the Dense baseline for consistency; Appendix K reports the encoder-controlled cross in which both CORPUS2SKILL and the baselines are run at either encoder size. At query time, collapsed tree retrieval searches across all hierarchy levels simultaneously, returning the top-5 passages from any level.

Agentic. An LLM agent is given access to three search tools (`search_bm25`, `search_dense`, `search_hybrid`), each returning top-k results. The system prompt instructs the agent to search first, then answer based on accumulated evidence, choosing BM25 for exact terms and product names, dense for conceptual queries, and hybrid as a default strategy. The agent can issue up to 10 rounds of iterative retrieval, reformulating sub-queries based on intermediate results. Evidence is accumulated across all turns; when the agent stops, it produces a structured JSON response containing the answer and curated top-5 references selected from the full evidence pool. For evaluation fairness, context recall is computed over the last 5 retrieved documents (matching the context window of single-shot baselines). This baseline tests whether agentic multi-turn retrieval over traditional search infrastructure can match navigation-based exploration.

I Evaluation Metrics

Lexical and semantic answer-quality metrics. *Token F1* tokenizes both the predicted and gold answers into word tokens and computes the harmonic mean of token-level precision and recall. *BERTScore-F1* (Zhang et al., 2020) computes contextual-embedding similarity F1 between the predicted and gold answers using a RoBERTa-large encoder (Liu et al., 2019) with `rescale_with_baseline=False`; it is more robust to paraphrase than n-gram-overlap metrics. We compute it once per query and report the mean.

LLM-judged answer-quality and grounding metrics. *Factuality*, *Faithfulness*$_{grd}$, *Context Recall*, *Context Precision*, and *Answer Relevance* use an LLM judge (Zheng et al., 2023) following the RAGAS framework (Es et al., 2024), scoring on a 5-point scale normalized to 0–1. Factuality compares the generated answer to the gold answer (independent of retrieved context). Faithfulness$_{grd}$ scores each generated claim against the retrieved context, flagging hallucinations regardless of whether the answer happens to match the gold. Context Recall asks whether the retrieved context covers the gold-answer claims; Context Precision asks whether retrieved chunks are predominantly relevant; Answer Relevance asks whether the generated answer addresses the query. We use Claude Sonnet 4.6 as the judge with greedy decoding and zero-temperature prompts, and apply the same judge model and standardized context window across every method to keep the comparison fair.

Hallucination Rate. We report *Hallucination Rate* as the fraction of queries with Faithfulness$_{grd}$ < 0.6 (i.e. generated content that is not adequately grounded in retrieved context, following the RAGAS convention). This metric is lower-is-better and captures the operational "how often does the system hallucinate?" question that a deployer cares about, rather than a mean over a continuous score.

Context Recall standardization. A critical design decision is the standardization of Context Recall measurement across different retrieval paradigms (single-shot, multi-turn, and navigation-based). For fair comparison, all methods follow the same protocol: (1) collect the last 5 documents or passages retrieved by the method; (2) join them with separator tokens; (3) truncate to a uniform 8,000-character window. The LLM judge receives this standardized context. This ensures that single-shot baselines contribute their top-5 retrieved passages, Agentic contributes the last 5 documents from its multi-turn retrieval, and CORPUS2SKILL contributes the last 5 documents retrieved via `get_document` tool calls.

Cost metrics. Input token count is the total number of tokens sent to the LLM across all turns. Per-query cost is computed using the model's published per-token pricing (input and output tokens combined) including the Anthropic ephemeral-cache discount (cached-read input at $0.1\times$ the base rate, cache-write at $1.25\times$). For single-shot methods, cost reflects one LLM call; for multi-turn methods (Agentic, CORPUS2SKILL), it reflects all turns including tool-call reasoning.

J Dataset Licenses, Intended Use, and Documentation

Licenses. We use two publicly released benchmark datasets. *WixQA* (Cohen et al., 2025) is distributed on Hugging Face[4] under the **MIT License**, which permits unrestricted use, modification, and redistribution with attribution. *RAGBench* (Friel et al., 2024) is distributed on Hugging Face[5] under the **Creative Commons Attribution 4.0 International (CC-BY-4.0)** license, which permits sharing and adaptation for any purpose provided appropriate credit is given. Our own code is released at `https://github.com/dukesun99/Corpus2Skill` under the MIT License.

Consistency with intended use. Both datasets were created as benchmarks for evaluating retrieval-augmented generation systems, which is exactly

[4] `https://huggingface.co/datasets/Wix/WixQA`

[5] `https://huggingface.co/datasets/galileo-ai/ragbench`

how we use them. WixQA was designed to evaluate enterprise RAG pipelines over a customer-support knowledge base; we use its 200 expert-written test queries and 6,221-article corpus for our primary evaluation (Section 5). RAGBench was designed as a multi-domain RAG evaluation benchmark; we use 10 of its 12 subsets for our generalization study (Section 5.3). We do not redistribute modified versions of either dataset. Our derived artifacts (compiled skill hierarchies) are deterministic transformations of the original corpora and are intended solely for research evaluation.

Documentation and coverage. All experiments use English-language data exclusively. WixQA covers the enterprise customer-support domain (Wix.com Help Center articles spanning commerce, site design, billing, SEO, and platform management) and contains 6,221 documents with three QA subsets: 200 expert-written queries, 200 simulated queries, and 6,221 synthetic queries. The 10 RAGBench subsets span five broad domains: medical (CovidQA), legal (CUAD), technical support (TechQA, EManual, DelucionQA), financial (FinQA, TatQA), and open-domain (HAGRID, ExpertQA, ClapNQ), with corpus sizes ranging from 221 to 12,727 documents per subset. Neither dataset contains personally identifiable information; WixQA consists of public help-center articles and RAGBench aggregates previously published academic datasets. We do not make claims about demographic representation, as both benchmarks are topical QA datasets rather than user-facing demographic studies. Full details on corpus statistics and evaluation protocols are provided in Section 5 and Appendix H.

K Statistical Testing and Additional Controls

This appendix gives the statistical protocol behind every win/tie/loss statement and the full results of the controls summarized in Section 5.4 and Section 5.5. Unless stated otherwise, tables use the primary setup: WixQA (6,221 support articles, 200 expert-written queries), Claude Sonnet 4.6 as compile and serving LLM, Qwen3-Embedding-8B as compile encoder, and Token F1 against the gold answer as the metric.

Paired bootstrap protocol. For each dataset we pair CORPUS2SKILL's per-query Token F1 with the best baseline's on the matched queries and resample the paired differences 10,000 times. A dataset is a *win* if the two-sided p-value for a positive mean difference is below 0.05, a *loss* in the reverse direction, and a *tie* otherwise. Per-metric confidence intervals are in the released code, and the same procedure underlies the deltas in Table 9.

Table 8: Encoder cross on WixQA (Token F1, 200 queries per cell): the encoder choice barely moves CORPUS2SKILL and helps the baselines by at most $+0.023$.

Method	0.6B encoder	8B encoder
Dense	0.364	0.378
Hybrid	0.361	0.360
RAPTOR	0.398	0.421
CORPUS2SKILL	0.461	0.458 $\pm$ 0.002 (3 runs)

Table 9: CORPUS2SKILL $-$ RAPTOR, paired bootstrap on WixQA (200 queries, 10k resamples), with compile resources matched at either encoder size. RAPTOR uses the same Sonnet 4.6 summarizer in all rows.

Metric	matched at 0.6B	matched at 8B
Token F1	$+0.064$ ($p<0.0001$)	$+0.035$ ($p<0.0001$)
BERTScore	$+0.025$ ($p<0.0001$)	$+0.020$ ($p<0.0001$)
Context Precision	$+0.181$ ($p<0.0001$)	$+0.085$ ($p<0.0001$)
Context Recall	$+0.084$ ($p=0.007$)	-0.036 ($p=0.22$, tie)

Table 10: Per-dataset Token F1 verdict (paired bootstrap vs. the best baseline per dataset). Δ is the mean per-query paired difference over matched queries and can differ slightly from the difference of column means.

Dataset	**Best bl.**	C2S F1	Bl. F1	Δ	p	**Verdict**
WixQA	RAPTOR	0.456	0.397	$+0.059$	<0.001	win
Emanual	Agentic	0.497	0.455	$+0.061$	<0.001	win
DelucionQA	Dense	0.478	0.439	$+0.034$	0.010	win
ExpertQA	Dense	0.356	0.330	$+0.027$	<0.001	win
ClapNQ	Dense	0.468	0.333	$+0.135$	<0.001	win
CovidQA	Hybrid	0.208	0.216	-0.009	0.33	tie
TechQA	Hybrid	0.396	0.383	$+0.013$	0.14	tie
FinQA	Dense	0.392	0.400	-0.009	0.49	tie
CUAD	BM25	0.272	0.314	-0.052	<0.001	loss
HAGRID	Agentic	0.188	0.356	-0.167	<0.001	loss
TatQA	Agentic	0.240	0.355	-0.116	<0.001	loss

Serve-time variance. Three independent 200-query serve passes over the same compiled WixQA tree give (mean $\pm$ sd) Token F1 0.458 $\pm$ 0.002, BERTScore 0.863 $\pm$ 0.001, Factuality 0.767 $\pm$ 0.008, Context Recall 0.699 $\pm$ 0.010, and Hallucination Rate 0.065 $\pm$ 0.018. The thresholded hallucination rate is the most variable metric.

Cost without prompt caching. Removing the ephemeral-cache discount from the raw token logs raises CORPUS2SKILL from \$0.153 to \$0.302 per query (1.97$\times$) and Agentic from \$0.082 to \$0.098 (1.19$\times$); the ordering is unchanged.