VERSION 1.0 · SEPTEMBER 2026
DOI: 10.5281/zenodo.23260209

CROW: Classifier-Routed Organization of LLM Wikis

Federico Cesarini **Marco Sassarini**

ABSTRACT

LLM wikis have been proposed as an alternative to classical retrieval-augmented generation (RAG): a large language model (LLM) incrementally organizes knowledge into a directory of notes that can be inspected directly and grows with little manual upkeep, at the price of LLM calls whose cost grows with the knowledge base. Many of these calls are spent on decisions rather than on writing: where a new source belongs, which notes it relates to, which notes answer a question. We introduce CROW (Classifier-Routed Organization of LLM Wikis), a decision layer for LLM wikis that delegates these routing and consolidation decisions to a general-purpose typed classifier, such as the recently released Jev, and reserves generative calls for the steps that require writing text. In CROW, ingestion relies on a short sequence of classifier calls and invokes an LLM only to summarize the incoming source, to name new folders and, when an existing note must be updated, to rewrite it, while retrieval uses the classifier to select the relevant notes and leaves to the LLM only the final answer. CROW does not replace the compilation of entity, concept and synthesis pages that gives LLM wikis much of their value. We specify how each decision maps onto classifier primitives, position the design with respect to memory systems for LLM agents, model routing and candidate-then-verify pipelines, and evaluate it on Google's Open Knowledge Format against an LLM that takes the same decisions, on 525 documents. CROW ingested the documents at 0.47 times the cost and 0.48 times the latency, with an LLM cost per document that stayed flat as the wiki grew, and at retrieval found the right note in 89.2% of asks against 61.8%, in about half the time. A cost model built on these measurements keeps an ingest at about $0.006 up to a million documents, so an LLM wiki with CROW can grow into a large, production knowledge store. The same mechanism also works as zero-shot, incremental clustering, in which named categories emerge from a stream of documents without training.

Keywords LLM wikis · Knowledge bases · Retrieval-augmented generation · Typed classifiers · Hierarchical classification · Zero-shot clustering

1 Introduction

LLM wikis have been proposed as an alternative to classical RAG [1, 2]. Instead of retrieving fragments of raw documents at every query, an LLM agent compiles sources into a directory of interlinked notes. This brings two advantages: the knowledge base can be inspected directly as an organized directory rather than as an opaque vector index, and knowledge accumulates across sources instead of being rediscovered at each query. Since the agent maintains the directory through tool calls [3], the knowledge base grows with little manual upkeep.

What LLM wikis lack compared with classical RAG is cost optimization. RAG ingestion depends mainly on the computation of embeddings, while LLM wiki ingestion costs come from LLM calls. Since each of these calls reads the wiki index alongside the new content, the cost of a single ingestion grows with the size of the knowledge base, and, as long as the index is read in full at each call, the cumulative cost of building a large knowledge base grows faster than linearly. Part of this cost is unavoidable, since summaries and updated pages have to be written; part of it is spent on decisions, such as where a source belongs or which pages are worth reading.

With this paper we introduce CROW, a decision layer for LLM wikis that uses a general-purpose typed classifier [4, 5] for these decisions. The underlying observation is that many steps in the lifecycle of an LLM wiki are decisions rather than generations, which is what typed classifiers are built for. CROW is not a complete alternative to an LLM wiki: it covers routing, retrieval and the consolidation of new content into existing notes, but not the compilation of entity, concept and synthesis pages, cross-referencing or periodic maintenance (Section 4).

Our contributions are: (i) a decomposition of ingestion and retrieval in an LLM wiki into atomic decisions and generation steps; (ii) a mapping of each decision onto the primitives of a typed classifier, with explicit escape options and a confidence-gated fallback to an LLM; (iii) a discussion of when such a classifier is expected to help relative to embeddings, cross-encoders and small LLMs; (iv) an implementation on Google's Open Knowledge Format and an evaluation on 525 documents against an LLM that takes the same decisions in the same pipeline; (v) the use of

the same mechanism as zero-shot, incremental clustering, in which the classifier assigns documents and the LLM names new categories; and (vi) an outline, left to future work, of classifier-driven maintenance, in which instruction-defined checks on candidate pairs of notes select the cases an LLM should fix. The evaluation measures the expected reductions in cost and latency: CROW ingested the documents at 0.47 times the cost and 0.48 times the latency of the LLM-decided baseline, its LLM cost per document did not grow with the wiki, and at retrieval it found the right note more often (89.2% against 61.8%) and faster.

Section 2 introduces LLM wikis and typed classifiers, and Section 3 reviews related work. Section 4 describes the lifecycle of an LLM wiki and the scope of CROW, Section 5 presents the architecture, and Section 6 its expected benefits. Section 7 describes the implementation and the evaluation, Section 8 the results, and Section 9 what they imply for large knowledge stores. Section 10 presents CROW as zero-shot clustering and its applications. Section 11 discusses the limitations, Section 12 outlines classifier-driven maintenance, and Section 13 concludes.

2 Background

2.1 LLM wikis

The LLM wiki was introduced by Karpathy in April 2026 as a pattern for building personal knowledge bases with LLM agents [1]. Its motivation is a limitation of RAG: since relevant fragments are retrieved from raw documents at every query, the model rediscovers knowledge each time and nothing accumulates. In an LLM wiki, the model instead compiles sources once into a persistent, interlinked artifact and keeps it current as new material arrives.

The pattern has three layers. *Raw sources* (articles, papers, data files) are immutable and act as the source of truth. The *wiki* is a directory of LLM-generated markdown pages, such as source summaries, entity and concept pages, comparisons and syntheses, which the LLM owns entirely. The *schema* is a configuration document, typically the instruction file of a coding agent, that specifies the structure of the wiki, its conventions and the workflows to follow. Three operations act on these layers. *Ingest* reads a new source, writes a summary page and updates the related entity and concept pages, the index and a log; a single source may touch ten to fifteen pages. *Query* locates the relevant pages, reads them and synthesizes an answer with citations, and valuable answers can be filed back as new pages. *Lint* periodically checks the wiki for contradictions, stale claims, orphan pages, missing cross-references and knowledge gaps.

Navigation relies on two special files: a content-oriented index that lists every page with a one-line summary, which the LLM reads first when answering a question, and a chronological, append-only log. Karpathy reports that the index alone works well at moderate scale (around a hundred sources and hundreds of pages) without embedding infrastructure, and suggests adding a dedicated search engine, such as hybrid lexical and vector search with LLM re-ranking, as the wiki grows. The division of labor is explicit: humans curate sources and ask questions, while the LLM performs the summarizing, cross-referencing and bookkeeping. Every operation, however, is carried out by an LLM agent loop, which is where cost and latency accumulate.

2.2 From task-specific to general-purpose typed classifiers

Classification has traditionally relied on task-specific models, for instance encoders such as BERT fine-tuned on labeled examples [6]: fast and cheap at inference, but dependent on training data and on a separate model for each label set. Zero-shot classifiers relaxed this constraint by scoring text against labels supplied at inference time, for example by framing classification as textual entailment [7]. In LLM applications, a common shortcut has been to prompt a generative model for structured output, which is flexible but slow and expensive for small, repeated decisions. Automated pipelines also need calibration, that is, predicted probabilities that match empirical accuracy [8], since this is what lets software decide when a prediction can be trusted.

General-purpose typed classifiers combine instruction-driven, zero-shot classification with probabilistic outputs intended to be calibrated. An example is Jev, released in early access by TypeSafe AI on 15 September 2026 and described by its vendor as the first model of this kind [4, 5]. Jev takes a *state*, expressed as text or structured data, and a set of typed *questions* written in natural language, and returns typed answers instead of text [5]. A *Choice* selects one option from a defined set of up to 255 and returns a probability for each option; a *Score* rates the state against ordered, descriptive levels; a *Noul*[1] returns the probability that a yes/no statement holds. Choice and Score answers also carry a confidence value derived from the shape of the probability distribution, highest when the mass concentrates on one option; it measures how peaked an answer is, not whether it is correct. The vendor recommends

[1] *Noul* is the vendor's name for its binary primitive.

acting automatically on high confidence, reviewing intermediate cases and escalating to a person or a more capable model on low confidence, with thresholds that scale with the cost of an error, and decomposing complex judgments into atomic questions combined in code. All questions in a request are evaluated in parallel against the same state.

The vendor reports end-to-end latencies of 70 to 500 ms and input pricing of $0.042 per million tokens with free output, against $0.20 to $10 per million input tokens for existing LLMs, and accuracy comparable to LLMs on workflow evaluations written by its own team and scored against the average answers of two large external models; it acknowledges possible bias and presents its speed and cost ratios as likely upper bounds [4]. These figures come from the vendor, and the architecture is undisclosed [4, 9]. The known weaknesses listed in the documentation include literal reading of instructions, poor handling of numbers and dates, lower accuracy on multi-step indirection and on large inputs full of irrelevant detail, a bounded context window (64k tokens per request, of which at most 32k for the state and the longest question [11]), sensitivity to adversarial content, and no text generation [10]. It also describes hierarchical classification, in which a Choice is asked at each node of a tree, such as a taxonomy or a file system, and several candidate paths can be explored in parallel [12]. In the rest of the paper, *classifier* denotes any model with these properties; CROW does not depend on a specific vendor, although we describe it with Jev's primitives.

3 Related work

Memory systems for LLM agents. Mem0 [13] extracts salient facts from conversations, retrieves similar existing memories with embeddings, and asks an LLM to choose among adding, updating, deleting or leaving the memory unchanged. A-Mem [14] builds Zettelkasten-style notes and uses agent-driven decisions to link related notes and to update the descriptions of existing notes as new ones arrive. The note-relevance and consolidation steps of CROW (Section 5.1) address the same problem; the difference is who decides. CROW replaces the LLM decision with a typed classifier, keeps the LLM for rewriting only, and organizes notes in an inspectable folder hierarchy that is also used for routing.

Model routing and cascades. FrugalGPT [15] and RouteLLM [16] reduce cost by answering queries with cheaper models when possible, through a cascade of models or a learned router between a stronger and a weaker model. CROW's confidence-gated fallback follows the same logic, but what is routed is a single decision within a knowledge base rather than a whole query.

Candidate generation and verification. Entity resolution scales pairwise comparison through blocking, a cheap step that restricts the candidates, followed by a more expensive comparison of the candidate pairs [17]. Retrieval pipelines often combine a first-stage retriever, for instance based on sentence embeddings [18], with cross-encoder re-ranking [19], and pairwise judgments of consistency are studied as natural language inference and fact verification [20, 21], for which encoders such as DeBERTa [22], fine-tuned for inference, can run locally. CROW's routing plays the role of blocking, and its note-scoring step is functionally a zero-shot re-ranker. For relevance alone, embeddings and a cross-encoder may well suffice. The case for a typed classifier rests on decisions defined by instructions rather than by similarity or entailment, such as whether new content should modify a note or become a new one, and on per-decision probabilities that can gate a fallback to an LLM. Whether this advantage holds in practice is an empirical question (Section 11).

Tree-organized retrieval. RAPTOR [23] builds a tree of recursive summaries by clustering text chunks and retrieves across its levels. CROW's tree is instead a folder hierarchy that humans can inspect and edit, grown incrementally as sources arrive, and traversed by classification rather than by embedding similarity.

What distinguishes CROW from these works is that it delegates the decisions of an LLM wiki to a general-purpose typed classifier and specifies each of them as a typed question with an explicit fallback.

4 The LLM wiki lifecycle

Following Section 2.1, we group the operations of an LLM wiki into two phases, ingestion and retrieval, and treat lint separately. In both phases the agent loop interleaves decisions, such as where a source belongs or which pages to read, with generation, such as writing a summary or an answer.

Ingestion. An LLM agent reads the new source together with the index, which lists every page with a one-line summary. It writes a summary page, decides which existing entity and concept pages the source affects, updates them together with their cross-references, and records the change in the index and in the log. A single source may touch ten to fifteen pages [1].

Retrieval. The agent reads the index, decides which pages are likely to contain the answer, reads them through tool calls, and writes an answer with citations.

Lint. Periodically, the agent checks the whole wiki for contradictions, stale claims, orphan pages and missing cross-references.

In all three operations, both the decisions and the writing are performed by the most expensive component of the system.

Scope of CROW. CROW assumes that the wiki is organized as a tree of folders, that each note belongs to exactly one folder, and that the index describes every folder and every note with a one-line summary. This is an additional assumption with respect to the original pattern, which organizes pages mainly through the index and through links. CROW redesigns the decisions of ingestion and retrieval: where a new source belongs, which notes it relates to, whether it should be merged into one of them, and which notes to read to answer a question. It does not reproduce the compilation and update of entity, concept and synthesis pages, the maintenance of cross-references, or lint, which are much of what distinguishes an LLM wiki from an archive of summaries. The comparison with the classical workflow in Sections 5 and 6 therefore concerns these routing and consolidation steps only, and Figure 1a shows the LLM-decided baseline of our evaluation, which performs them.

5 The CROW architecture

CROW keeps the notes and the two phases but changes who takes the decisions: the index becomes a set of options for a classifier, and the LLM is called only when text must be written (Figure 1b). We describe each decision with Jev's primitives; any classifier that returns probabilities for a single choice among options and for yes/no statements can implement it. A note consists of a title, a one-line summary, a body and links to its sources, and its index entry is its title and one-line summary. Except for the consolidation step, which reads the single best-matching note in full, every decision is taken on these compact descriptions rather than on full note texts, following the documented drop in accuracy on long inputs full of irrelevant detail [10]. Since thresholds depend on the model, CROW calls a pinned model version rather than an alias that moves with new releases.

5.1 Ingestion

0. **Summarization (optional).** An LLM reads the source and returns, in a single call, the title, one-line summary and body of a candidate note. This is the only generative call that every ingestion pays; for long sources it may account for most of the ingestion cost, in CROW as in a classical LLM wiki. If summarization is disabled, the note stores the source text, its title and one-line summary are taken from the source metadata, and the wiki reduces to a filed archive of sources; the later steps then use as state the source truncated to its title, metadata and opening passage, so that it fits the state budget of a request.

1. **Folder routing.** The state is the candidate note from step 0, or the truncated source when summarization is disabled. This compact state keeps every decision within the context budget and away from the documented drop in accuracy on long inputs full of irrelevant detail [10]. At each folder, a Choice lists its subfolders, each described by its name and one-line summary, plus three explicit options: *Here*, meaning that the note belongs in the current folder rather than in any of its subfolders; *New subfolder*, meaning that it belongs under the current folder but neither in the folder itself nor in any existing subfolder; and *None of these*, meaning that it does not belong under the current folder at all. At the root, which contains every note, *None of these* is omitted, so that an empty wiki starts from *Here* and *New subfolder* and the tree can grow from the first ingestion. Since a note belongs to exactly one folder, a single-label Choice is the appropriate primitive. Descent starts at the root and continues while an existing subfolder wins; it stops when *Here* or *New subfolder* wins, so that a folder without subfolders is still asked whether the note belongs in it or in a new subfolder beneath it. When the path chosen for filing ends in *New subfolder*, an LLM writes the name and one-line summary of the new folder from the candidate note and the names of its sibling folders, and code creates the folder and its index entry. When the confidence at a level falls below a threshold τ_{route}, the classifier keeps the b most probable options instead of one, and each kept path continues independently. Paths that end in *None of these* are discarded. The note is filed at the end of the remaining path with the highest path score, the geometric mean of its edge probabilities as in the vendor's cookbook [12], and the folders along all remaining paths provide candidate notes for step 2. If every path ends in *None of these*, or the best path score is below τ_{path}, the decision falls back to an LLM (Section 5.3).

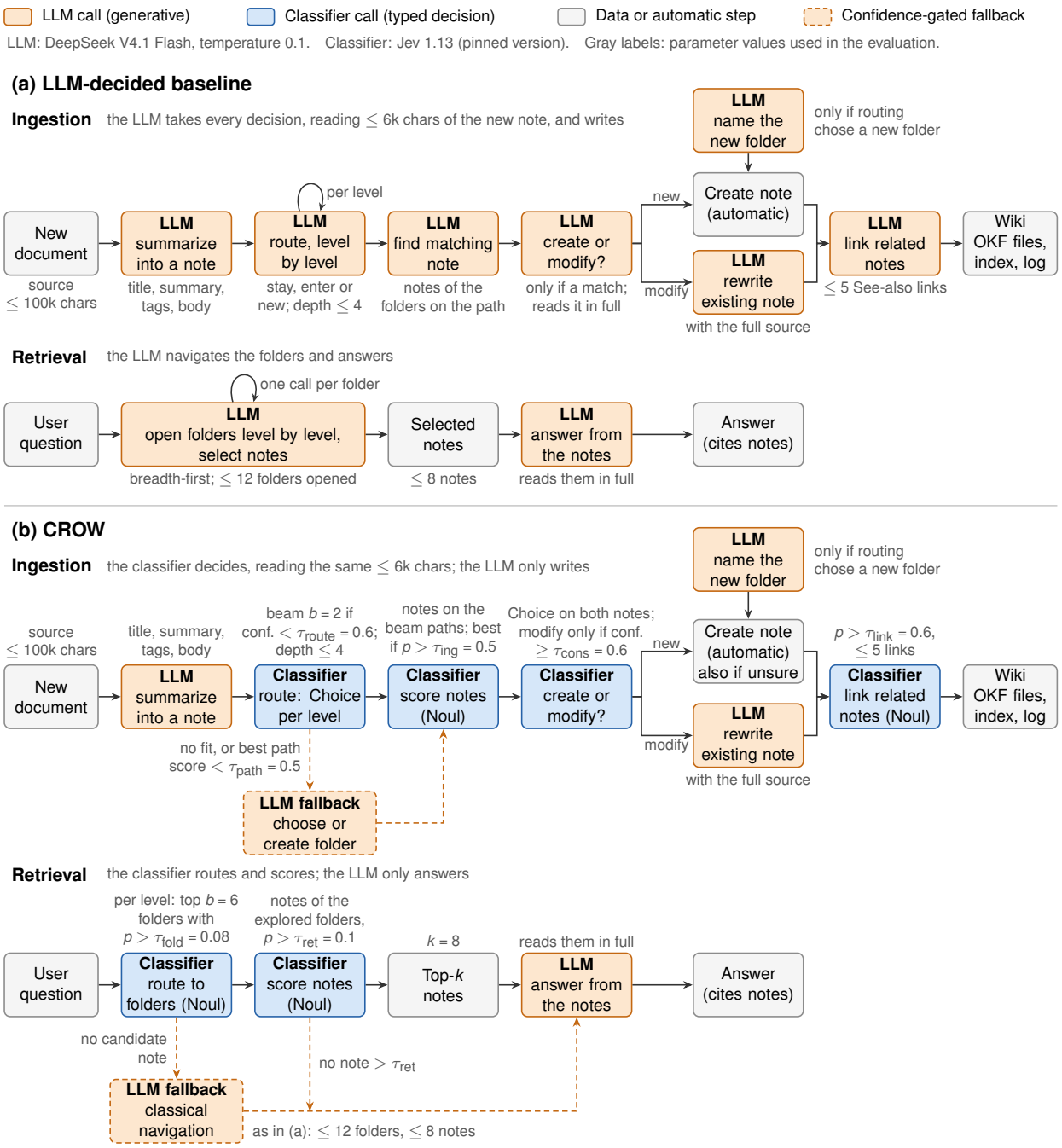


Figure 1. **(a)** LLM-decided baseline, restricted to the routing and filing steps that CROW replaces: every decision is taken by an LLM. **(b)** CROW: routing, relevance, consolidation and linking decisions are delegated to a typed classifier; the LLM is called only to summarize the source, to name a new subfolder, to rewrite an existing note or to generate the final answer. Dashed paths are confidence-gated fallbacks to an LLM (Section 5.3). Gray labels: parameter values used in the evaluation (Table 1).

2. **Note relevance.** The state is again the candidate note, or the truncated source. For each note in the candidate folders, the classifier receives one Noul question, "Does the incoming content concern the same subject as the note titled t, summarized as s?", with the note's title and one-line summary written into the question. All questions are evaluated in parallel against the same state; when the candidate notes do not fit in one request (for Jev, 64k tokens for the state and all questions together [11]), the questions are split across requests, each of which sends the state again. Unlike the probabilities of a single Choice, which are normalized across options and split when several notes are relevant, these probabilities are independent, which allows ranking notes from different folders on a common scale, although notes with vaguer summaries may be scored systematically differently. Notes whose probability exceeds a threshold τ_{ing} are candidates for consolidation.
3. **Consolidation.** If no note exceeds τ_{ing}, a new note is created from the output of step 0 and filed in the selected folder, and its index entry is appended by code; no generation is involved. Otherwise, the state contains the candidate note, or the truncated source, and the full text of the best-scoring note, and a Choice asks whether the incoming content should *modify* that note (same subject, new or updated information) or become a *new note* (related but distinct subject). If the confidence of this Choice is below τ_{cons}, CROW creates a new note, on the assumption that a spurious new note does less harm than a wrong modification. If modification is chosen, an LLM receives the note and the full incoming source, rewrites the note, and returns an updated one-line summary for the index. Other notes above τ_{ing} are left unchanged.

Generation is thus limited to the optional summary, the naming of new folders and the rewriting of modified notes. How much this saves with respect to a classical ingestion depends on how often modification is chosen, which Section 8 measures.

5.2 Retrieval

1. **Folder routing.** The state is the user question. Since the answer may be spread across several folders, CROW asks one Noul per subfolder, "Is this folder likely to contain notes that help answer the question?", instead of a single Choice, whose probabilities would split across the relevant folders. At each level, the folders whose probability exceeds τ_{fold} are explored, up to b of them, and the notes held directly in every explored folder are candidates for the next step.
2. **Note scoring.** For each candidate note, a Noul asks whether the note, described by its title and one-line summary, helps answer the question. These probabilities are independent and can be ranked across folders, with the same caveat as in ingestion.
3. **Answer generation.** The k notes with the highest probability, or fewer if fewer exceed τ_{ret}, are passed in full to an LLM, which answers the question based on them.

Every selection step is handled by the classifier, and generation is reduced to the final answer. Unlike an agent loop, which can decide to read further pages after reading the first ones, this pipeline reads the selected notes once. Questions that require several hops across notes, or that aggregate over many notes, such as listing every note on a topic, may therefore be answered less well, and the one-line summary on which notes are scored becomes a bottleneck; the fallback of Section 5.3 is a partial remedy.

5.3 Fallback and thresholds

Every classifier decision in CROW has an explicit escape: *None of these* and τ_{path} in ingestion routing, τ_{cons} in consolidation, and τ_{fold} and τ_{ret} in retrieval. When ingestion routing falls back, an LLM receives the candidate folders and the incoming content, and either selects one of them or creates a new folder, whose name and one-line summary it writes. When no folder or note clears its threshold in retrieval, the question reverts to classical LLM navigation. Since classifier choices are bounded (Jev accepts up to 255 options), large indexes are traversed one level of the folder tree at a time, following the hierarchical classification pattern, where keeping several candidate paths can recover from an ambiguous early choice [12].

The fallback intercepts uncertain answers only. A classifier that is confidently wrong passes every threshold silently, and the confidence of a Choice measures how concentrated its distribution is, not whether it is calibrated. The thresholds τ_{route}, τ_{path}, τ_{ing}, τ_{cons}, τ_{fold} and τ_{ret}, the beam width b and the number of notes k must therefore be set on a small labeled sample of decisions from the target wiki, following the vendor's guidance that thresholds grow with the cost of an error [5], and set again whenever the model version changes. Section 8.4 shows how we set the retrieval thresholds from the classifier's probabilities.

5.4 Linking

Our implementation adds one step to ingestion. After the note is written, a Noul per nearby note asks whether a reader of the note would want to open the other one, and code writes a See-also link in both notes when the probability exceeds a threshold τ_{link}, up to five links; the LLM never writes links. Linking is a first, restricted instance of the maintenance outlined in Section 12.

6 Expected benefits

The effects below were stated in the preprint of this paper [24] as hypotheses motivated by the design and by vendor-reported figures; Section 8 measures them.

Cost. In CROW the generative model never reads the index, only the source to summarize, the note to rewrite or the top-k notes, so its cost no longer depends on the size of the wiki. The part that does depend on it moves to the classifier, which reads one level of the folder tree at each routing step and the one-line descriptions of the candidate notes. At a vendor-reported input price of $0.042 per million tokens, against $0.20 to $10 for LLMs, and with free output, this share is negligible compared with the generative calls. The overall saving therefore depends on how much of a classical ingestion is spent on decisions rather than on writing, on how much of the index a classical agent can serve from a prompt cache, and on how often CROW chooses to modify a note. The comparison concerns the routing and filing steps only: a classical ingestion that also updates related pages and cross-references does work that CROW does not do.

Latency. A classifier call takes 70 to 500 ms according to the vendor, and all questions in a request are answered in parallel, while a tool-calling loop needs several sequential generations. Routing in CROW needs one call per level of the folder tree, so its latency grows with the depth of the tree rather than with the number of notes. Retrieval latency should then be dominated by answer generation, and creating a new note in an existing folder requires no generation beyond the optional summary.

Consolidation and reliability. The explicit create-or-modify decision may reduce near-duplicate notes, although it could also merge too much content into ever longer notes; which effect prevails is an empirical question. Routing can only select existing folders and notes or an explicit escape option, and low-confidence decisions become visible and can be escalated; confidently wrong decisions do not (Section 5.3).

7 Implementation and evaluation

Implementation. We implemented CROW in the package `llmw2` (open-source repository crow-hq/llm-wiki-v2, used for all the experiments in this paper), which takes its file format from Google's Open Knowledge Format (OKF) [25], not its code. The wiki is a standard OKF bundle: markdown notes with YAML frontmatter, an index file per folder that lists its subfolders and notes with their one-line descriptions, a log, and an immutable copy of every source cited by its note. A wiki is thus plain files, readable without any tool. Usage is recorded in two separate ledgers, one for the LLM and one for the classifier, per step, with tokens, time and cost.

Baseline. The package also implements the same pipeline with an LLM in each decision, one call per decision (Figure 1a): it routes one folder level at a time, picks the matching note from a list of titles and summaries, decides whether to merge, and picks related notes; at retrieval it navigates the tree breadth-first, one LLM call per folder, selects notes and then answers. Every step that writes text or files is shared, with the same prompts. The comparison therefore replaces the LLM with a classifier inside one pipeline, and does not replace an agent loop with a fixed pipeline.

Models. The LLM is DeepSeek V4.1 Flash [26] at temperature 0.1, reached through OpenRouter [27]; the classifier is Jev 1.13, reached through OpenRouter. The LLM calls also passed through a local gateway, since removed, which adds latency to every call: the reported latencies include it, and it weighs more on the baseline, which makes more LLM calls. The runs have not been repeated without it. We price tokens at list prices: $0.30 and $1.20 per million input and output tokens for the LLM [28], $0.042 per million input tokens for the classifier, whose output is free. A classifier token is thus 7 times cheaper than an LLM input token and 29 times cheaper than an LLM output token.

Corpus. 525 markdown documents exported from the first author's personal knowledge base, 1.23 million characters in all (mean 2,339 characters per document, median 1,472); 69.9% are in Italian and 28.0% in English. The corpus is personal and is described here only in aggregate.

Table 1. Parameters of the evaluation. *Retrieval*: the main (long) run. A first retrieval run used the package defaults ($\tau_{\text{fold}} = \tau_{\text{ret}} = 0.5$, $b = 2$, $k = 5$, baseline cap 5 notes); a second run used the calibrated values on a 278-document snapshot with 14 questions (Section 8.4).

Parameter	Ingestion	Retrieval	Role
CROW decisions			
τ_{route}	0.6	–	Routing confidence below which b paths are kept
τ_{path}	0.5	–	Path score below which routing falls back to the LLM
τ_{ing}	0.5	–	Relevance needed to become the consolidation candidate
τ_{cons}	0.6	–	Confidence needed to modify a note
τ_{link}	0.6	–	Relatedness needed for a See-also link
b (beam)	2	6	Paths (ingestion) or folders (retrieval) kept per level
τ_{fold}	–	0.08	Probability needed to explore a folder
τ_{ret}	–	0.1	Probability needed to read a note
k	–	8	Notes passed to the answering LLM
Limits			
`max_depth`	4	4	Deepest folder level
`max_links`	5	–	See-also links added per document
`max_notes` / `max_steps`	–	8 / 12	Notes read and folders opened by the baseline
`state_chars`	6,000	–	Characters of the candidate note read by each decision
`source_chars`	100,000	–	Source characters sent to the LLM
`request_chars`	90,000	90,000	Characters per classifier request (questions are batched)
Models and runs			
LLM	DeepSeek V4.1 Flash, $T = 0.1$		\$0.30 / \$1.20 per million input / output tokens
Classifier	Jev 1.13		\$0.042 per million input tokens, output free
Timeouts, attempts	120 s / 60 s; 3 attempts		LLM / classifier; on network errors and HTTP 408, 429, 500, 502–504, 529
Seed	428636	112232	Document order; question documents
Size	525 documents	20 sizes, 26–525	20 questions asked at every size

Ingestion protocol. Both pipelines ingested the 525 documents in the same random order, in lockstep, starting from empty wikis, with the default thresholds of the package, which had not been calibrated. A network outage interrupted the run halfway; the failed documents were retried in their place in the order, and only final attempts are reported.

Retrieval protocol. Retrieval is measured on 20 wiki sizes, from 26 to 525 documents, each rebuilt from the ingestion order. The LLM wrote one factual question from each of 20 documents filed within the smallest size, so that every question is answerable at every size; the *right note* of a question is the note that cites its document. Each question is asked at every size in each pipeline, on the wiki that pipeline built: 400 asks per pipeline. A *hit* means the right note was among the notes passed to the answering LLM. Rates carry Wilson 95% intervals [29], paired comparisons use the exact McNemar test [30], and growth rates are least-squares slopes with bootstrap intervals [31]. Table 1 lists every parameter.

8 Results

8.1 Ingestion

Table 2 summarizes the results and Figure 2 the cumulative cost of ingestion. CROW ingested the 525 documents for \$2.62 at list price, classifier included, against \$5.62 for the baseline (0.47 times; \$2.35 against \$5.11 as billed), with 0.27 times the LLM calls and 0.32 times the LLM tokens. It was cheaper for 494 of the 525 documents. Its mean latency per document was 18.4 s against 38.6 s (0.48 times), and it was faster for 463 documents.

The reason is visible in where the tokens go. The baseline spent 71.8% of its LLM tokens, and 60.2% of its cost, on decisions: routing, finding a matching note, consolidation and linking. In CROW these decisions took 2,697 classifier calls, at 0.39 s each on average, and cost \$0.19, 7.2% of the total; the LLM mostly writes, and the summary, identical in both pipelines, is the largest item of CROW's cost. Routing fell back to the LLM in 11.6% of ingests, and no classifier call failed.

Table 2. Main results. Ingestion: 525 documents per pipeline. Retrieval: the long run, 20 questions at 20 wiki sizes from 26 to 525 documents, 400 asks per pipeline. Costs at list price in USD unless noted; hit rates with Wilson 95% intervals.

	Baseline	CROW	Ratio
Ingestion			
Cost, total (billed)	$5.62 ($5.11)	$2.62 ($2.35)	0.47
of which classifier	–	$0.19	
LLM calls	2,945	796	0.27
LLM tokens	9.10 M	2.89 M	0.32
Share of LLM tokens spent on decisions	71.8%	4.8%	
Latency per document, mean / 95th percentile	38.6 / 134.1 s	18.4 / 54.3 s	0.48
LLM tokens per document, growth per 100 notes	+1,197	flat	
Retrieval			
Right note found (hit)	61.8% [56.9, 66.4]	89.2% [85.8, 91.9]	
Questions answered	90.5%	98.0%	
Latency per question, mean / 95th percentile	4.88 / 10.03 s	2.61 / 3.93 s	0.53
LLM calls per question	3.27	1.11	0.34
Cost per question (billed)	$0.0029 ($0.0021)	$0.0020 ($0.0013)	0.69

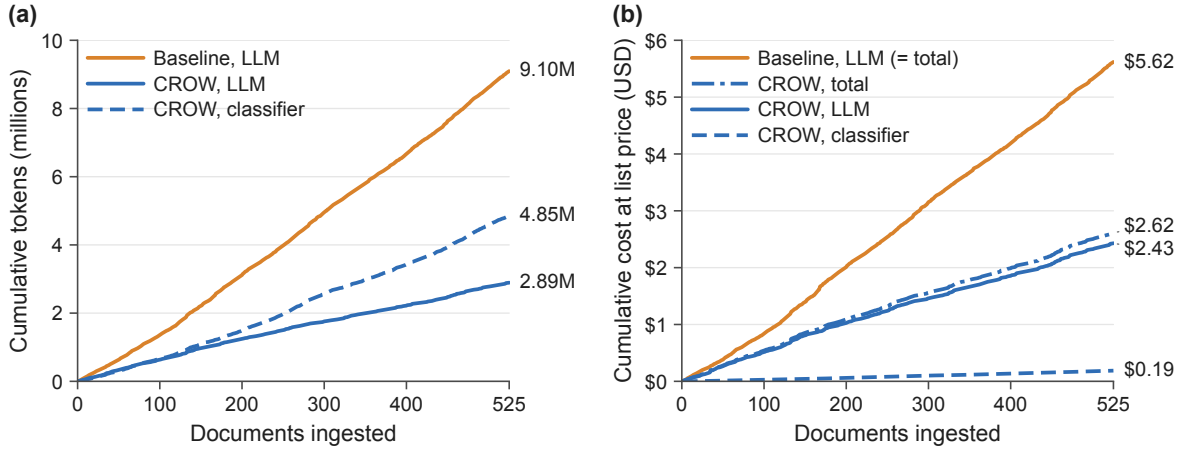


Figure 2. Cumulative cost of ingesting the 525 documents. **(a)** Tokens: 9.10 million LLM tokens for the baseline, against 2.89 million for CROW plus 4.85 million classifier tokens (dashed; input and output, of which only input is billed). **(b)** Cost at list price: $5.62 against $2.62, of which the classifier is $0.19.

8.2 Cost as the wiki grows

Figure 3 shows how the tokens per ingested document change as the wiki grows. The baseline's LLM tokens per document grew by 1,197 for every 100 notes already in the wiki (95% CI 772 to 1,634), driven by its decisions (1,346 per 100 notes), because each decision prompt lists the folders or notes to choose from, and these lists lengthen as the wiki fills. In CROW, the same lists are read by the classifier: its tokens grow at a similar rate (1,067 per 100 notes), but at the classifier's price this adds only $0.000038 per document for every 100 notes, against $0.000513 for the baseline. CROW's LLM tokens per document even fell slightly, because new folders, whose naming costs LLM tokens, became rarer as the tree filled (a new folder in 71% of ingests at the start, 19% at the end) and because later documents happened to be shorter; accounting for both, its LLM cost per document is flat. Latency per document did not grow in either pipeline.

8.3 The resulting wikis

Both wikis are valid OKF bundles, with no broken link and no conformance problem. They differ in shape. CROW built a finer tree: 178 folders (41 at the top level) for 494 notes, against 75 folders (22) for 485 notes, and 53.4% of its folders hold a single note, against 36.0%. It merged new content into existing notes 31 times, against 41, following its rule of creating a new note when unsure, and added 526 See-also links against 909. Part of this finer granularity

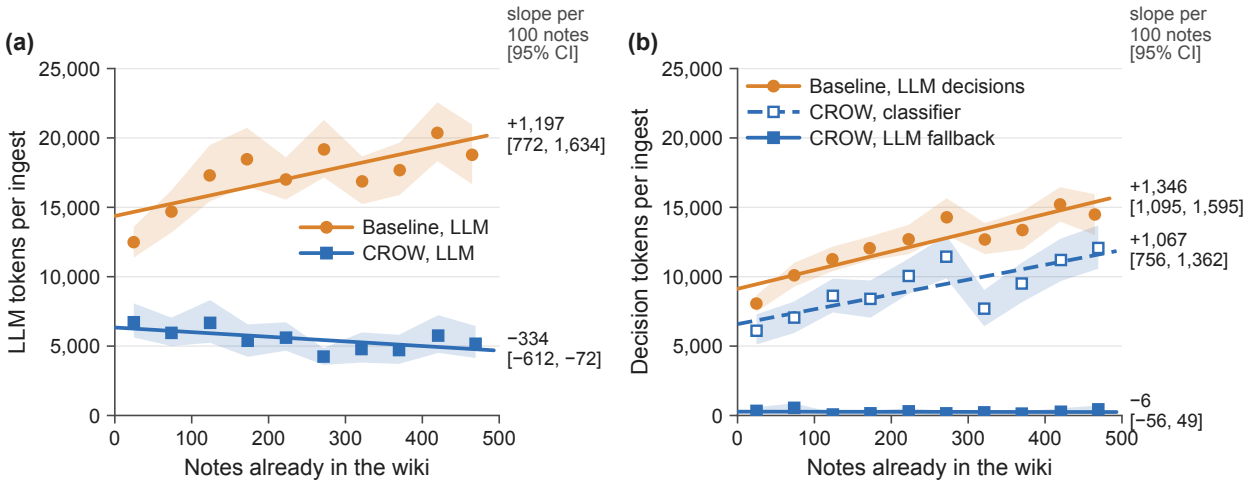


Figure 3. Tokens per ingested document as the wiki grows (binned means with 95% bands, least-squares lines, slope per 100 notes in the margin). **(a)** LLM tokens: the baseline grows by 1,197 per 100 notes; CROW's slight decline disappears once document length and new folders are accounted for (−91, interval −245 to 72), so its LLM cost is flat. **(b)** Decision tokens: the baseline's LLM decisions grow by 1,346 per 100 notes; in CROW the growth is in classifier tokens (dashed), priced 7 to 29 times lower, and its only LLM decision, the routing fallback, stays flat.

Table 3. Retrieval before and after setting the thresholds; the baseline reads at most k notes too. *Right note found*: the right note was among the notes passed to the answering LLM. The first two runs share a 278-document snapshot and 14 questions (8 and 20 sizes up to 278 documents; 110 and 280 asks per pipeline); the full run asks 20 new questions at 20 sizes from 26 to 525 documents (400 asks, Table 2).

	Default	Calibrated	
	thresholds	278 documents	full run
τ_{fold} / τ_{ret}; beam b, notes k	0.5 / 0.5; 2, 5	0.08 / 0.1; 6, 8	0.08 / 0.1; 6, 8
Right note found, baseline	78.2%	71.8%	61.8%
Right note found, CROW	60.0%	97.5%	89.2%
CROW falling back to LLM navigation	79.1%	0.4%	4.5%
LLM calls per question, baseline / CROW	3.01 / 2.85	3.08 / 1.00	3.27 / 1.11
Latency per question, baseline / CROW	3.76 / 8.01 s	9.34 / 3.26 s	4.88 / 2.61 s

reflects the uncalibrated ingestion thresholds: the baseline's routing prompt carries rules against opening folders for single documents, while the classifier's *New subfolder* option is described by one sentence. Calibrating ingestion, and merging small folders (Section 12), are the natural next steps.

8.4 Retrieval

Setting the thresholds. Table 3 shows the three runs over which the retrieval thresholds were set. With the package's defaults, CROW fell back to LLM navigation in most questions and found the right note less often than the baseline. The classifier's probabilities explained it: the right note usually ranked first among all candidates (70.4% of asks, 95.4% within the first two), but its probability was low in absolute terms (median 0.17), so a threshold of 0.5 discarded it before the ranking could act. We therefore set τ_{fold} and τ_{ret} at or just below the lowest probabilities that right folders and notes received, widened the beam, raised k, and gave the baseline the same cap of 8 notes. The thresholds then only discard clear non-candidates and leave the selection to the ranking.

Results. The full run confirms the calibrated values on the whole corpus (Table 3, Figure 4 and Table 2): CROW found the right note in 89.2% of asks against 61.8% for the baseline (exact McNemar $p = 1.2 \times 10^{-21}$), doing better on 10 questions, worse on 2 and the same on 8, and it answered almost every question with about one LLM call, the answer, in about half the time. Its accuracy held as the wiki grew twentyfold, from 26 to 525 documents: 90% at both the first and the last size, while the baseline fell from 75% to 55%. The baseline's misses were mostly wrong selections: 31.8% of its answers were written without the right note, against 8.9% for CROW.

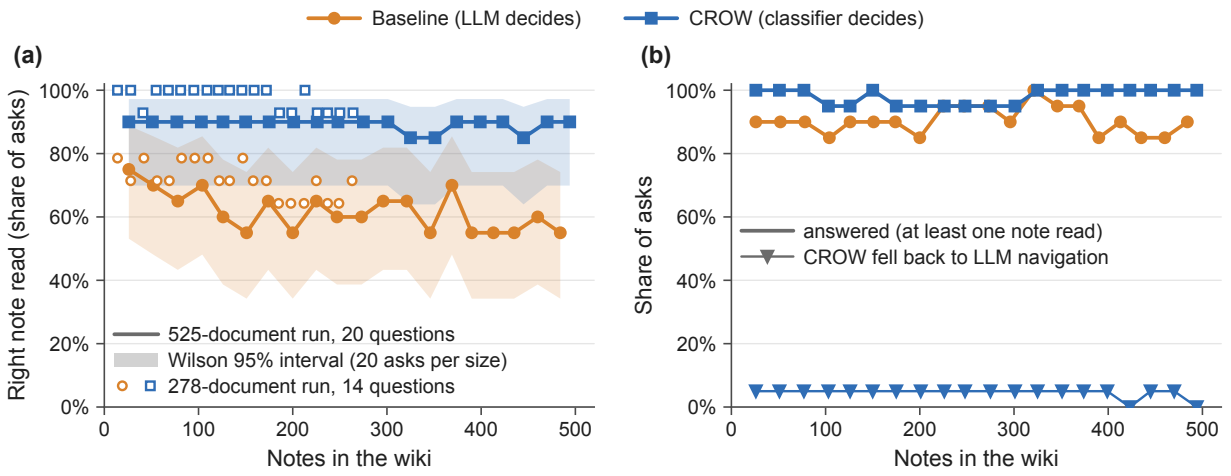


Figure 4. Retrieval as the wiki grows, 20 questions asked at each of 20 sizes from 26 to 525 documents. **(a)** Share of asks in which the right note was read, with Wilson 95% bands: 89.2% for CROW and 61.8% for the baseline over the run; CROW holds at 85–90% at every size. Hollow markers: the calibration run on a 278-document snapshot (97.5% against 71.8%). **(b)** Questions answered, and CROW's fallbacks to LLM navigation (4.5%).

Two readings complete the picture. CROW reads more notes per question (4.03 against 1.65, within the same cap of 8), and counting only the first note read it still leads, 60.8% against 48.8%. And the thresholds were set on questions drawn from some of the same documents: on the 9 questions from documents never used for calibration, CROW found the right note in 77.8% of asks against 69.4%, a smaller lead that nine questions cannot establish statistically.

8.5 The expected benefits, measured

The hypotheses of Section 6 hold on this corpus. *Cost*: decisions took 71.8% of the baseline's LLM tokens, so moving them to the classifier halved the cost of ingestion; the prompt cache did not change the comparison (billed LLM costs were 0.91 and 0.89 times list price); and modification was chosen rarely in both pipelines. *Latency*: a classifier call took 0.39 s, within the vendor's range, ingestion was twice as fast, and retrieval needed about one LLM call, the answer. *Consolidation and reliability*: with a new note whenever unsure, CROW merged less rather than over-merging, routing always ended in an existing folder or an explicit option, and uncertain routing decisions became visible and went to the LLM in 11.6% of ingests.

9 Toward large knowledge stores

The measurements point to a property that matters most in production: in CROW, the cost that grows with the knowledge base is carried by the classifier, not by the LLM. The LLM cost of adding a document depends on the document, mainly on its length, and not on how many documents are already stored; the decisions, whose inputs lengthen as the store grows, are paid at classifier prices; and retrieval kept its accuracy across a twentyfold growth, with a mean latency that stayed below 3.3 s. An LLM wiki with CROW can therefore hold large amounts of data.

To see how far, we built a cost model from the measured per-item costs (tokens per listed folder, per scored note and per call; the model reproduces the measured decision costs to within 4%) and extended it beyond 525 documents (Figure 5). The costs that follow are projections, not measurements: they assume the list prices of Section 7, per-item token costs that do not change with scale and, for a bounded tree, folder splitting, which the package does not yet implement. If the tree keeps widening as it did in our run, decision prompts grow without bound; this scenario is not realistic for either pipeline, since CROW's root folder would exceed the classifier's 255 options at about 15,000 documents. A production store keeps its tree bounded instead, by splitting folders that grow too large. With folders capped at the largest sizes we measured (41 subfolders and 40 notes), the tree stays about three levels deep at a million documents, and a CROW ingest would cost $0.0052 at 1,000 documents, $0.0055 at 10,000, $0.0057 at 100,000 and $0.0059 at one million, of which $0.0012 is classifier. Building a million-document wiki would cost about $5,800 with CROW and $19,800 with the baseline.

Capacity is not a constraint either. The classifier's binding limit is the 255 options of a Choice, which allows over 250 subfolders per folder; with the package's maximum depth of four levels and folders as full as the largest

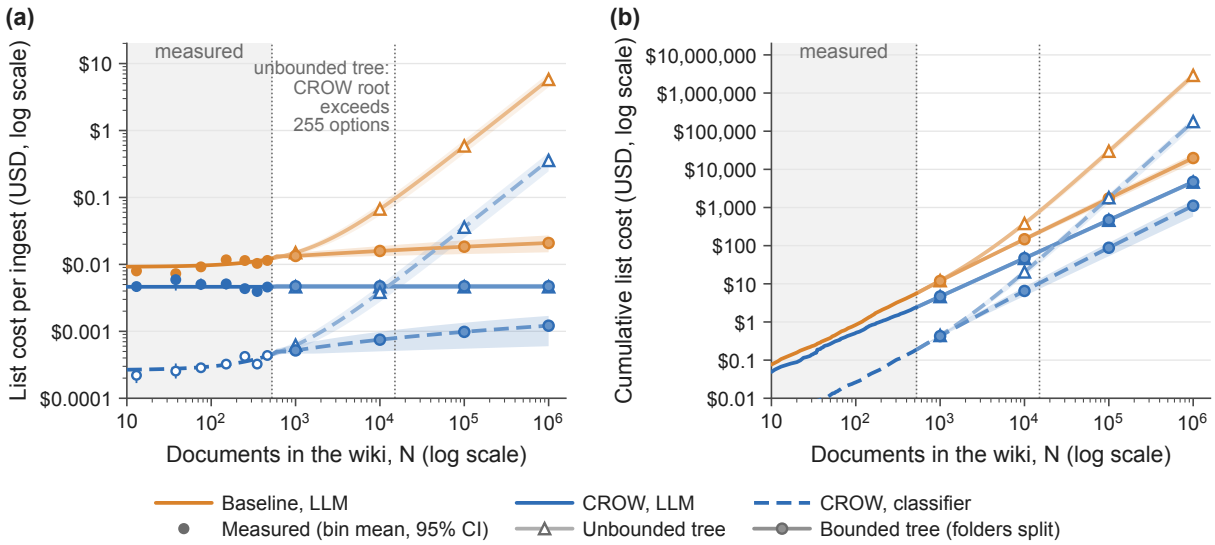


Figure 5. Projected list cost of ingestion beyond the 525 measured documents (gray), log-log. Circles: a tree kept bounded by splitting folders at 41 subfolders and 40 notes; triangles: the measured growth continued without bound, not realizable at large sizes. **(a)** Cost per ingested document: in a bounded tree, a CROW ingest stays near $0.006 up to one million documents (LLM solid, classifier dashed), against $0.021 for the baseline. **(b)** Cumulative cost of building the wiki: about $5,800 against $19,800 at one million documents.

we measured, a tree holds about 10^8 notes. Storage is plain files, about 6.3 kB per document, or 6.3 GB for a million documents. Two engineering steps separate the current package from such a store: folder splitting, to keep the tree bounded, and an indexed store in place of reading the whole wiki from disk at every operation. Retrieval accuracy at that scale remains to be measured.

10 CROW as zero-shot clustering

CROW's ingestion is not specific to wikis. Stripped of notes and folders, it is a zero-shot, incremental clustering procedure: each incoming item is either assigned to an existing cluster or opens a new one, and every cluster has a name and a description. The classifier assigns items, by reading the descriptions of the clusters rather than by comparing vectors, and the LLM writes the name and description of each new cluster. Nothing is trained, no embedding space is needed, the number of clusters is not fixed in advance, and the clustering is never recomputed: it grows one item at a time, and a new cluster can receive items as soon as it exists.

What is new is the split between the two roles. Embedding clustering and topic models group documents by their position in a vector space, describe the groups at best with keywords, and are usually refitted as data arrive [32]. LLM-based topic modeling produces named, described topics, but an LLM takes every assignment decision [33], and with it the cost that grows with the collection. In CROW the classifier clusters and the LLM names. Three properties follow:

- **Readable, hierarchical clusters.** Every cluster has a written description, and subclusters form as subfolders, so the result is a taxonomy that people can read rather than a list of numbered groups.
- **Editable without retraining.** A cluster is defined by the text the classifier reads, so editing its description changes how later items are assigned.
- **A novelty test at every level.** *New subfolder* and *None of these* are explicit options with probabilities, so each item receives a decision on whether any existing cluster fits, the role that first-story detection plays in topic detection and tracking [34]; uncertain items can go to a person instead of being forced into a cluster.

Our ingestion run is, in effect, such a clustering: in a single pass over 525 documents, with no training and no categories defined in advance, CROW organized them into 178 named and described categories up to four levels deep, while its LLM cost per document stayed flat as the categories multiplied.

Fixed trunk, emergent leaves. Many domains prescribe a standard taxonomy, such as patent classes, fields of science or procurement codes, where a category created automatically at a standard level would be an error. There, *New*

Table 4. Applications of CROW as zero-shot, incremental clustering. None has been evaluated yet.

Field	Stream of items	Categories that emerge
Technology transfer	Invention disclosures, research results	Technology areas and new fields of research
Research–industry matching	Technology offers, company requests	Shared areas where offers meet needs
Patent landscaping	Patent filings	Subtechnologies below the official classes
Research funding	Proposals, projects	New topics below fixed fields of science
Customer support	Tickets, chats	Emerging types of issue
News monitoring	Articles, reports	New storylines
Systematic reviews	Search results	Subtopics of the evidence map
Public procurement	Calls for tenders	A buyer's segments below the official codes

subfolder is disabled at the standard levels and allowed only below them, or turned into a proposal that a curator accepts or rejects; the LLM-written description is what the curator reviews. The European Union's EuroSciVoc taxonomy of fields of science already works this way: its top levels follow a standard classification, while its lower levels grow with emerging topics found in project content, under the validation of its editors [35].

Technology transfer. Technology transfer is a natural first application. A transfer office receives a stream of invention disclosures, publications and project results, and groups them into technology areas to assign case managers and to market related intellectual property together; the areas must follow research into new fields. With CROW, each disclosure is routed down the office's tree of areas, *New subfolder* proposes a new area with a name and description that can be reused in marketing material, a Noul per existing case flags related disclosures for bundling, and uncertain cases go to the case manager. Language models are already being applied to predict the commercial potential of research [36]. A second task is matching research with industry: technology offers and company requests arrive as two streams, and text mining has been used to trace university knowledge in company documents [37]. Routed into one tree, each new offer lands next to the requests it may satisfy, where a Noul per request can rank the fit, and a request that opens a new folder marks a need that no offer yet covers. A third task is patent landscaping, in which analysts group the patents of a field into subtechnologies, work that takes time and resources [38]: the official classification is the fixed trunk, and the subtechnologies of a landscape are emergent leaves kept current as new filings arrive.

Other fields. The same pattern applies wherever a stream of documents must be organized into categories that change over time (Table 4). In customer support, *New subfolder* signals an emerging type of issue, with a description a support lead can read at once. In news and intelligence monitoring, new folders are new storylines. In research funding, new topics grow below fixed fields of science, as in EuroSciVoc. In public procurement, calls for tenders are already classified zero-shot into the EU's Common Procurement Vocabulary from its code descriptions alone [39], which is CROW's routing with *New subfolder* disabled; enabling it below the codes would add a buyer's own segments. In each field the thresholds must be calibrated on a labeled sample, and the rate at which new categories are opened must be tuned to the domain.

11 Limitations

Our evaluation covers one corpus of 525 short personal documents, mostly in Italian, one ingestion order, one LLM and one classifier version; other corpora, languages and models may give different results. It compares CROW with an LLM that takes the same decisions in the same pipeline, which separates the effect of replacing an LLM with a classifier from that of replacing an agent loop with a fixed pipeline. The other comparisons remain to be made: the same pipeline with a small LLM that returns a single constrained token, embeddings with a cross-encoder, and, for question answering, the agent loop of the classical wiki and the hybrid search suggested for it at larger scale [1], including multi-hop and aggregation questions. The ingestion thresholds were not calibrated, and ingestion should also be judged by the resulting wiki, for instance by the rate of misfiled and near-duplicate notes, which we did not label. The retrieval thresholds were set after a first run, on questions from some of the documents later evaluated, and a hit means that the right note was read, not that the answer was correct. Beyond 525 documents, costs come from a model.

The documented weaknesses of the classifier [10] matter beyond long inputs: poor handling of numbers and dates affects the decision to update a note with newer information, and lower accuracy on multi-step indirection affects

questions whose answer lies in an unexpected folder. Since routing, relevance and consolidation are taken in sequence, their errors compound, and in the present design a misfiled note stays where it is, where later retrievals may miss it. Sources are ingested and merged automatically, and content written to manipulate the classifier, which the vendor lists among its weaknesses, could misroute sources or trigger modifications of existing notes; sources that are not trusted should not modify notes without review.

CROW modifies only the selected note, whereas a classical ingestion may revise many pages and cross-references at once. The compilation of entity, concept and synthesis pages, the merging of existing notes, periodic maintenance and the splitting of folders that grow too large remain open.

12 Outlook: classifier-driven maintenance

The same division of labor could extend to maintenance. After a note is created or modified, candidate notes obtained by blocking, such as the notes in its folder, in the other folders its routing reached, linked notes or embedding neighbors, could be checked with atomic, instruction-defined questions: whether a link from one note to the other would help a reader, whether the new note contains information that belongs in the other, or whether the two notes duplicate each other. Code would apply links, and an LLM would handle only the flagged pairs. The expected advantage over embeddings and entailment models is the ability to ask questions defined by instructions rather than by similarity or entailment; the expected advantage over an LLM is a lower cost and latency per candidate pair at equal candidate sets, a constant factor rather than a change in complexity. The linking step of Section 5.4 is a first instance, and the same checks on pairs of folders could merge small folders and split large ones, keeping the tree bounded as Section 9 assumes.

Three conditions would have to hold. The total cost would be dominated by the flagged pairs, false positives included, since each of them costs an LLM call on two full notes, so the precision of each check must be measured. Checks for contradictions and outdated claims need the full text of the notes and, ideally, claim-level provenance, since one-line summaries do not contain the claims and a more recent source is not necessarily a correct one. And automatic changes driven by untrusted sources must be limited. We leave the design and evaluation of these checks to future work.

13 Conclusion

CROW reorganizes the routing and consolidation steps of the LLM wiki lifecycle around a simple division of labor: typed classifiers take the decisions, LLMs write the text. It is a decision layer rather than a complete LLM wiki: it keeps an organized knowledge base that grows on its own, while confining generation to summarizing sources, rewriting notes and writing answers, and it leaves the compilation of entity and concept pages and maintenance to future work. On 525 documents it reduced the cost and latency of ingestion by about half, kept the LLM cost of each new document flat as the wiki grew, and found the right note at retrieval more often and faster than an LLM taking the same decisions. Because the part of the cost that grows with the knowledge base is paid at classifier prices, an LLM wiki with CROW can grow into a large, production knowledge store, and the same division of labor turns ingestion into zero-shot clustering of any stream of documents. Whether it keeps its accuracy against embeddings and cross-encoders, and at larger scale, is the main question for further evaluation.

Code and data availability

The implementation, with both pipelines and the prompts, is the package `llmw2` of the repository llm-wiki-v2, which takes its wiki format from Google's Open Knowledge Format [25], not its code; all the experiments in this paper were run with it, and the benchmark harnesses used for this paper are in the repository's history, at commit f51e926. The repository is public at https://github.com/crow-hq/llm-wiki-v2; the analysis scripts that compute every number and figure in this paper from the saved run files will be released with it. The corpus and the run files are not released, since they contain the first author's personal notes.

Declaration of interests

The authors declare that they have no competing interests. They have no financial, employment, advisory or other relationship with TypeSafe AI, the developer of Jev, or with the providers of the language model and the routing service used in the evaluation (DeepSeek and OpenRouter).

References

[1] A. Karpathy. LLM Wiki: a pattern for building personal knowledge bases using LLMs. GitHub Gist, April 2026. https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f.

[2] P. Lewis, E. Perez, A. Piktus, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in NeurIPS*, 2020.

[3] T. Schick, J. Dwivedi-Yu, R. Dessì, et al. Toolformer: Language models can teach themselves to use tools. In *Advances in NeurIPS*, 2023.

[4] D. Almeida. Introducing System One models & Jev. TypeSafe AI Blog, September 2026. https://typesafe.ai/blog/introducing-system-one-models-and-jev.

[5] TypeSafe AI. Documentation: introduction, primitives and confidence. 2026. https://docs.typesafe.ai.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.

[7] W. Yin, J. Hay, and D. Roth. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *Proceedings of EMNLP-IJCNLP*, 2019.

[8] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *Proceedings of ICML*, 2017.

[9] S. Raschka. It's easy to dismiss Jev as just a classifier. Blog post, September 2026. https://sebastianraschka.com/blog/2026/jev-classification-generalization.html.

[10] TypeSafe AI. Jev 1.13 jaggedness. Documentation, September 2026. https://docs.typesafe.ai/model-jaggedness/jev-1.13.

[11] TypeSafe AI. Models. Documentation, 2026. https://docs.typesafe.ai/models.

[12] TypeSafe AI. Hierarchical classification (cookbook). Documentation, 2026. https://docs.typesafe.ai/cookbooks/hierarchical_classification.

[13] P. Chhikara, D. Khant, S. Aryan, T. Singh, and D. Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. arXiv:2504.19413, 2025.

[14] W. Xu, K. Mei, H. Gao, J. Tan, Z. Liang, and Y. Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in NeurIPS*, 2025.

[15] L. Chen, M. Zaharia, and J. Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024.

[16] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica. RouteLLM: Learning to route LLMs from preference data. In *Proceedings of ICLR*, 2025.

[17] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys*, 53(2):31:1–31:42, 2020.

[18] N. Reimers and I. Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of EMNLP-IJCNLP*, 2019.

[19] R. Nogueira and K. Cho. Passage re-ranking with BERT. arXiv:1901.04085, 2019.

[20] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of EMNLP*, pages 632–642, 2015.

[21] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal. FEVER: a large-scale dataset for fact extraction and VERification. In *Proceedings of NAACL-HLT*, pages 809–819, 2018.

[22] P. He, X. Liu, J. Gao, and W. Chen. DeBERTa: Decoding-enhanced BERT with disentangled attention. In *Proceedings of ICLR*, 2021.

[23] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *Proceedings of ICLR*, 2024.

[24] F. Cesarini and M. Sassarini. CROW: Classifier-routed organization of LLM wikis. Preprint, version 0.1.1, Zenodo, September 2026. https://doi.org/10.5281/zenodo.22900601.

[25] Google LLC. Open Knowledge Format (OKF), version 0.2. GitHub repository, August 2026. https://github.com/GoogleCloudPlatform/open-knowledge-format.

[26] DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. arXiv:2609.19969, 2026.

[27] OpenRouter. Provider routing. Documentation, 2026. https://openrouter.ai/docs/guides/routing/provider-selection.

[28] OpenRouter. DeepSeek V4.1 Flash. Model page, September 2026 (prices accessed September 2026). https://openrouter.ai/deepseek/deepseek-v4.1-flash.

[29] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[30] Q. McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[31] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993.

[32] M. Grootendorst. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. arXiv:2203.05794, 2022.

[33] C. M. Pham, A. Hoyle, S. Sun, P. Resnik, and M. Iyyer. TopicGPT: A prompt-based topic modeling framework. In *Proceedings of NAACL-HLT*, pages 2956–2984, 2024.

[34] J. Allan, J. Carbonell, G. Doddington, J. Yamron, and Y. Yang. Topic detection and tracking pilot study: Final report. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, pages 194–218, 1998.

[35] CORDIS, Publications Office of the European Union. EuroSciVoc. Web page, accessed September 2026. https://cordis.europa.eu/about/euroscivoc.

[36] R. Masclans, S. Hasan, and W. M. Cohen. Measuring the commercial potential of science. NBER Working Paper 32262, 2024.

[37] S. L. Woltmann and L. Alkærsig. Tracing university–industry knowledge transfer through a text mining approach. *Scientometrics*, 117(1):449–472, 2018.

[38] A. Trippe. *Guidelines for Preparing Patent Landscape Reports*. World Intellectual Property Organization, 2015. https://www.wipo.int/edocs/pubdocs/en/wipo_pub_946.pdf.

[39] F. Moiraghi, M. Palmonari, D. Allavena, and F. Morando. Zero-shot hierarchical classification on the Common Procurement Vocabulary taxonomy. arXiv:2405.09983, 2024.